OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Demande-Application**

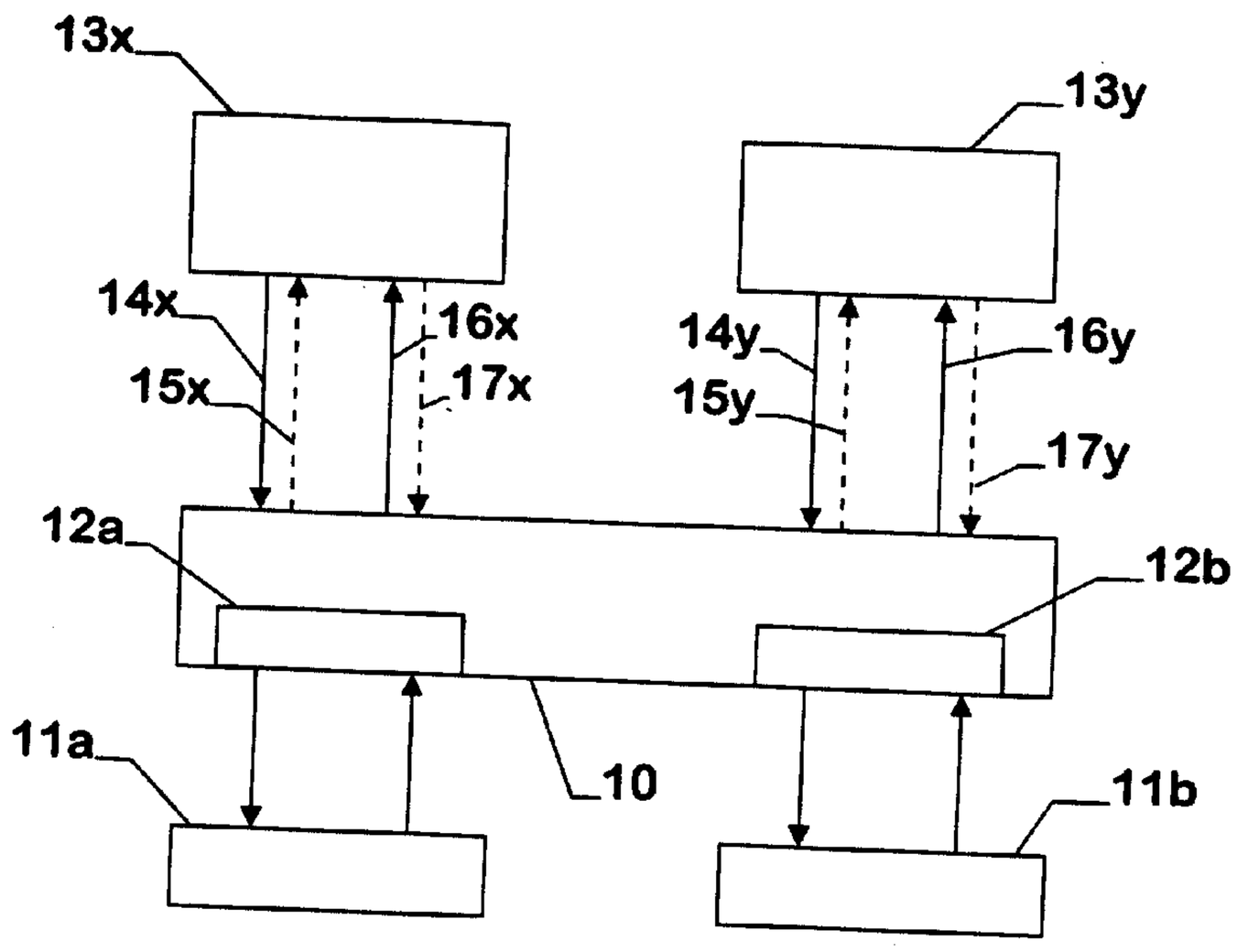
(21) (A1) **2,302,587**
(86) 1998/07/17
(87) 1999/04/01

(72) GRUPE, FRANK, DE
(71) SIEMENS NIXDORF INFORMATIONSSYSTEME AG, DE

(51) Int.Cl.[7] G06F 9/46, G06F 9/445
(30) 1997/09/24 (19742149.0) DE

(54) **PROCEDE DE COMMUTATION D'ETAPES DE TRANSACTIONS**

(54) **METHOD FOR SWITCHING TRANSACTION STEPS**



(57) L'invention concerne un procédé d'exploitation d'une station de données avec une interface utilisateur servant au chargement et à l'exécution de programmes partiels, le système d'exploitation mettant à disposition des moyens d'exploitation par l'intermédiaire d'une interface pour l'exploitation orientée liaison et un programme de commutation actif en permanence, recevant des instructions à partir des programmes partiels pour commander les moyens d'exploitation par l'intermédiaire d'une interface à datagrammes.

(57) The invention relates to a method for operating a data station with a user interface for loading and executing partial programs, wherein the operating system provides operating means via an interface for connection-oriented operation and a permanently active switching program receives instructions from the partial programs via a datagram interface to operate the operating means.

Abstract

Method for operating a data station with a user interface for loading and executing program elements, the operating system making available resources via an interface for connection-oriented operation, and a continuously active switcher program receiving from the program elements jobs for the operation of the resources via a datagram interface.

Fig. 1

Method for switching transaction steps

Technical Field

The invention relates to an operating method for data-processing units, in particular those in which a client accesses local transaction services using transient programs.

Prior Art

A large number of devices and automatic equipment, here in particular automated teller machines, service stations and information stations as well as automated booking machines, contain a data-processing system for control purposes and thus constitute a data-processing unit. Often, corresponding to the available computer technology and power, said unit is derived from a commercially available universal system such as a personal computer or working-group server, and uses its operating system. Screens with the contemporary graphic features are available for the user interface. The information is presented preferably by means of an interpreter, referred to as a "browser", for in particular the document description language HTML (T. Berners-Lee, D. Connolly: Hypertext Markup Language. RFC 1866, Nov. 1995).

More recent versions and variants of HTML provide for short programs, which are referred to as "Applets" and which are referred to below as program elements, to be embedded. Known examples of this are the language JAVA developed by Sun Microsystems, the subset JAVA script derived therefrom, published as ECMA script in the ECMA-262 standard

(June 1997), and the interface "ActiveX" developed by Microsoft. The fundamental property of such program elements is that they, access services of the operating system or its environment, the HTML Interpreter. They serve in particular to make available functions and sequences which are not provided by the language HTML.

Associated with this are, in particular, the processing of transactions with and the access to resources of the data station on which the HTML pages are displayed. These resources include chip card readers, remote data-processing connections with various protocols or receipt and voucher printers. As is the case with virtually all resources of this type, they are operated via a connection-oriented interface in which initially the communication with the resource is opened by means of OPEN, the processing then takes place by means of a series of READ, WRITE or IOCTL instructions, and the connection is disconnected by means of CLOSE. Program elements such as JAVA applets or ActiveX Controls also provide these possibilities.

However, since a program element is always active only for as long as the page is displayed, program elements of different pages or successive calls of program elements cannot communicate with one another. In addition, the sequence of the program elements is not predefined, since it is defined by the user behaviour. This means that each program element must always firstly open the resource with OPEN in order to be able to access it. However, this is quite time-consuming since the opening of a connection with OPEN for the purpose of more rapid processing with READ/WRITE is relatively slow and complex. This then makes itself felt through long waiting times to set up a page, which considerably negate the benefit of this solution

using HTML interpreters and subprograms.

The object on the invention is therefore to specify a solution with which program elements can quickly access connection-oriented resources without themselves having to open a connection; including rapid processing of transactions of any desired type.

The means of achieving the object makes use of the knowledge that the subprograms operate to an optimum degree with a datagram interface but require the resources of a connection-oriented interface. For this reason, a switching program or process is used which has a datagram interface in the program elements and which itself uses the connection-oriented interface. Thus, the switching process can carry out the time-consuming opening of a resource once when it starts and can execute very quickly the jobs received via the datagram interface.

The invention is therefore a method for operating a data station with a user interface for loading and executing program elements, the operating system making available resources via an interface for connection-oriented operation, and a switching program which is continuously active receiving from the program elements orders to operate the resources via a datagram interface.

The following description, which also presents variants and developments, uses two types of communication between programs or processes, namely the datagram communication and the connection-oriented communication.

Predominantly connection-oriented communication means are used. These include relatively old analogue means, such as telephone or telefax, or relatively recent digital means, such as ISDN or

TCP/IP. In all cases, an association is established between a calling party and a called party by means of a dialling process which takes a certain amount of time. Then, resources are reserved and there is available an actual or virtual communication channel via which the parties can exchange messages without delay, compared with the time for the dialling process, because the resources have already been assigned (and can also be billed as a service). These connections do not ensure immediate delivery, but do always ensure that the data are delivered in the sequence in which they were transmitted.

In contrast to this, explicit use of a communications method which is designated a datagram protocol and in which data are transferred and delivered autonomously in small units, i.e. without correlation of successive transmissions, is made relatively rarely. The internet protocol IP operates in this way and makes the UDP/IP protocol available for applications. The advantage of this protocol lies in the low degree of complexity because there is no set up or release of a connection. If only one or a small number of data packets have to be sent in total, a datagram protocol is considerably more efficient than a connection-oriented one. This is the case even if the receiver of a message has to acknowledge it actually by means of a further datagram. The IP protocol utilizes the transfer of the datagram to protect against network faults, since it is possible to decide afresh for each datagram what path it will be transmitted on. The protocol TCP/IP implements a connection-oriented protocol by means of a datagram protocol, therefore implements, in contrast to the present invention, connection-oriented communication by means of a datagram communication.

According to the principle involved, the customary interfaces for processing files in an operating system also constitute a

connection-oriented protocol by virtue of the fact that processing is opened by means of OPEN and closed by means of CLOSE, and a number of READ or WRITE are possible in between. This is apparent from the fact that the INET Demon in a POSIX operating system calls services which operate with the connection-oriented protocol TCP/IP in that the standard input and output is, instead of being diverted to files, diverted to the communication connections and, in terms of the principle involved, it makes no difference to the program whether it is operating locally or as a network application.

Common memory areas ("shared memory") make available a datagram service in structural terms as soon as the access is serialized by means of semaphores. A plurality of processes can file messages in an uncorrelated fashion without the opposite station having to be active to permit this. For this reason, general mention is made below of a datagram service, including implementation by means of a common memory.

A further communication method, also referred to as "named pipes" could also be referred to as "known channels". They have a special position insofar as they are structurally a connection-oriented service but can also easily be used for datagrams. Thus, it is known, for example, that a background process receives commands via a "named pipe" and, for this purpose, opens the "named pipe" once and then waits for commands. However, these commands are independent and may be of a non-predefined order and type and can come in a non-predefined sequence from numbers and types of transmitter which are likewise not predefined, said transmitters using the "named pipe" in each case by virtue of the "OPEN-WRITE-CLOSE" sequence. The open procedure for a "named pipe" is very quick here because no peripherals are affected. If the receiver of the "named pipe" therefore considers each data set as an autonomous

job, "named pipes" constitute an alternative to other datagram methods.

As already mentioned, the program elements of an HTML page are short programs with an extremely volatile existence, which are actuated only for the duration of the activation of an HTML page. For this reason, a datagram service is considerably better suited for controlling locally situated peripherals than a connection which firstly has to be set up and then released again, although only a very small message has to be sent once.

If this situation is detected, a solution can be achieved by virtue of the fact that a switcher maps the desired datagram protocol onto the existing switching protocols.

In the preferred embodiment, the switcher is an independent process which is started in a known fashion when the system is started. If UDP/IP is used as the datagram interface, starting can also be carried out "on demand" by the known Inet Demon.

Transactions are to be equated with the connection oriented resources insofar as a transaction is opened, carried out in steps and then either aborted or completed. A program element would then also firstly have to be registered here in a transaction program, would have to open the transaction, register the transaction step, confirm the transaction and close the connection. The switcher according to the invention opens the connection only once and receives from the program element only the data for the transaction step, the other steps being permanently provided in the switcher. Although this solution is appropriate only for simple transactions, they constitute a large proportion of the entire transactions so that accelerating them is effective.

Further features and advantages of the invention emerge from the following description, which explains the invention with reference to an exemplary embodiment in conjunction with the appended drawings.

Brief description of the drawing

Fig. 1 shows a block circuit diagram of the components involved and

Fig. 2 shows a diagram of the chronological sequence according to the invention.

Description of an embodiment of the invention

Fig. 1 is a symbolic block circuit diagram of the components involved in the invention. The switching program 10, referred to below for short as "the switcher", is preferably started during the starting of the entire system. A plurality of resources 11a, 11b, such as chipcard readers, connection-oriented remote data connections, for example according to X25 or in the SNA network system or voucher printers are operated by means of an interface 12a, 12b. This interface is frequently made available by the operating system and is handled in a connection-oriented fashion by means of instructions such as OPEN, READ/WRITE/CNTL and CLOSE. Even if the interface is not part of the operating system, an interface is usually provided as a module which behaves largely similarly to an operating system interface.

The switcher 10 reads designations, from a configuration file (not illustrated) for the interfaces 12a, 12b to be operated,

and attempts to open them. The handling of faults occurs in the usual way and is not a subject-matter of this invention. It is assumed below that the resources are opened and available. The number of resources is, of course, any number between one and the quantity of resources present in total in the data-processing unit insofar as they are not used exclusively by other components.

In addition, the switcher 10 makes available means with which it can be accessed via datagrams, as is assumed below. Here, the datagram protocol which is used can be filed in parameter files or a recording database, with the result that it does not have to be contained permanently in the program elements insofar as these can access such means. The same applies to the resources which are available through them.

In the course of the operation of the data-processing unit, program elements 13x, 13y are loaded and activated. While the switching program 10 with the resources 11a, 11b is provided statistically, the program elements 13x, 13y are present only temporarily and are also, as a rule, not present at the same time. The two program elements illustrated in Fig. 1 therefore represent here symbolically any desired number or type of program elements which are activated by the behaviour of the user and can follow one another in a chronologically overlapping fashion desired.

The description below refers to the program element 13x with the communications steps 14x, 15x, 16x and 17x; the same applies to the same degree to the program element 13y with the communications steps 14y, 15y, 16y and 17y.

The program element 13x, which wishes to use one of the resources 11a, 11b, creates a job

control block (not illustrated) and transmits it as a message 14x via the agreed datagram interface to the switching program. In a simplified and schematic form this is as follows:

```
procedure ReqAct;
     struct TReqAct thisReq
     thisReq.Op = 'r'
     thisReq.dev = '11b'
     ...
     rc = subDgram ('10', thisReq)
```

A control block "thisReq" is set up and the instruction information "Op" and the desired resource 11b are specified in the field "dev"; further information in the control block depends as a rule on the resource and is filled in in a known manner similarly, for example, to a READ instruction for the resource 11b. For this purpose, the control block may comprise variants, something with which a programr of, for example, PASCAL or C is familiar. Then, using the call "subDgram", the control block "thisReq" is sent to the switcher 10; as a result, a code "rc" is delivered.

In the switcher 10, which preferably utilizes the possibilities of a plurality of monitoring flows by means of "threads", contains a component for reception. This is carried out using resources in the operating system in such a way that a procedure is registered as a receiver and is activated when a message is received. This procedure generates a new monitoring flow ("thread") for processing the order and transmits back an acknowledgement 15x which is received and analysed as code "rc" in the program element 13x. This code can specify in a known fashion whether the order has been accepted or for what reasons it had to be refused.

At the latest when the order has been accepted, the program element must specify a program code which, for example as a subroutine, is to receive a message from the switcher.

For synchronization reasons, it is expediently carried out directly beforehand. This function is part of the "host environment" and is therefore not described in the aforesaid ECMA-262, but rather can be found in the documentation of the respective component which calls the program element. The documentation relating to the NETSCAPE component, for example, can be called up under 'http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html'. Depending on the assignment of an event to a JavaScript statement by the function "onReset", a message 16x is assigned a JavaScript statement by the switcher 10.

The known "reset" event is assigned, for example, an "alert" program element, for example by means of the call of "onReset", as follows:

```
<FORM NAME="form1"
onReset="alert('Defaults have been restored.')">
State:
<INPUT TYPE="text" NAME="state" VALUE="CA"
SIZE="2"><P>
<INPUT TYPE="reset" VALUE="Clear Form"
NAME="reset1">
</FORM>
```

In the same way, the event "message from switcher 10" can be assigned a program element by "onMsg(10)':

```
<FORM NAME="form1"
onMsg('10') ="procReply">
State:
<INPUT TYPE="text" NAME="state" VALUE="CA"
SIZE="2"><P>
>INPUT TYPE="reset" VALUE="Clear Form" NAME=
"reset1">
</FORM>
```

Here, no parameter is given under the function name 'procReply' because, at the call, a reference is transferred to a suitable structure for the transferred data packet as a parameter. A code for the switcher and for the format of the data packet could also be specified as a further parameter. This function is not yet present in

the aforesaid environment for the above specified NETSCAPE Interpreter and must be introduced there if it is actually this which is to be used.

The data packet contains the data obtained by the resources, said data being, for example, the number of the inserted chipcard. After the operation, here READ onto a chipcard, have been terminated in the switcher and the data are present, a message is built up and sent to the program element. As a result, the program element which is associated with the message, the procedure 'procReply' in the example, is called and receives the data via an agreed data structure.

The procedure described using the parallel work by means of 'threads' and 'events' corresponds to the possibilities of current operating systems. However, an application without using it is equally possible in that the acknowledgement of the job already contains the result and not only a code of the type "job accepted". By means of this synchronous call, for example, by means of a "remote procedure call", RPC, which can be introduced particularly easily into any existing program language, that is to say also JAVA, JavaScript or ActiveX, the program element waits for the execution of the job.

In the description it is also assumed that the datagram interface supplies a result code. If one is not provided, for example in the protocol UDP/IP, it is possible, in the first instance, for the reverse transmission of one to be agreed. The program element can equally well simply wait for the response with the data and set itself a timer, for example, by means of the JavaScript function "setTimeOut", and consider the request to have failed when the time out occurs.

Fig. 2 illustrates the aforesaid sequences once more in a chronological relationship. Since the subprograms in, for example, JavaScript, are preferably programmed in an event-oriented fashion,

two execution paths are illustrated for the program element 13x and the switcher 10.

Firstly, in a first part 20, a job 14x is sent to the switcher 10 and handled in the second part 21a. There, a new execution path 21b is initiated and an acknowledgement sent back. The new execution path 21b presents a job to the resources and waits until its acknowledgement. This job is processed there in a third part 22 and then further handled in a fourth part 23 in the switcher in synchronism in the sense of a connection-oriented control. In the example it is assumed that a fifth part 24 is necessary in the control for the resources 12a, after the end of which the procedure will be terminated so that the sixth part 25 sends back the result or results 16x and itself receives an acknowledgement from the seventh part 26 in the switcher.

The previous description assumed that the interface 12a to a resource is opened when the switcher 10 starts and is not closed until the end. Since this delays the start, the interface can, in a known fashion, be opened at the first request but then not yet closed when said request is dealt with. If the number of interfaces which can be kept open simultaneously is small, it is, of course, also possible to close the interface which has not been used for the longest time by means of a "least recently used" strategy where necessary, and for it to be opened again when it is necessary. If it initially becomes apparent from the sequence of program elements that no further use of a resource is to be expected, an instruction to close the interface may also be provided. Whether the switcher does this immediately, after a delay or not at all, is for it to decide and can be dependent, for example, on use statistics which are simultaneously registered.

Patent Claims

1. Method for operating a data-processing unit having the features:

- a user interface comprises a means for loading and executing program elements (13x, 13y) by activating elements of the user interface,
- an operating system makes available at least one resource (11a, 11b) which is not administered by the user interface and whose interface (12a, 12b) provides connection-oriented operation,
- a switcher (10) receives from program elements (13x, 13y) jobs (14x, 14y) for the operation of resources (11a, 11b) whose interface (12a, 12b) provides connection-oriented operation, the jobs (14x, 14y) being transferred via a connectionless interface.

2. Method according to Claim 1, the switcher opening the connection to a resource when a job which relates to the resource is received.

3. Method according to Claim 1, the switcher opening the connection to a resource directly after the start of operation and before a related job is received.

4. Method according to Claim 2 or 3, a job being provided which closes the connection to the related resource.

5. Method according to Claim 4, the closure being dependent on use statistics which are registered simultaneously.

6. Method as claimed in one of the preceding claims, the switcher registering simultaneously a status for each resource which can be operated,

and enabling both the status and the principle consequent states and/or the permissible jobs to be interrogated via the datagram interface.

7. Method according to one of the preceding claims, the switcher indicating via a recording data base the datagram interface.

8. Method according to Claim 7, the switcher indicating in the recording database the resources which can be operated.

9. Method according to one of the preceding claims, the user interface using a 'Hyper-Text Markup Language'.

10. Method according to one of the preceding claims, the data station being connected to a network and program elements being transferred via the network.

11. Method according to one of Claims 1 to 10, a common memory ('shared memory') being used as datagram interface.

12. Method according to one of Claims 1 to 10, channels ('named pipes') which are appointed as the datagram interface being used.

13. Method according to one of Claims 1 to 10, the protocol UDP/IP being used as datagram interface.

Fetherstonhaugh & Co.
Ottawa, Canada
Patent Agents

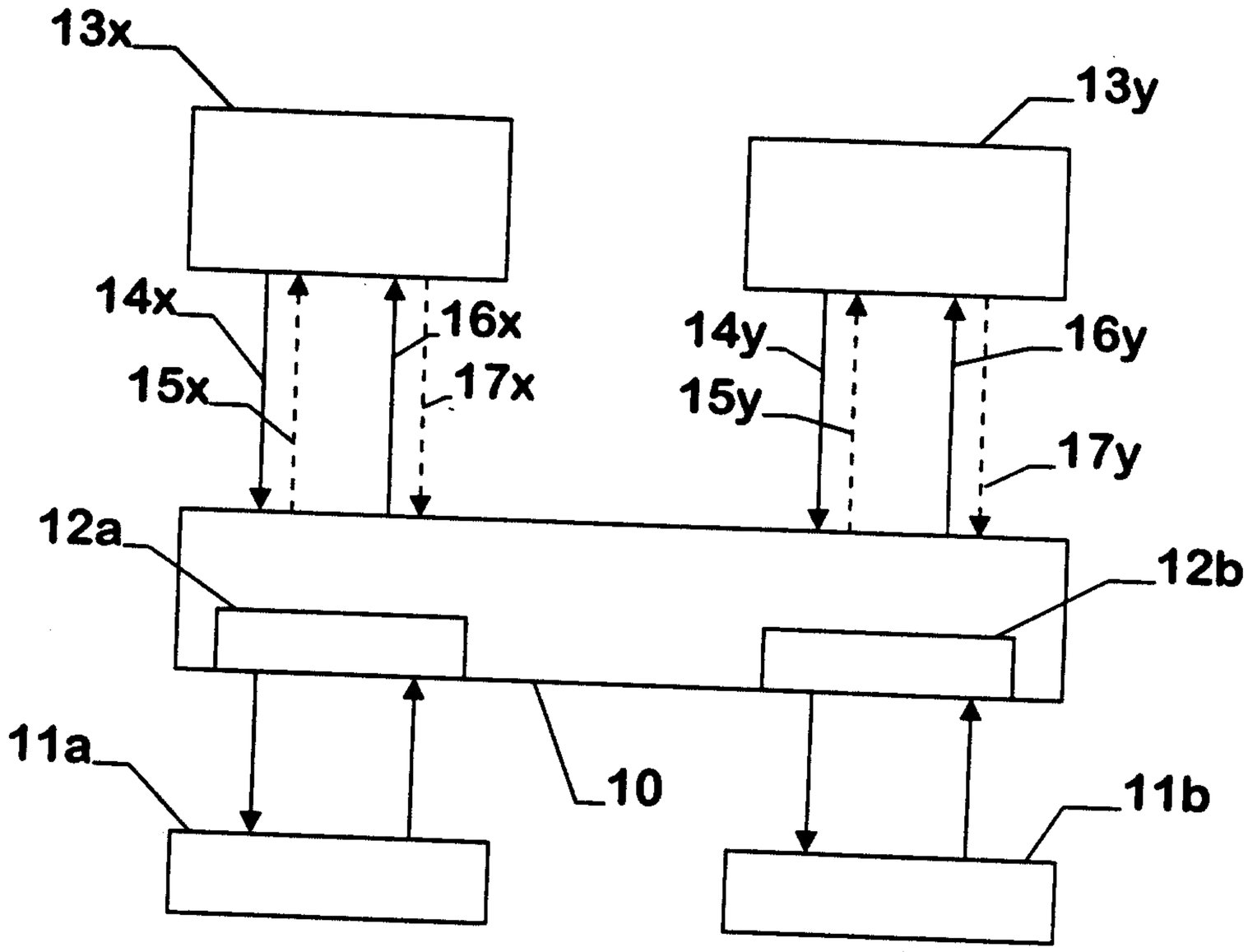

13x
13y
14x
15x
16x
17x
14y
15y
16y
17y
12a
12b
11a
10
11b

Fig. 1

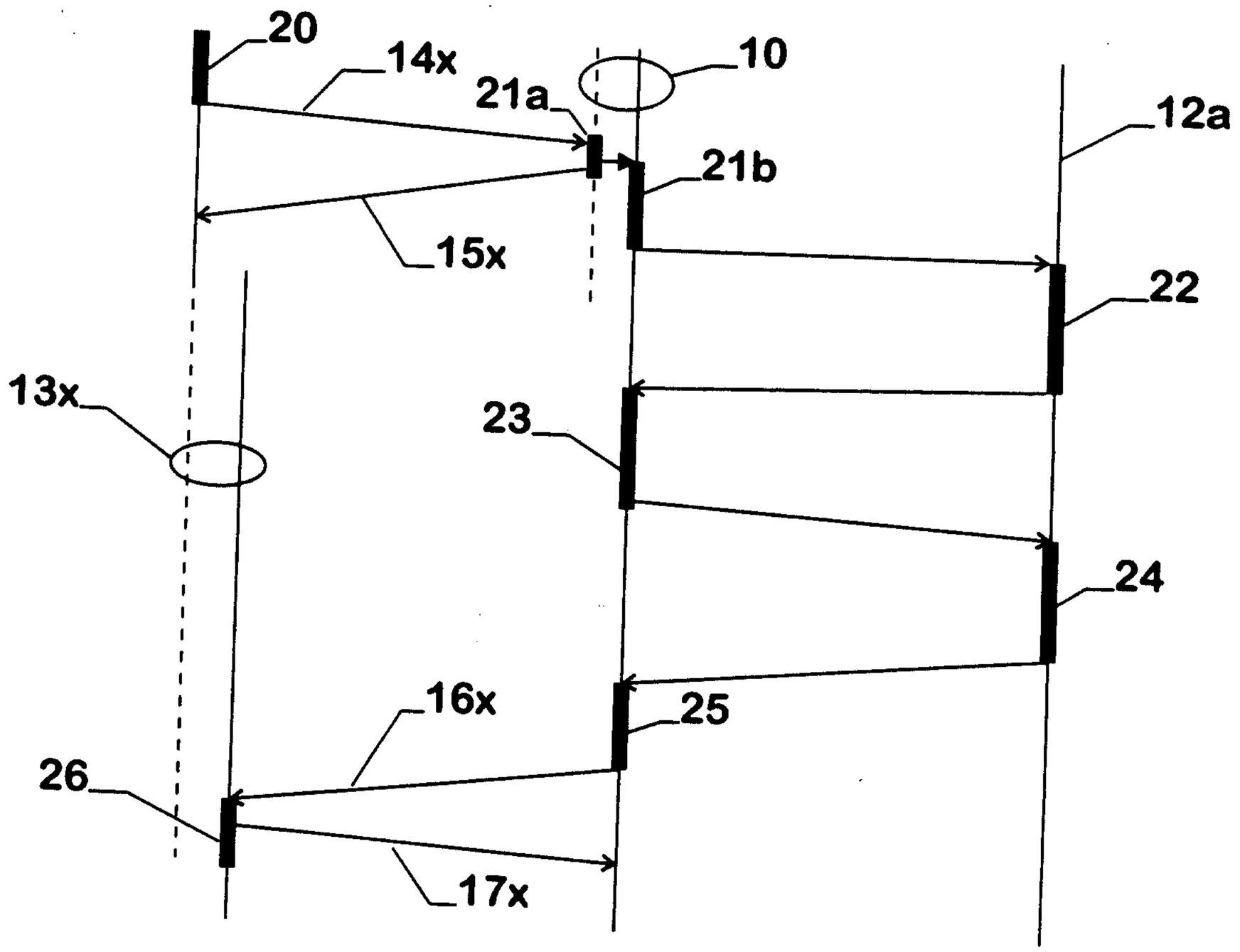

20
14x
21a
10
12a
21b
15x
22
13x
23
24
16x
25
26
17x

Fig. 2