A METHOD OF TRANSFERRING AN ACTIVE APPLICATION FROM A SENDER TO A RECIPIENT

BACKGROUND AND FIELD OF THE INVENTION

This invention relates to a method of transferring an active application from a sender to a recipient.

Many user tasks performed on a computer span across multiple log-in sessions. However, the computing process, that is to say the combination of data, program module(s) and current execution state, that implements a user task typically does not live beyond a log-in session. This forces the user to save out the data, from which a new computing process will be constructed later to attempt to resume the user task, possibly on another machine, and possibly for another user. As a result, an application program either has to be written to save out all pertinent data, which is non-trivial, or, more often than not, the user has to put up with losing certain states of the original computing process. An example is Web surfing, where the navigation history is lost even though individual Web pages can be saved.

It is an object of the invention to address this problem.

SUMMARY OF THE INVENTION

According to the invention in a first aspect, there is provided a method of transferring an active application from a sender to a recipient, the method comprising the steps of:

(a) creating a hibernaculum of a process containing some or all of the program modules, data and execution state of the active application,

(b) sending the hibernaculum to a location for retrieval by the recipient; and

(c) reconstructing the active application from the retrieved hibernaculum.

Preferably the location is the mailbox of the recipient or a personal information manager of the recipient and hibernaculum may be sent as an attachment to an e-mail sent to the recipient's mailbox. Prior to step (c) the hibernaculum may be transferred by the recipient to a further location and opened from the further location and the further location may be a calendar or scheduler.

Preferably, the hibernaculum is sent to a storage location remote from the recipient and a reference to the storage location may be sent to a mailbox or a personal information manager of the recipient, the storage location being most preferably on a World-wide Web server with the reference being a URL link. The reference may be transferred by the recipient to a further mailbox or personal information manager and opened from there.

According to the invention in a second aspect, there is provided a method of transferring an active application from a sender to a recipient, one being an initiating party and the other being a target, the method comprising the steps of:

(a) specifying the address of the target that is to participate in the transfer with the initiating party;

(b) creating a hibernaculum of a process containing some or all of the program modules, data and execution state of the active application;

(c) sending the hibernaculum to the recipient;

(d) reconstructing the active application from the hibernaculum.

Either the sender or the recipient may initiate the transfer.

"Sender" and "Recipient" are defined broadly. For example, the sender and recipient may be different machines, different users or the same entity spaced in time.

Apparatus for performing one or more of the methods described above is also envisaged within the scope of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Fig.1 is a general schematic model of an operating environment of a computing system,

Fig.2 schematically illustrates a process life-cycle and operations that may be performed on a process,

Fig.3 is a flow-chart illustrating the Hibernaculum Construct operation,

Fig.4 is a flow-chart illustrating the Assimilate operation, and

Fig.5 is a flow-chart illustrating the Mutate operation.

Fig.6 is a view of a computer screen illustrating the task manager of the embodiment of the invention together with an active application.

Fig.7 shows use on the Send command of the task manager illustrated in Fig. 6.

Fig.8 shows the computer screen of a recipient of the file sent to an E-mail application using the Send command of Fig. 7.

Fig. 9 shows the active application reconstructed on the recipient's machine.

Fig. 10 is a view similar to Fig. 8 showing the file sent to a Calendar application.

DETAILED DESCRIPTION OF THE PREFFERRED EMBODIMENTS

The preferred embodiment of the invention uses process suspension and migration techniques discussed in the applicant's copending PCT applications PCT/SG98/00102 and PCT/SG99/00018, the contents of which are incorporated herein by reference. In order to provide background to the present invention, however, description of these techniques will first be given, after which the application of these techniques in the embodiment of the present invention will be described.

In this specification the following terms will be used with the following meaning:

"First class entity": an object that can be manipulated directly.

"Process": a combination of data, program module(s) and current execution state.
"Execution state": the values and contents of transient parameters such as the contents of registers, frames, counters, look-up tables and the like.
"Hibernaculum": a data type used to store a suspended process.

Figure 1 shows the general model of a computing system. An application program 30 comprises data 10 and program modules 20. The operating system 60, also known as the virtual machine, executes the application 30 by carrying out the instructions in the program modules 20, which might cause the data 10 to be changed. The execution is effected by controlling the hardware of the underlying machine 70. The status of the execution, together with the data and results that the operating system 60 maintains for the application 30, form its execution state 40.

Such a model is general to any computing system. It should be noted here that the present invention starts from the realisation that all the information pertaining to the application at any time is completely captured by the data 10, program modules 20 and execution state 40, known collectively as the process 50 of the application 30.

The process 50 can have one or more threads of execution at the same time. Each thread executes the code of a single program module at any given time. Associated with the thread is a current context frame, which includes the following components:

- A set of registers
- A program counter, which contains the address of the next instruction to be executed
- Local variables of the module
- Input and output parameters of the module
- Temporary results of the module

In any module A, the thread could encounter an instruction to invoke another module B. In response, the program counter in the current frame is incremented, then a new context frame is created for the thread before it switches to executing module B. Upon

completing module B, the new context frame is discarded. Following that, the thread reverts to the previous frame, and resumes execution of the original module A at the instruction indicated by the program counter, i.e., the instruction immediately after the module invocation. Since module B could invoke another module, which in turn could invoke some other module and so on, the number of frames belonging to a thread may grow and reduce with module invocations and completions. However, the current frame of a thread at any given time is always the one that was created last. For this reason, the context frames of a thread are typically stored in a stack with new frames being pushed on and popped from the top. The context frames of a thread form its execution state, and the state of all the threads within the process 50 constitute its execution state 40 in Fig.1.

The data 10 and program modules 20 are shared among all threads. The data area is preferably implemented as a heap, though this is not essential. The locations of the data 10 and program modules 20 are summarized in a symbol table. Each entry in the table gives the name of a datum or a program module, its starting location in the address space, its size, and possibly other descriptors. Instead of having a single symbol table, each process may alternatively maintain two symbol tables, one for data alone and the other for program modules only, or the process could maintain no symbol table at all.

It is preferred that the data and program code of a process are stored in a heap and a program area respectively and are shared by all the threads within the process. In addition the execution state of the process comprises a stack for each thread, each stack holding context frames, in turn each frame containing the registers, local variables and temporary results of a program module, as well as addresses for further module invocations and returns. Before describing an embodiment of the invention in more detail, however, it is first necessary to introduce some definitions of data types and functions that are used in the embodiment and which will be referred to further below.

In addition to conventional data types such as integers and pointer, four new data types Data, Module, Stack and Hibernaculum are defined as follows:

Data: A variable of this data type holds a set of data references. Members are added to and removed from the set by means of the following functions;

Int AddDatum(Data d, String dataname) inserts the data item dataname in the heap of the process as a member of d.

Int DelDatum(data d, String dataname) removes the data item dataname from d.

Module: A variable of this data type holds a set of references to program modules. Members are added to and removed from the set with the following functions;

Int AddModule(Module d, String modulename) inserts the program module modulename in the program area of the process as a member of d.

Int DelModule(Module d, String modulename) removes the program module modulename from d.

Stack: A variable of this data type holds a list of ranges of execution frames from the stack of the threads. The list may contain frame ranges from multiple threads, however no thread can have more than one range. Variables of this type are manipulated by the following functions:

Int OpenFrame(Stack d, Thread threadname) inserts into d a new range for the thread threadname, beginning with the thread's current execution frame. This function has no effect if the thread already has a range in d.

Int CloseFrame(Stack d, Thread threadname) ends the open-ended range in d that belongs to the thread threadname. This function has no effect if the thread does not currently have an open-ended range in d.

Int PopRange(Stack d, Thread threadname) removes from d the range belonging to the thread threadname.

Hibernaculum: A variable of this data type is used to hold a suspended process.

As will be explained in more detail below a process may be suspended and stored in a hibernaculum prior to being transferred from one operating environment to another operating environment and/or may be subject to the following evolutionary operation:

Hibernaculum Construct(Stack s, Module m, Data d): This operation creates a new process with the execution state, program table and data heap specified as input parameters. The process is immediately suspended and then returned in a hibernaculum. The hibernaculum may be signed by the originating process as indication of its authenticity. Fig.3 is a flow-chart showing the hibernaculum construct operation.

A hibernaculum may be sent between operating environments by the following send and receive functions:

Int Send(Hibernaculum h, Target t) transmits the process contained within h to the specified target.

Hibernaculum Receive(Source s) receives from the specified source a hibernaculum containing a process.

A hibernaculum may be subject to the following evolutionary function:

Int Assimilate(Hibernaculum h, OverrideFlags f) activates the threads of the process stored within h and runs them as threads within a calling process's operating environment. Where there is a conflict between the data and/or program modules of the hibernaculum and the operating environment, the override flags specify which to preserve. Fig.4 is a flow-chart illustrating the steps of the assimilate operation.

Int Mutate(Stack s, int sflag, Module m, int mflag, Data d, int dflag) modifies the execution state, program table and data heap of the calling process. If a thread has an

entry in s, only the range of execution frames specified by this entry is preserved, the other frames are discarded. Execution stacks belonging to threads without an entry in s are left untouched. In addition, program modules listed in m and data items listed in d are kept or discarded depending on the flag status. Fig.5 is a flow-chart illustrating the steps of the mutate operation.

Fig.2 illustrates very schematically how these operations may act on a process 230 (which may be loaded from an application 210 or a hibernaculum 220). The process 230 may be subject to a Construct operation 110 to create a hibernaculum, a hibernaculum may be sent to a stream by a Send operation 120, or received from a stream by a Receive operation 130. The contents of a hibernaculum may be assimilated in the process by an Assimilate operation 140, and a process may be caused to mutate by a Mutate operation 150. The process 230 may of course also be subject to traditional operations.

An example will now be described in which it is assumed that an executing process p1 is required to migrate from a first host machine t1 to a second host machine t2.

To begin with the process p1 calls the following function:

Hibernaculum h = Construct(Stack s, Module m, Data d)

where s contains all the execution stacks in the process, m contains all the application specific modules in the process (ie m excludes those modules that are system specific to the first host machine), and d contains all the application specific data (ie d excludes data that are system specific to the first host machine). This function creates a new process p2 that contains only the data, program code and execution states of the original process p1 that are specific to the application and not to the system of the first host machine. Process p2 is immediately suspended and returned within a hibernaculum h.

The original process p1 then calls the function:

Int Send(hibernaculum h, Host t2)

which transmits the process p2 contained within hibernaculum h as a stream to the new host machine t2.

At the new host machine t2 a new process p3 is created containing initial system specific data and program code relating to the new host t2. This new process p3 then immediately executes the function:

Hibernaculum h = Receive(Host t1)

which receives from the first host machine t1 the hibernaculum h containing the process p2 which includes only the application specific data, program codes and execution states.

Process p3 then calls the function:

Int Assimilate(Hibernaculum h)

to activate the threads of the process p2 stored in h and to run them as new threads within the environment of the calling process p3. The original thread in p3 then terminates, resulting in a process that has all the application specific portions of process p1, but with the system specific portions replaced to suit the requirements of the second host machine t2.

The original process p1 terminates.

In the example described above the process p1 discards first host specific information prior to migration. However, the discarding may be done after migration to the second host. This is described in the following embodiment.

To begin the process p1 calls the following function:

Hibernaculum h = Construct(Stack s, Module m, Data d)

where s contains all the execution stacks in the process, m contains all modules in the process (including both system specific and application specific modules), and d contains all data in the process (including both system specific and application specific data). Thus the entire process p1 is contained within the hibernaculum as a new process p2 that is identical to p1. Process p2 is immediately suspended and returned within hibernaculum h.

The original process p1 then calls the function:

Int Send(hibernaculum h, Host t2)

which transmits the process p2 contained within the hibernaculum h as a stream to the new host machine t2 where the process p2 is then re-activated. The process p2 then calls the function:

Int Mutate(Stack s, int sflag, Module m, int mflag, Data d, int dflag)

where s contains all the execution stacks in the process, m contains all the application specific modules in the process, and d contains all the application specific data in the process, and where the flags are set to retain the contents of s, m and d. In this way all execution states, and all application specific modules and data are retained in the process p2, but all data and modules specific to the system of the first host are discarded. Process p2 is then stored within a hibernaculum h in the second host using the construct operation.

At the new host machine t2 a new process p3 is created containing initial system specific data and program code relating to the new host t2. This process p3 then calls the function:

Int Assimilate(Hibernaculum h)

to activate the threads of the process p2 stored in h and to run them as new threads within the environment of the calling process p3. The original thread in p3 then terminates, resulting in a process that has all the application specific portions of p1, but with the system specific portions replaced to suit the requirements of the second host machine t2.

It will also be understood that in a variation of this second embodiment the mutate and assimilate functions may be reversed. That is to say, following the transfer of process p2 (identical to p1) from host t1 to host t2, process p2 may assimilate process p3 (containing the t2 system specific data and code) before the mutate operation is used to discard the t1 system specific data and code.

Thus it will be seen that there is provided a method by means of which a process can migrate from one host machine to another host machine and in doing so can adapt to the system requirements and configurations of the second host machine. Such a method allows processes to be readily transferred between host machines thus substantially facilitating the creation of a mobile computing environment.

To implement the process migration system in a Java environment, a package called snapshot is introduced. This package contains the following classes, each of which defines a data structure that is used in the migration and adaptation operations:

```
public class Hibernaculum {
        ...
}

public class State {
        ...
}

public class Module {
```

```
        ...
    }

    public class Data {
        ...
    }

    public class Machine {
        ...
    }
```

In addition, the package contains a Snapshot class that defines the migration and adaptation operations:

```
    public class Snapshot {
        private static native void registerNatives();
        static {
            registerNatives();
        }

        public static native Hibernaculum Construct(State s, Module m, Data d);
        public static native int Send(Hibernaculum h, OutputStream o);
        public static native Hibernaculum Receive(InputStream i);
        public static native int Assimilate(Hibernaculum h, int f);
        public static native int Mutate(State s, int sflag, Module m, int mflag, Data d, int
        dflag)

        // This class is not to be instantiated
        private Snapshot() {
        }
    }
```

The methods in the Snapshot class can be invoked from application code. For example:

```
try {
        if (snapshot.Snapshot.Construct(s, m, d) != null) {
                // hibernaculum has been created
        } else {
                // failed to create hibernaculum
        }
        catch(snapShot.SnapshotException e) {
                // Failed to create hibernaculum
        }
```

The migration and adaptation operations are implemented as native codes that are added to the Java virtual machine itself, using the Java Native Interface (JNI). To do that, a Java-to-native table is first defined:

```
#define KSH "Ljava/snapshot/Hibernaculum;"
#define KSS "Ljava/snapshot/State;"
#define KSM "Ljava/snapshot/Module;"
#define KSD "Ljava/snapshot/Data;"

static JNINativeMethod snapshot_Snapshot_native_methods[] = {
        {
                "Construct",
                "("KSSKSMKSD")"KSH,
                (void*)Impl_Snapshot_Construct
        },
        {
                "Send",
```

```
            "("KSH"Ljava/io/OutputStream;)I",
            (void*)Impl_Snapshot_Send
        },
        {
            "Receive",
            "(Ljava/io/InputStream;)"KSH,
            (void*)Impl_Snapshot_Receive
        },
        {
            "Assimilate",
            "("KSH"I)I",
            (void*)Impl_Snapshot_Assimilate
        },
        {
            "Mutate"
            "("KSSKSM"I"KSD"I)I"
            (void*)Impl_Snapshot_Mutate
        },
};
```

After that, the native implementations are registered via the following function:

```
JNIEXPORT void JNICALL
Java_snapshot_Snapshot_registerNatives(JNIEnv *env, jclass cls) {
    (*env)->RegisterNatives(    env,
                                cls,
                                snapshot_Snapshot_native_methods,
                                sizeof(snapshot_Snapshot_native_methods) /
                                sizeof(JNINativeMethod)     );
}
```

Besides the above native codes, several functions are added to the Java virtual machine implementation, each of which realizes one of the migration and adaptation operations:

```
void* Impl_Snapshot_Construct(..) {
        // follow flowchart in Figure 3

        ...
}

void* Impl_Snapshot_Send(..) {
        // send given hibernaculum to specified target
        ...
}

void* Impl_Snapshot_Receive(..) {
        // receive a hibernaculum from a specified source
        ...
}

void* Impl_Snapshot_Assimilate(..) {
        // follow flowchart in Figure 4
        ...
}

void*Impl_Snapshot_Mutate(..){
        //follow flowchart in Figure 5
        ...
}
```

The embodiment of the present invention is concerned with utilising the techniques described above in a manner that can be easily performed by the user.

In this respect, an application termed a task manager providing a visual control panel on the user's desktop, is provided to perform the process suspension and migration operations and functions. The computing processes are active Java applications running on a Microsoft Windows machine. To suspend and resume these processes, the Java virtual machine is enhanced as described above. This enhanced JVM, installed on every user machine, can start up new Java applications and resume suspended processes.

When starting or resuming a process, the JVM activates the task manager if it is not already running on the user machine. Once started, the task manager listens at a pre-defined port. Each new process contacts the task manager at that port, and the two then establish a two-way socket connection between them. Using the connection, the process updates the task manager on any changes in status, including process termination, window creation and closure. The connection is also used by the task manager to send commands to the process.

The task manager 110 shown on the computer screen desktop 100 of a user is illustrated in Fig. 6. Every computer process on the user's machine is registered with the task manager and in selection box 115 the name of the window(s) owned by the process and other information such as status etc. are listed. In Fig. 6, a bar chart process, owning the window 200 with the label "Test 29", is shown open on the desktop.

The task manager has function buttons 120 – 127 as follows:

E-mail button 120: Clicking on this button allows the user to suspend a process and send it to a recipient as an attachment to an e-mail message.

Save button 121: Clicking on this button allows the user to suspend a process and deposit it in a database. An e-mail containing a reference to the location of the process may optionally be sent to the intended recipient, so that the process can be retrieved later by clicking on the reference.

Open button 122: Clicking on this button allows the user to retrieve a saved process. A pop-up list of process that the user is allowed to retrieve is shown. On selection of the process, this is reconstructed on the user's machine.

Push button 123: Clicking on this button allows the user to suspend a process, then transfer (move or copy) it directly to another machine. The process is selected first and the destination address (target machine name and login information) is entered to activate the transfer.

Pull button 124: Clicking on this button allows the user to transfer (move or copy) a process from another machine. Upon entry of the source address (target machine name and login information), a pop-up list of active processes that the user can transfer is shown. Selection of the desired process causes the process to be transferred.

New button 125: Clicking on this button allows the user to select an application and run it as a new process on the machine.

Exit 126 and Info (help) 127 buttons have their usual meanings.

The functions identified by function buttons 120-124 are performed as follows:

1) E-mail Function

   a) The sender picks process to send and clicks on the e-mail button 120 (Fig. 6);

   b) A hibernaculum of the process is created using the Hibernaculum Construct operation;

   c) An output stream is opened to a target in the form of a file and the hibernaculum is transferred to the target file using the Int Send function;

   d) An e-mail sending program (e.g. Microsoft Outlook) is called for input of the destination address of the recipient, subject and any text (Fig. 7);

e) The file is attached to the e-mail and sent; (the e-mail fnction ends here, the next steps being performed by the recipient)

f) The recipient on his machine opens the e-mail 135 received in his mailbox using an e-mail application (e.g. Microsoft Outlook), and selects the file attachment 140 in the received e-mail (Fig. 8)

g) This causes the suspended process in the file attachment 140 to be reconstructed. Since the task manager is not already running, it is activated. After that, the reconstructed process registers itself with the task manager (Fig. 9).

h) The reconstructed process opens an input stream from the (source) file attachment 140 and the hibernaculum is read from the input stream using the Hibernaculum Receive function

i) The reconstructed process absorbs the hibernaculum using the Int Assimilate function (Fig. 9), thereby resuming the suspended process 200.

The screens presented to the user (sender or recipient, where appropriate) at certain stages in the performance of steps (a)-(h) are illustrated in Figs. 6-9 as noted. Similar screens (not shown) adapted to the tasks to be performed are provided for the other functions.

2) Save Function

a) The sender picks process to send and clicks on the Save button 121;

b) A hibernaculum of the process is created using the Hibernaculum Construct operation;

c) An output stream is opened to a target in the form of a file and the hibernaculum is transferred to the target file using the Int Send function;

d) A file-sending program is called for entry of the address of the save location. The save location is on a web server running at a pre-specific machine and port;

e) The file is sent to the location;

f) An e-mail sending program is called for input of the destination address of the recipient and subject. The body of the e-mail contains any text and a handle (a URL link) specifying the location of the file;

g) The e-mail is sent to the recipient; (the Save function ends here, the next steps being performed by the recipient)

h) The recipient on his machine clicks on the handle in the received e-mail

i) This causes the suspended process associated with the handle to be reconstructed. If the task manager is not already running, it is activated. Following this, the reconstructed process registers itself with the task manager.

j) The reconstructed process opens an input stream from the (source) file specified by the handle and the hibernaculum is read from the input stream using the Hibernaculum Receive function.

k) The reconstructed process absorbs the hibernaculum using the Int Assimilate function, thereby resuming the suspended process.

In order to open the e-mail attachment or server-based file, a new MIME-type is registered on the recipient's machine so that, when the recipient clicks on file attachment or URL link, the file is automatically opened with the enhanced JVM. On Windows 95/98, this is done by double-clicking the "My Computer" icon, picking the "Options" entry in the "View" menu, clicking the "File Types" tab, and finally adding a "New Type" for "application/x-java-task" with the ".jpm" extension that is associated with the enhanced-JVM program.

The user may transfer (move or copy) the file attachment or URL link from its receipt location (e-mail In box) to another location or application. One particularly preferred transfer is to a calender 300 or scheduler in which the attachment/link 310 is further associated with scheduling information or a deadline 320. The result of this transfer is illustrated in Fig. 10.

3) Open Function

a) The user enters his login information, after which he is shown a list of handles of suspended processes (created through steps a)-e) of the Save function) on the web server;

b) Upon selecting a handle, steps (i)-(k) of the Save function are followed.

4) Push Function

a) The sender picks the process to send from the list displayed in the task manager and clicks on the Push button 123;

b) The task manager notifies the recipient machine to create a new process. The task manager is activated if it is not already running.

c) The sender machine opens an output stream to the new process on the recipient machine;

d) The new process on the recipient machine calls the Hibernaculum Receive function to operate on the (input) stream from the sender;

e) The sender machine creates a hibernaculum of the process using the Hibernaculum Construct operation;

f) The sender machine writes the hibernaculum to the output stream using the Int Send operation;

g) The recipient machine reconstructs the process from the hibernaculum using the Int Assimilate function.

5) Pull Function

a) The recipient clicks on the Pull button 124, specifies the sender machine and selects the process to be transferred;

b) A new process is created on the recipient machine. The task manager is activated if it is not already running;

c) The sender machine opens an output stream to the new process on the recipient machine;

d) The new process on the recipient machine calls the Hibernaculum Receive function to operate on the (input) steam from the sender;

e) The sender machine creates a hibernaculum of the process using the Hibernaculum Construct operation;

f) The sender machine writes the hibernaculum to the output stream using the Int Send operation;

g) The recipient reconstructs the process from the hibernaculum using the Int Assimilate function.

This embodiment is not to be construed as limitative. For example, the E-mail function sends the hibernaculum by e-mail to the mailbox of the recipient which may be accessed by any suitable means, for example an e-mail application or personal information manager (PIM) used in any operating system. The transfer may be via any transfer medium, for example via Internet, Intranet, local area network or other communications link. The recipient may open his mailbox on the same machine as the sender, so that the transfer is within the machine. The Save function in a similar way can use any transfer medium and deposit the hibernaculum in any storage location, for example on a server, or on the same or a different machine to the sender. Similar variations are envisaged for the Open, Push and Pull functions. The sender and recipient may be identified as different machines, different users, accounts or logins on the same or different machines or can be effectively the same, spaced simply in time, with the sender effectively saving the process for subsequent reactivation by himself on the same machine later.

CLAIMS

1. A method of transferring an active application from a sender to a recipient, the method comprising the steps of:

(d) creating a hibernaculum of a process containing some or all of the program modules, data and execution state of the active application,

(e) sending the hibernaculum to a location for retrieval by the recipient; and

(f) reconstructing the active application from the retrieved hibernaculum.

2. A method as claimed in claim 1 wherein the location is the mailbox of the recipient.

3. A method as claimed in claim 2 wherein the hibernaculum is sent as an attachment to an e-mail sent to the recipient's mailbox.

4. A method as claimed in claim 1, wherein the hibernaculum is sent to a personal information manager of the recipient.

5. A method as claimed in claim 1 wherein the hibernaculum is sent to a storage location remote from the recipient.

6. A method as claimed in claim 4 wherein a reference to the storage location is sent to a mailbox of the recipient.

7. A method as claimed in claim 5 wherein the storage location is on a World-wide Web server and the reference is a URL link.

8. A method as claimed in claim 1 wherein the hibernaculum is sent to a storage location remote from the recipient and a reference to the storage location is sent to a personal information manager of the recipient.

* C L M S *

9. A method as claimed in any one of the preceding claims wherein, prior to step (c) the hibernaculum is transferred by the recipient to a further location and opened from the further location.

10. A method as claimed in claim 9 wherein the further location is a calendar or scheduler.

11. A method as claimed in any one of claims 6 to 8 wherein, prior to step (c) the reference is transferred by the recipient to a further mailbox or personal information manager and opened from there.

12. A method of transferring an active application from a sender to a recipient, one being an initiating party and the other being a target, the method comprising the steps of:

(a) specifying the address of the target that is to participate in the transfer with the initiating party;

(b) creating a hibernaculum of a process containing some or all of the program modules, data and execution state of the active application;

(c) sending the hibernaculum to the recipient;

(d) reconstructing the active application from the hibernaculum.

13. A method as claimed in claim 12 wherein the sender initiates the transfer.

14. A method as claimed in claim 12 wherein the recipient initiates the transfer.

15. A method as claimed in any one of the preceding claims wherein the sender and recipient are different machines.

16. A method as claimed in any one of the preceding claims wherein the sender and recipient are different users.

17. A method as claimed in any one of claims 1 to 14 wherein the sender and recipient are the same, the steps of sending and recovering being spaced in time.

18. Apparatus for performing the method of any one of the preceding claims.

