(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 January 2001 (11.01.2001)

PCT

(10) International Publication Number
WO 01/02952 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/GB00/02560

(22) International Filing Date: 3 July 2000 (03.07.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
9915692.9 5 July 1999 (05.07.1999) GB

(71) Applicant *(for all designated States except US)*: **AQTIVE LIMITED** [GB/GB]; Birmingham Research Park, Vincent Drive, Edgbaston, Birmingham, West Midlands B15 2SQ (GB).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **DIX, Alan, John** [GB/GB]; 142 Windemere Road, Kendal, Cumbria LA9 5EZ (GB). **BEALE, Russell** [GB/GB]; Sunrise Cottage, 2 The Rocks, Holy Cross, Clent DY9 9QE (GB). **WOOD, Andrew, Michael** [GB/GB]; 110 Westminster Road, Selly Park, Birmingham B29 7RS (GB).

(74) Agents: **MCCALLUM, William, Potter** et al.; Cruikshank & Fairweather, 19 Royal Exchange Square, Glasgow G1 3AE (GB).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/02952 A2

(54) Title: IMPROVED SOFTWARE INTERFACE AGENT

(57) Abstract: A software agent is described that can suggest to the user appropriate software services, documents or other resources on a network or local computer. It also describes a novel software framework for the implementation of software agents and several other embodiments of the underlying invention. The underlying software framework, aQtiveSpace, is based on component programs called Qbits which have a variety of types of interaction which together allow highly flexible interconnection. The main embodiment, referred to in this document as aQtiveDesk and available commercially as onCue, is a desktop software agent that watches the users activity and each time the user copies text or other data it uses various recogniser programs (implemented as Qbits) to determine what kind of data is provided and then invokes various service programs (also Qbits) depending on the kind of data. These services include local desktop applications, Internet applications and shortcuts to web pages. Other embodiments, BrainStorm, SiteStore and DeskStore, also incorporate the principle of suggesting appropriate resources based on the users current activity and context.

IMPROVED SOFTWARE INTERFACE AGENT

The present invention relates to software interface agents particularly, but not exclusively, for providing assistance to a user in performing computer-based tasks.

Interface agents that work collaboratively with users on the desktop have recently received a lot of attention. These interface agents operate alongside a user, helping the user to use more traditional applications to complete everyday tasks. Such agents can provide a variety of assistance, such as context sensitive help, automation of repetitive tasks or presenting sensible default values or actions.

To date, one difficulty has been defining the term "interface agent". There are many definitions. However, as used hereinafter, a software interface agent is defined as a software entity that provides active assistance to a user with computer-based tasks.

There are various types of interface agents, such as user agents, autonomous agents, guiding agents and co-operative agents. Agents may also be based on combinations of these categories. The assistance provided by agents can be active or passive, or a combination of both.

Many existing software systems provide, or support, integration mechanisms of one form or another. Such systems may be conveniently organised into four general categories; application-application interaction; user-application interaction; agent-agent interaction, and agent-application interaction. Applicant-application interaction covers techniques such as data integration, client/server architecture, distributed objects, message queues and buses and component architectures. User-application interaction includes user interfaces, such as side-by-side windows, cut-and-paste and drag-and-drop, and scripting systems for allowing users to extend their computer systems by automating relatively simple tasks with programs/scripts written in high level language,

such as Apple's Open Script Architecture (OSA) and user-interface technology, such as the X Window System and SmallTalk 80 for enabling the production of highly interactive graphical user-interfaces. Agent-agent interactive systems have been developed for the field of artificial intelligence and distributed artificial intelligence. One example is the Knowledge Query and Markup Language (KQML) which is a product of the DARPA Knowledge Sharing Effort/Initiative which defines a format and meaning of a set of messages that can be communicated between agents that comply with the format. Other types of agent-to-agent interaction is achieved using mobile agent systems which are pieces of code which have the ability to transport themselves across a network to new locations. Agent-application interaction systems include a toolkit for building agents, such as IBM's Agent Building Environment (ABE) toolkit which is written in C++. A further development of agent-application interaction systems is the proposal and development of a standard to cover such interactions. The Foundation for Intelligent Physical Agents (FIPA) is a recently created standards organisation to "identify, select, augment and develop specifications of generic agent technologies which are usable across a large number of applications and provide a high level of inter-operability amongst applications"
(http://drogo.cselt.stet.it/fipa/spec/f7613.htm).

The FIPA group identified four classes of components in the environments the agents might interact with; humans, other agents, software applications and the physical world.

Existing software interface agents suffer from a number of disadvantages as listed in the thesis of Wood (Wood, Andrew M., PhD Thesis, Cameo; Supporting Agent-Application Interaction Catalogued April 26 1999). A software interface agent should satisfy a number of requirements to allow an interface agent to be used with

a single machine, networked and used with multiple user machines. Interface agents need to be able to actively and passively sense, and effect, some subset of an application's data, operations and events. Interface agents also need the ability to detect at runtime which facilities (including events) an application provides (at some semantic level) how to use them at a syntactic/ protocol level. In addition, interface agents need the capability to explore the local environment, as defined by the domains in which they participate, allowing them to discover the applications that are available to them at runtime. The environment itself must also be flexible enough to deal with, and provide notification of, changes in the number of applications and their capabilities dynamically at runtime and, finally, the environment should be extendible enough to take advantage of distributed and possibly temporarily networked machines and also to support multiple users.

An object of the present invention is to provide a software interface agent which obviates or mitigates at least one disadvantage of known interface agents.

A further object of the present invention is to provide a software interface agent which satisfies the above-mentioned requirements to provide an integrated software interface agent.

This is achieved, in its broadest aspect, by using the concept of status/event analysis to decompose systems into a set of software interface agents which can interact with each other and which are modelled as consisting of a set of software components, the components defining an internal state, a set of in-events, in-status, out-events and out-status. Interactions between the software components are further modelled as a set of state transitions and the interstitial behaviour. These components interact with each other and each of the components can be queried to obtain a specification of the interfaces so that further

objects in the system can find components which provide a certain type of interface functionality at runtime.

The software interface agent may be best explained by its implementation as a software interface agent which assists a user of a computer. In a preferred arrangement, this interface referred to as "aQtiveDesk" (aQtiveDesk is the internal development name of the product released commercially under the name "onCue"), is implemented in Java. The aQtiveDesk interface agent is one of many interface agents which use specific software objects which include instructions and data which may be embodied in the processing unit of a computer or incorporated onto software in a software product, such as a floppy disk or a CD. The aQtiveDesk is a software agent which augments existing user interfaces.

Existing user interfaces usually allow a user to select a certain subset of data from a current application, for example a word processing document, and let the user cut or copy this selection to a clipboard. The user then locates the destination for the data by loading or switching to a different application (or a different place in the same application) and then pastes the selection from the clipboard into this new location. The aQtiveDesk interface agent augments this by suggesting destinations for the data once it has been selected. If an appropriate suggestion is made by the interface agent, then the user does not have to find the destination; the user simply accepts the agent's suggestion and the agent will do the work. It should be understood that the aQtiveDesk interface agent does not stop the user from locating a destination in the normal way, if desired. It works collaboratively with the user by supplementing the user's usual actions.

According to a first aspect of the present invention, there is provided a method of providing active assistance to a user performing a computer type task by using a software interface agent with a control unit,

said method comprising the steps of:

providing a software interface agent for monitoring the activities of a user, said agent recording data representative of the user's activities,

analysing the data obtained from said user's activities,

identifying events/data present in said user's activities and generating data corresponding to said identified events/data,

comparing said generated data with a database of information services or data accessible by said user's control unit, and

selecting at least one service corresponding to the analysed and identified data and displaying to a user a representation of said selected service, said representation being selectable by the user to access said selected service.

Preferably, said method is performed in real-time when a user is using the control unit. Alternatively, said method may be performed when the control unit is not being used on a stored work product of the user.

In a preferred arrangement, the method is usable with a user control unit which is conveniently a computer. Alternatively, the method is usable with any similar device which incorporates an embedded computer processing unit, such as a microprocessor.
Conveniently, such similar devices may be a television control, a telephone handset, either landline or cellular phone.

Preferably, a graphical representation of the service is presented to the user. Alternatively, an audible representation of the service is provided to the user.

Therefore, a user's activities may be monitored via a keyboard, a television/video control, a telephone, use of items having of personal data, such as smart cards, magnetic strip credit cards, bank cards and the like.

In fact, the software interface agent may be used for any activities which are electronically communicatable and recordable for generating suggestions to the user of a control unit for related activities or areas.

According to a further aspect of the present invention, there is provided a software interface agent for monitoring the activities of a user using a control device and for providing prompts based on a review of data from the user's activities to the user to allow the user to decide to accept said prompts, said software interface agent comprising,

clipboard program means for monitoring the activities of a user, data recognition program means for comparing user input data with a user definable database of recognition program elements, and for providing a corresponding data recogniser output data,

user service program means coupled to at least one user service for receiving said recogniser output data and for activating said service, obtaining the results of said services, and

indication means for providing to the user an indication that a service has been performed, the user being capable of interacting with the indication means to obtain the results of the service performed.

Preferably, said indication means is a graphical display means. Alternatively, said indication means is an audible message provider.

Preferably, said computer program is operable in a host personal computer, laptop, palmtop or the like. Preferably, the software interface agent is downloadable from the internet to a user's host machine. Alternatively, said computer program is operable in a control unit such as a web TV remote control, a telephone, a cellphone which have a processing unit with a memory for receiving said program.

Preferably, said at least one service is a web browser program which is launched when said recogniser

output data is received.

Conveniently, the computer program includes further program elements for calculating what has been copied to the clipboard program means from the user's activities.

Conveniently, the service program means includes data types corresponding to a plurality of different services, said data type being generated by said data recognition program means.

In one arrangement, said services and data types are related:

Histogram	Table
Services Encyclopaedia	Words
Thesaurus	Single Word
SumIt!	Number List
Excel - Qbit	Table
Web Searches	Words

Conveniently also, recognition program elements have in-type and out-type data related as follows :

Recogniser	**in-type**	**out-type**
words recogniser (Wr)	text	words
table recogniser (Tr)	text	table
single word recog (SWr)	words	single word
number list recogniser (NLr)	text	number list

Each recogniser program element has two nodes, a first node being settable and corresponding to the input of the recogniser, a second node being listenable and corresponding to the output of the recogniser.

Conveniently, said service program means has a first settable node, a listenable node for receiving data items received at said clipboard, and a callable node returning the results of a service.

Preferably, the graphical display means is a computer screen and a graphic window representing a desktop service is displayable, said window being clickable-on and being interactive with said user.

Conveniently, said window displays a plurality of services icons found by said software interface agent,

each said services being actuatable by said user by clicking on a selected icon.

Advantageously, said icons may include search engines for browsing the worldwide-web, user desktop items such as CD encyclopaedias and dictionaries and graphical display programs.

Conveniently, the software interface agent operates in real-time while the user is using the computer. Alternatively, the software interface agent may not operate in real-time; it may operate on a user's work product after it has been created.

Conveniently, software interface agent data recognition program means and said user service program means may operate in synchronous or asynchronous versions.

Advantageously, said software interface agent supports both synchronous and asynchronous operation and converts automatically between said versions.

Conveniently, said asynchronous versions are used in networked environments.

According to a further aspect of the present invention, there is provided a computer program product comprising:

a computer usable medium having computer readable code means in said medium for monitoring the activities of a user operating a control device and for providing to the user prompts or suggestions to allow the user to perform further activities using said control device, said computer program product having:

computer readable program code means for monitoring the activities of a user and for receiving data from the user's activities, computer readable program code means for comparing the data from said user's activities with a user definable database of data recognition elements and for providing outputs from said data recognition elements,

computer readable program code means for receiving

said data recognition element outputs and for executing a service function and obtaining the results of said service,

computer readable program and code means for causing said control unit to indicate to the user when the results of said service are available to the user to access.

Preferably, said computer program product is stored on a computer disk and said control unit is a PC-type, laptop or a palmtop computer. Alternatively, said computer program product is stored on any other form of electronically readable medium, such as a programmable read-only-memory integrated circuit (PROM) which may be located in a computer, TV control unit, cellphone or the like.

Preferably also, the computer program product is created at the user's location by downloading the software interface agent from the internet.

Preferably also, the computer program product includes computer readable code means for causing the computer to display a window or icon on a computer screen to indicate the results on said service, said icon being clickable-on by a user to display the results of the service.

According to a further aspect of the present invention, there is provided a control unit having a software interface agent loaded in the form of a computer program product, said computer program product having a computer usable medium having a plurality of executable program code means for monitoring the activities of a control unit user, executing at least one service based on data from the user's activities and for obtaining results of said service, and for providing an indication to the user that results have been obtained for inspection by said user.

Preferably, the control unit is a computer. Preferably, the computer program product is a disk;

either a floppy disk or a CD or delivered over the internet. Conveniently, the program code means includes display code means for displaying icons in a window display to the user, said icons being representative of said services performed by said software interface agent. Alternatively, the indication to the user is provided by an audio signal from audio signal generating means.

Preferably, said software interface agent operates in real-time as the user performs activities with the control unit or computer.

Alternatively, said software interface agent may operate on a user's work product not in real-time.

Conveniently, the software interface agent supports synchronous and asynchronous operation.

These and other aspects of the invention will become apparent from the following description, when taken in combination with the accompanying drawings, in which:

Fig. 1 is a schematic diagram of the system architecture of an embodiment of the software interface agent according to the present invention;

Fig. 2 is a diagrammatic representation of node interactions of one of the Qbits shown in Fig. 1;

Fig. 3 is a schematic diagram of a demand-driven currency exchange Qbit;

Fig. 4 is a schematic diagram of a data-driven currency exchange Qbit;

Fig. 5 is a screen display of an E-mail message which contains text, a table and a web page URL and an aQtiveDesk floating window with icons selectable by the user;

Fig. 6 depicts part of the E-mail message with a word selected and the aQtiveDesk window with additional icons;

Fig. 7 depicts part of the E-mail message of Fig. 5 with a table selected by the user and the floating window containing different icons;

Fig. 8 depicts a display of dancing histograms after

the histogram icon in the floating window is selected;

Fig. 9 depicts an Excel (trademark Microsoft) 3D chart which is displayed when the Excel icon in the floating window is selected by the user;

Fig. 10 depicts a flowchart of the sequence of events performed by the user in order to select the dancing histogram shown in Fig. 9;

Fig. 11 is a general flowchart of the sequence of events performed by a recogniser program used with the aQtiveDesk software interface agent;

Fig. 12 depicts a screen display when a user is using a Brainstorm software interface agent;

Fig. 13 is a flowchart of a sequence of events which is performed by the Brainstorm software interface agent when being run by a user;

Fig. 14 depicts a screen display of the results obtained using software interface agent called SiteStorm which is part of the Brainstorm software interface agent family and which is simulated using Macromedia space Dreamwaver (trademark), and

Figs 15a-15e are flowcharts of the sequence of events performed by the SiteStorm interface agent leading to the screen display shown in Fig. 4.

Reference is first made to Fig. 1 of the drawings which depicts the system architecture of the software interface agent. On the top level are a number of software interface agents or products, generally indicated by reference numerals 10a, b, c and d, which are connected to and sit on top of smaller software components which are computer programs 12a,b,c etc. in their own right, each such program being known as a Qbit (trademark), which interact with the overlying software interface agents 10a, 10b, 10c etc. and which may interact with each other, as will be later described. The Qbits 12 sit upon a software framework 14, known as aQtiveSpace (trademark), which produces context/sensitive applications from the Qbits 12. The aQtiveSpace program

14 is particularly suited for systems that dynamically reconfigure themselves as new Qbits are added. This ability to reconfigure is exemplified in the aQtiveDesk framework (described later) which automatically links together components (Qbits) based on the type of data they accept. Each Qbit program interacts with the aQtiveSpace software 14 which is built on top of the Java virtual machine 16. Each of the software interface agent products, such as aQtiveDesk 10a or Brainstorm 10b, comprises a selected set of Qbits which may interact with each other. As will be later described these Qbit programs can be written by any suitable third party developer taking advantage of the status event features and self-description features offered by the aQtiveSpace software framework.

Each Qbit 10 has a series of named nodes, each node acting like the named attributes and methods of an object but with some differences and additional semantics. The nodes act like plugs and sockets by which each Qbit 12 can be connected to its environment, i.e. software interface product 10a, b, the aQtiveSpace software framework 14, and to each other Qbit 12a,b,c etc.

Each node performs one or more of six kinds of interaction;

- Set - value can be given to the node (e.g. setting an attribute);
- Get - a value can be requested from the node (e.g. getting the value of an attribute);
- Call - the node can be called as the normal object method call;
- Listen - the node can give a value to a "settable" node;
- Give - the node can request a value from a "giveable" node, and
- Supply - the node can invoke a "callable" node.

The above six kinds of interaction can be classified in two ways, namely by data flow and by initiative

(control flow).

Firstly, dealing with data flow, in the case of Set and Give data flows into the node. In the case of Get and Listen, data flows out from the node. In the case of Call and Supply, the data flow is bi-directional.

Insofar as initiative interactions are concerned, in the case of Set, Get and Call, the control comes from outside the node (external initiative), another Qbit (or arbitrary Java code) has invoked the relevant Set, Get or Call method on the node. In the case of the Listen, Give and Supply interactions, the control comes from within (internal initiative) as the node invokes the appropriate interaction when it is ready. The internal initiative interactions correspond to "callbacks" which are found in many systems. They each have a means of establishing a connection to one or more nodes and then invoke these nodes when ready (in the reference implementation, a Java Node object has appropriate methods Listen, Give or Supply; the connection is established by passing the required node objects, which are to be invoked later).

The interactions can be matched in pairs as each internal initiative interaction has a corresponding external interaction. For example, a listenable node is given a settable node in its Listen method. It invokes the Set method on the node every time it is ready it is ready to donate a value. Table 1 set forth below depicts the node interaction characteristics:

Table 1 : Interaction Characteristics

Interaction	Data Flow	Initiative	Pair
set	in	external	
get	out	external	
call	bidirectional	external	
listen	out	internal	set
give	in	internal	get
supply	bidirectional	internal	call

The input and output data of each node, where relevant, are also typed (e.g. number, text, image).

The nodes of one Qbit can be connected to nodes of another Qbit where they are compatible, that is where they can function as complementary pairs and have compatible types. The node interactions are represented as a block as shown in Fig. 2 where the control flow runs from left to right, input is represented by a hole 20 waiting to be filled and output is represented as a peg 22.

In the representation shown in Fig. 2, two nodes are connectable if they have corresponding hole/peg combinations. Although not shown, the "shape" (in programming terms, the type of data they produce or consume) of the holes and pegs can be altered so that if there is interaction between a hole and a peg, then the shapes of the respective holes and pegs must also correspond.

The method by which the nodes are described means that Qbits can be dynamically connected together by other Qbits or program code. This is different from, for example, object oriented programming languages, where an object tends to have a lot of information about many other types of objects. Each Qbit may have internal structure, including other Qbits, but the external behaviour of each Qbit is very similar in that it can be fully connected to other Qbits in the way schematically represented in Fig. 2.

Although connections between Qbits can be

established statically when an application is configured, the "plug-and-play" nature of the Qbit programs means it is particularly straightforward to connect Qbits together and then disconnect them from one to another while a program is running. In Fig. 2, the right hand side interactions, Listen, Give and Supply are established by a method/function call on the node which is establishes the link.

Note that Qbits are a "plug-and-play" technology because of their external connectivity. Typically computer programs know about the sub-programs they invoke at design time. Although this is also possible for Qbits, they can be written so that many or all of their interactions with other bits are via their Qbit nodes. The semantic completeness of these nodes means that Qbits can be linked externally (i.e. by another program or Qbit) without previously knowing about each other. In the preferred implementations the Qbits nodes are also explicitly typed allowing automatic linking of related nodes. This is the technique used in the aQtiveDesk embodiment where nodes in the service and recogniser Qbits are invoked based on whether their types match those of the available data.

The interactions between Qbits is explained by the following example which deals with two variants of a pounds to dollars currency converter to see how the interactions described above work together. Each of the currency converters makes use of the same "exchange rate" Qbit which monitors online sources to obtain the current pounds to dollars exchange rate. The Qbits have one node called "rateNow" which is both gettable and listenable. The current exchange rate can be obtained by "get"-ting the node value or by registering a settable node with the listen part of the node. In the latter case, the listening node will be "set" if the exchange rate changes.

The first type of currency converter is a demand-

driven currency converter which is shown schematically in Fig. 3 of the drawings. The currency converter has two Qbits: a currency converter 30 and an exchange rate 32. The "currency converter" Qbit 30 has two nodes. The "convert" feature is a callable node 34, that is, if it is given an amount in pounds, the convert nodes returns the corresponding amount in dollars. The "rate" feature is a giveable node 36 and is where it looks to get the current exchange rate. As part of the configuration of the demand-driven currency converter, the "rateNow" node 38 of the "exchange rate" Qbit 32 is registered with the "rate" node 36 of the "currency converter" Qbit 30. When an external call is made to the convert node 34, the currency converter Qbit 30 asks its rate node 36 to obtain the value and it performs a "get" function on the "rateNow" node 38 which obtains the required value. Accordingly, the currency converter Qbit 30 is then able to complete the currency conversion and return the result of the "convert" call.

The second type of currency converter is a data-driven currency converter shown schematically in Fig. 4 of the drawings which has a currency converter Qbit 40 and an exchange rate Qbit 42. The converter Qbit 40 has a "convert" node 44, just like the converter 30, but instead of the giveable "rate node" it has a settable "rate node" 45. This node can be linked to the exchange rate Qbit 42 but using a slightly different method: this time it is the "rate" node that is registered as a listener to the "rateNow" node. When the exchange rate Qbit notices a change in the current rate, it checks to see if there are any registered listeners and, if so, does a "set" function on each listener with the new rate. The currency converter is then responsible for keeping an internal copy of the value associated with the "rate" node. When the data-driven currency converter is next asked to perform a conversion, it simply uses this same value of rate knowing that it is up-to-date.

Both the demand-driven currency converter and the data-driven currency converter perform the same function, that is they perform a conversion that is correct with respect to the current status of the exchange rate. The variants allow different ways of obtaining up to date status information. These examples show that :

1) the software framework is equally good at managing demand-driven and data-driven actions;
2) the same software components, for example, an exchange rate Qbit, can be used with either variant, and
3) the Qbits can be combined together dynamically without Qbits necessarily "knowing about each other".

As well as data that is passed in an interaction, a "context" object can also be passed. The initiator of interaction supplies the context object. In a case of set, get and call, this context object is passed to the node when the interaction is invoked. In the case of listen, give and supply, a context object is passed through the node when the relationship is established. For example, node A is "listenable" and node B is "settable". Some external code, for example part of node B Qbit, but possibly completely external, creates a context of object C, then establishes a listen relationship with the method such as:

A.listen (C,B)

Later A has data (D) ready and evokes B's set interaction with code like:

B.set (C,D)

Node A passes the context on so that node B can, for example, match the "set" with a corresponding "listen".

These context object transfers can be used to establish "conversations" for several independent interactions and can be regarded as part of a more protracted pattern. In particular, "call" can be regarded as a simple conversion "set" followed by the

corresponding "get", which has been included as a primitive for convenience.

With regard to the nodes described above, it will be understood that all node interactions may have both synchronous and asynchronous versions. For example, one can invoke the "set" method and wait for the method to return indicating that the "set" was successful (synchronous). Alternatively, a variant can be used that establishes the request to set the node but allows the "setting" Qbit/code to continue to execute (asynchronous). The asynchronous version of "set" can be thought of as a kind of "fire and forget" node. For "call" and "get", where a return value is required, a "callback" can be registered for the value when it is ready. This may be a "settable" node, or a simpler callback object, depending on the implementation.

Asynchronous interactions are particularly useful in networked environments where there may be considerable delays if Qbits are located in different parts of the network. The current reference implementation allows either form of interaction, that is synchronous or asynchronous, and automatically converts between synchronous and asynchronous versions as described later.

The current referenced aQtiveSpace implementation is coded in Java and is particularly well suited for certain aspects of the implementation. However, it is not a necessary part of the aQtiveSpace software framework: it is possible to have the aQtiveSpace software implementation built over other platforms, such as C++.

The main parts of the aQtiveSpace interface are the Qbit interface and the node interface and these are further explained in detail below:

The Qbit interface defines four methods:

String getName()
get the name of the Qbit (optional)

Node getNode (String name)
get the named node

Node[] getNodes(nodeTypeSpec spec)
find all nodes of a particular type

Node getNode(String name, NodeTypeSpec spec)
get the node and verify its type

The "getNodes" method makes it possible to discover the interaction possibility of a Qbit without knowing the names of its nodes beforehand (a reflection mechanism).

The Node interface is more complicated, as it includes methods corresponding to all the interaction kinds:

First of all there are several methods to get the name, parent Qbit and type of the node:

String getName()
Get the name of this node..

Qbit getQbit()
Return a reference to the Qbit that this node is part of.

NodeType getType()
Get the type(input type, output type and interactions) of this node.

boolean is(Interaction inter)
Says whether the node can perform the required interaction. The parameter inter can be either a specific interaction kind such as GIVE, or SET, or can be a combination interaction kind such as:
"SET" and (GET)".

Then there are methods for set, get and call, with a synchronous and asynchronous version of each.

void set(Context, SyncSetOptions, Data)
A synchronous version of set. This method will wait until the set has happened or an exception is returned.

void set(Context, SetOptions, Data, SetManager setter)
An asynchronous version of set. Any exceptions are returned to the setter object at some later point in time. This function returns immediately. The setter object may be null if a result is not expected or needed.

Data get(Context, SyncGetOptions)
A synchronous version of get.This method will block until the data is available.

void get(Context, GetOptions, GetManager getter)
An asynchronous version of get. The results of the get are returned to the getter object at some later point in time. This function returns immediately.

Data call(Context, SyncCallOptions, Data)
A synchronous version of call. This method will block until the parameter data has been sent and a result is returned.

void call(Context, CallOptions, Data, Call Manager)
An asynchronous version of call.The results of the call are returned to the caller object at some later point in time. This function returns immediately.

void listen(Context, ListenOptions, Node listen, ListenManager)

This method sets up a listening dependency between nodes. When an event occurs, a set message will be sent/set method will be called on the node listener. This method is essentially an event registration.

void given(Context, GiveOptions, Node giver, GiveManager)

This method sets up a giving(supply) dependency between nodes. When a value is needed by this node it will get it from the giver. This method is to get()what listen() is to set().

void supply(Context, SupplyOptions, Node supplier, SupplyManager)

This method sets up a supply(need) dependency between nodes.When a function is required by this node it will call the supplier. This method is to call()whatgive() is to get().

The Java implementation of aQtiveSpace uses the two different types of Java method for each node type to automatically convert between synchronous and asynchronous versions of the nodes. For example, suppose a particular Qbit supplies a synchronous version of a gettable node:

```
Data get(Context, SyncGetOptions)
```

The aQtive space architecture uses Java's built-in inheritance mechanisms to supply a asynchronous stub similar to the following skeleton code:

```
void get(Context context, GetOptions options, GetManager getter)
{
    Thread thread = new GetThread(this,context,options,getter);
    this.start();
    return;
}
```

where the GetThread class is like the following skeleton code:

```
class GetThread
{
    public GetThread (Node node,Context context, GetOptions options, GetManager getter)
    {
```

```
            // code to save parameters in Java instance
    variables
        }
        void run()
        {
            Data val = node.get(context,options);
                // N.B. possibly long wait for synchronous
    get to complete
            getter.setResult(val);
        }
    }
```

Similar translations are performed for the other node types.

Finally, there are methods to cancel a part-finished interaction (asynchronous) and to deregister listeners etc.

```
void cancel(Context, CancelOptions)
    Method that allows a client of the node to quit the
    current interaction.
void unlisten(Context, UnlistenOptions)
    This method removes a listening dependency between nodes.
    This method is essentially an event deregistration.
void ungive(Context, UngiveOptions)
    This method removes a give(supply) dependency between
    nodes.This method is to get() what unlisten() is to set().
void unsupply(Context, UnsupplyOptions)
    This method removes a supply (need) dependency between
    nodes.This method is to call() what ungive() is to get().
```

A specific embodiment of the invention will now be described with reference to Figs. 5 to 10 of the accompanying drawings. The embodiment is a desktop application using a software interface agent hereinafter referred to as aQtiveDesk (trademark) which is the internal development name of the product to be released commercially under the name "onCue".

The aQtiveDesk software interface agent will first be described with reference to series of actions performed by a user and then it will be explained how the software interface agent monitors the users actions and then prompts the user via icons on a displayed window to

take further action.

Reference is first made to Fig. 5 of the drawings which depicts a computer screen display 50 on which is an E-mail message from a colleague. The E-mail message contains text 52, a table 54 laid out with spaces and a URL of a web page 56 in the text. When the aQtiveDesk software agent is active, a small floating window 58 appears on the display with icons to allow the user to obtain help about the software interface agent and to set preferences etc.

As the user scrolls through the E-mail message, either using keys or a mouse, the user selects the word "histograms", as shown in Fig. 6. When this word is selected (shown inverted), the aQtiveDesk window 58 changes. Several new icons appear suggesting to the user several possibilities which the user may wish to do with the words "histograms". For example, the aQtiveDesk software interface agent suggests looking up "histograms" in various online search engines: Ask Jeeves (trademark) 60, HotBot (trademark) 62, AltaVista (trademark) 64 and Yahoo (trademark) 66. It also suggests looking up an online thesaurus 68 and dictionary 70. Finally, it also suggests looking up the words "histograms" in the user's CD copy of Encyclopaedia Britannica (trademark) 72.

The user decides to select the thesaurus and clicks on the thesaurus icon 68 and the aQtiveDesk software interface agent then launches a web browser program and directs the web browser program to a thesaurus service which then returns a web page listing similar words such as "charts", "diagrams" etc.

The user then decides to select the table 54 in the E-mail message. This is illustrated in Fig. 7 of the drawings with the table 54 shown inverted. When the table is selected, the aQtiveDesk window 58 changes again and displays different icons. This time, search engines are not suggested. Instead, the aQtiveDesk software agent suggests three desktop programs; dancing histograms

74, a calculation program called sumIt! $\sum$+ 76 and Microsoft Excel (trademark) 78. The user views the suggested icons and then selects the dancing histograms icons 74. The aQtiveDesk software interface agent launches the dancing histogram application which is displayed in Fig. 8 as a window 80 on top of the E-mail message displayed. It will be seen that the dancing histogram 74 represents graphically, and in an animated form, the data contained in the table 54 in the E-mail message. The user then selects the Excel icon 78 and in presses down a mouse button (not shown), to reveal a menu of possibilities of things to do with Excel (not shown), including drawing a 3D chart. The user selects the 3D chart and the aQtiveDesk software interface agent responds by opening the Excel program and pastes the table data into a new worksheet 82 shown in Fig. 9 and then instructs the Excel program to draw the 3D chart 84 which appears within the worksheet 82. The user just has to watch the 3D chart 84 appear.

Thus, it will be seen that the aQtiveDesk software agent monitors the activities of the user and creates prompts in the aQtiveDesk window 58 for the user to select to expand the scope of the user's activities in a certain area. This can be done either online by generating a web browser or offline by activating the user's peripheral, such as a CD drive.

The sequence of operations will also be described with reference to the flow chart shown in Fig. 10 of the drawings which explains how the aQtiveSpace software interface agent 14 interacts with the aforementioned Qbit programs to generate the further services for the user which are displayed by the icons. This will now be explained with reference to what happens when the user selects the table shown in Fig. 7 of the drawings.

Firstly, the aQtiveDesk software agent 14 is selected by the user. The agent 14 loads a collection of specific Qbit programs, some of which are integral to the

aQtiveDesk product, such as a clipboard watcher Qbit 90 that monitors for the user's cut/copy actions, an aQtiveDesk window Qbit 92 for displaying aQtiveDesk's suggestions to the user in the form of icons in the window, a web browser Qbit program which is used to send the default web browser, for example NetScape, Navigator or Microsoft Explorer, to a selected URL. Other Qbit programs are optional and a configuration file is used to record which Qbit programs require to be loaded. The user can modify this set. The optional Qbits are of two kinds: recogniser Qbit programs which use simple heuristics and artificial intelligence (AI) to calculate what has been copied to the clipboard, and services which encapsulate things which can be suggested by the user. In addition to all of these Qbit programs are the code for the aQtiveSpace underlying software component infrastructure and the aQtiveDesk software framework which is the code constructed on top of the aQtiveSpace and which brings together further aQtiveDesk components.

In one sense the aQtiveDesk framework is simply another software component but this component acts as the "glue" between the other Qbit programs orchestrating the efforts and activities of these Qbit programs. In addition, the other Qbit software programs are written in special patterns to enable the aQtiveDesk framework to link them together. The particular patterns used in this embodiment include specially named nodes for the Qbits (depending on their function within the framework). For recognisers this includes TryRecognise (settable) and Recognise (listenable) and for services TryProvide (settable) and Provide (listenable). The "glue" perfomed by the framework is the particular linking together of these depending on the types of their nodes as described in the example below. Java code for a sample recogniser Qbit and service Qbit is given in appendix A.

Each of the services is a Qbit program which has a data type which it can accept. In this example, the

following services correspond to certain data types:

Service	Type
Histogram	Table
Services Encyclopaedia	Words
Thesaurus	Single Word
SumIt!	Number List
Excel - Qbit	Table
Web Searches	Words

Each recogniser is a Qbit program which has a type which it looks at (in-type) and a type it recognises (out-type), as will be later described.

Recogniser	**in-type**	**out-type**
words recogniser (Wr)	text	words
table recogniser (Tr)	text	table
single word recog (SWr)	words	single word
number list recogniser (NLr)	text	number list

In use, when the aQtiveDesk software interface agent is selected, it generally sits in the background. Only the clipboard watcher Qbit program 90 is active which waits for a copy or cut to happen. When the aQtiveDesk framework initialises itself, it establishes itself as a listener with the clipboard Qbit 90, all the recogniser Qbits 92 recognise nodes and all the service Qbits 94 provide nodes.

When the user selects and copies the word "histograms", the clipboard watcher Qbit program 90 notices and passes the copied text to the aQtiveDesk framework. The clipboard watcher Qbit 90 sets the node used by the aQtiveDesk framework when it is registered with the Qbit 90. The aQtiveDesk framework then looks for Qbit recogniser 92 or service Qbits 94 that can use the text. The general form of a recogniser 92 is shown in Fig. 11. The Java code for `MiniFigureRecog` in Appendix A can be seen to follow this flow diagram. The recogniser Qbits Wr, Tr and NLr are all activated. Each recogniser must have two nodes: TryRecognise 140 and Recognise 142. The recogniser may have other nodes as well, but these are ignored by aQtiveDesk framework. The TryRecognise node is settable and corresponds to the input of the

recogniser. The Recognise node is listenable and corresponds to the output.

In this example, because it is only text that has been selected, both the Tr and NLr recogniser Qbits fail to recognise the text (it is neither a table nor a list of numbers) but the word recogniser Qbit (Wr) does recognise the text. The aQtiveDesk framework looks at the type of each recognisers' TryRecognise node. If any match it "sets" their TryRecognise nodes (box 144). Some recognisers may instantly be able to check the text, others may need to consult local or network resources (box 146). If the recogniser does not recognise the text it does nothing. If it does recognise it, it tells all Recognise listeners (box 148), in particular the aQtiveDesk framework which matches the revised data type against all recognises' TryRecognise nodes and sets the TryRecognise node of any that match; these may again recognise the data etc. The Wr simply looks at the text and decides whether this text could be considered a sequence of "words". It clearly can and so announces to the aQtiveDesk that the text can be regarded as words. The aQtiveDesk software interface agent records this (box 96) and because the aQtiveDesk now knows the selected text is words, it activates the single word recogniser Qbit 92a which is based on matching the in-type of single words (SWr). At the same time, the aQtiveDesk software program activates the web search service Qbits 94a and also the CD-ROM encyclopaedia 94b because these services just expect words. Because the text is recognised to a collection of words, it involves the services posting back (line 98) to the aQtiveDesk interface agent, data structures that make it easy to view the text as a series of words.

The single word recogniser recognises that the words are, in fact, a single word as there is only one of them and it announces this back to the aQtiveDesk framework 96 which activates the Thesaurus and Dictionary services as

they require a single word each.

Finally, the aQtiveDesk interface agents creates and displays icons of all the aQtiveDesk services in the aQtiveDesk window on the user display.

Each service Qbit has one or more of the following: A settable node called TryProvide and corresponding listenable node called Provide - used for services that return new data items to be copied to the clipboard, for example, the SumIt! Service that adds together selected numbers and copies back the result.

Any other settable node - used for services that perform actions, for example, the histogram that launches a window.

A callable node returning a result of type URL - used to call a browser, for example, the Thesaurus service.

The aQtiveDesk framework also keeps track of which of the converted data is relevant for each particular service. The exception to this is the TryProvide node which the aQtiveDesk framework sets there and then. The service Qbit can then announce zero, one or more copyable data items using its Provide node (in a similar fashion to recognisers). In particular, this will include the aQtiveDesk framework which records the copyable data for later use and marks the service as active.

Depending on how fast the recognisers operate, this process may finish before the user has time to act, or if some of the recognisers are slow, extra services may be added even as the user interacts with the aQtiveDesk window.

It will be understood that the selection of services offered depends dynamically on the kind of data selected by the user and also it will be understood that the recognition of the type of data may take several steps; for example, text is recognised as words which, in turn, may be recognised as a single word.

When the user selects an icon in the aQtiveDesk

window, the aQtiveDesk framework program examines the settable or callable nodes of the service whose input types match the available data. The aQtiveDesk framework treats services that generate a URL for the browser specially because URLs are so common. It asks the service for the URL and then the active framework passes this URL to the browser.

The above explanation of what happens when a word is selected is similar to what happens to when the table 54 is selected by the user as is shown in Fig. 10. When the table 54 is selected initially, all the aQtiveDesk framework knows is that it has seen more copied text. This copied text (the table) is passed to the same three Qbit recognisers 92a,b,c for processing: Wr, NLr and Tr. This time the word recogniser Qbit Wr, fails to recognise the table because it is too long, split over several lines or it has too many numbers. However, the number list recogniser Qbit 92b (NLr) recognises that the table because it ignores other words and looks for any numbers in the data. The table recogniser Qbit 92c (Tr) also recognises the data as a table.

This time there are no repeat runs through the recogniser Qbits because none of the recogniser Qbits can deal with tables and number lists; the recogniser Qbits can just generate them. However, two services Qbits 994c,94d require tables (histograms and Excel) and one requires a numbers list. These three services are then activated and presented to the user as icons 74,76,78 in the aQtiveDesk window 58.

In this case, the recogniser Qbits 92 and the data structures they post back are more complex. For the table, this includes: the title of table (where present); the number of columns; the number of rows; column labels (where present); row labels (where present), and numerical table data. However, the same principles of operations by the recogniser Qbits hold as for the simple word recogniser Qbits. The text is recognised as having

a certain form and the fact that the text does have a certain form, together with the transformed data, is "announced" by the recognisers to all Qbits that can use this type of data.

In this example, when the user selects the histogram icon 74, the aQtiveDesk framework software program receives the user selection 99, sets the relevant node, simply passes the table data to the histogram Qbit and requests that the histogram Qbit 99 uses this as a signal to create its window etc. This histogram Qbit has only a single action to perform which is to set the data structure, codifying the table data and displaying it as an interactive histogram 74 in window 80. This Qbit program is a sort of mini-application which runs entirely within the aQtiveSpace framework.

When the Excel icon 78 is selected, the activity is different. Whereas the histogram Qbit only has a single action to perform, there are several possible actions that the Excel Qbit can perform on the table data, including pasting the table into an Excel spreadsheet and using variants of the Excel charting functions. The user therefore chooses the appropriate actions from a menu (not shown) that drops down from the icon. Only actions which are appropriate for the table data are suggested and revealed in the dropdown menu. When the user selects the 3D chart 84 (Fig. 9), the Excel Qbit starts the Excel program, if not already started, and then remotely controls the Excel application to produce the 3D chart. The user needs to perform no further interaction with the display or keyboard/mouse to create this chart. In this case, it will be understood that the Excel service Qbit program acts as a wrapper or controller for the existing application on the user's desktop.

As previously noted, one of the advantages of the invention and the aQtiveDesk embodiment is the ability to easily extend it with Qbits developed either by the

inventors and third-party developers. The code samples in appendix A show how Qbits can be written in Java to fit into the aQtiveDesk framework. In the prefered embodiment, these Qbits can then be easily added into the aQtiveDesk by inserting the Java class files into particular locations and updating configuration files.

In addition, simple Qbits can be added to the aQtiveDesk embodiment using XML files. Appendix B shows examples of such XML Qbits.

Reference is now made to a second embodiment of the present invention in which a software interface agent product 10b, known as Brainstorm (trademark), as shown in Fig. 1 is used with the Qbits 12 and aQtiveSpace software framework 14 which sits on top of the same Java virtual machine 16. A description of the operation of Brainstorm will be given with reference to Figs. 12 and 13 of the drawings.

Fig. 12 depicts part of a desktop display 100 running a word processing package. The user is a pupil doing homework, for example on the French Revolution. The Brainstorm software interface agent 10b is active and running in the background. As user types, text is displayed by the word processing document. The Brainstorm interface agent 10b simply appears as an unobtrusive icon 102 on the edge of the desktop display 100.

As the user types, the user notices that the Brainstorm icon has become highlighted. The user knows that this means that the Brainstorm interface agent 10b has found something useful in relation to the word processing product and accordingly the user clicks on the Brainstorm icon.

When the user clicks on the Brainstorm icon, a window 104 opens which lists a selection of web pages which are relevant to different parts of the essay on the French Revolution. It will be seen from the window 104 that some of the pages 104a,b have already been

downloaded so the user can view them immediately.

Reference is now made to Fig. 13 of the drawings which depicts a flowchart of the sequence of events being performed by the Brainstorm software interface agent 10b.

When the Brainstorm software interface agent 10b is launched, it uses the user's personal profile to select appropriate local and Internet resources, including search engines and subscription services. As the user types the essay into the word processor, the Brainstorm interface agent Qbit 106 monitors the current document. It extracts the text (box 108) and strips out common words using appropriate keywords for the document as a whole, each paragraph, each sentence, each phrase and the user's current words (box 110). The keywords which are extracted are used to construct searches (box 112) which are submitted to the resources 114 which may either be local or remote. If the search for a phrase, sentence or paragraph returns too many results, keywords from the surrounding text are added to reduce the items to more relevant ones.

When the Brainstorm software interface agent 10b has begun to receive sufficient results (box 116), it highlights the Brainstorm icon 102 to show to the user that the results are available. The Brainstorm interface agent 10b continues to submit further searches as the user types more text. It also starts to retrieve the most relevant documents/pages which are stored (box 118) so that they are instantly available to the user and they are also used to further check the relevance of pages and to ascertain which part of the document to which they are most relevant. The resulting pages/documents from the search results are retrieved and further filtered and sorted (box 120). When the Brainstorm icon is clicked it opens the window 104 which lists the most important sentences, phrases etc. against the search results as shown in Fig. 12 and the user can select these results for display (box 122).

Reference is now made to further embodiments of the present invention relating to further software interface agents or products; SiteStorm (trademark) and DeskStorm (trademark) which are two additional software interface agent products from the Brainstorm family. The discussion on the preceding pages refer to the key agent of the Brainstorm family which works alongside a Word processor which is referred to below as Wordstorm to distinguish it from the Brainstorm family in general. Each of these products takes, as an input, a collection of documents (SOURCE): in the case of SiteStorm, the input is web pages in a web site, and in the case of DeskStorm, the files on the user's machine, i.e. on a disk. SiteStorm/DeskStorm then extracts keywords and other indexing information from the SOURCE and uses this information to search a selected remote document archives including the web (TARGETS). The most appropriate documents from this search (the RESULTS) are then organised using the structure of the SOURCE and can be displayed alongside the SOURCE. Whereas Wordstorm and aQtiveDesk operate in real-time as the user works, SiteStorm and DeskStorm in the background building up and elaborating the RESULTS, while the user is working on other topics.

The following description relates to the SiteStorm software interface agent/product which operates in accordance with the sequence of events in Figs. 15a-15e resulting in a screen display of the type shown in Fig. 14. A user opens the SiteStorm agent and selects his web site as the SOURCE.

The user leaves the SiteStorm agent to work in the background and continues with day-to-day work on the computer. When the user is not connected to the Internet, the SiteStorm agent is quiescent but when the user is connected the occasional whirr of the hard disk is noticed.

Some time later the user opens the SiteStorm agent

(or a site editing tool with SiteStorm support) and the user views the web site. Against each web page are listed potential remote web pages or other documents that may be relevant. When pages are arranged in groups, for example in a directory hierarchy, suggestions are also made for the group as a whole.

The results of the SiteStorm agent search are displayed to the user in a window 130 which appears on the user's desktop. Although the result is superficially similar to online indexes such as Alexa (trademark), SiteStorm's results are subtlely different. Instead of looking at each page of the SOURCE independently for related TARGETS document, the whole SOURCE is taken into account. Whereas a simpler search using keywords for the individual pages would have given the same or similar results for related pages within the SOURCE because they cover a similar overall area, SiteStorm's contextual features mean that different, more specific pages are offered.

Firstly, SiteStorm records the chosen site and if there is not a local copy of the site SiteStorm, may, optionally, download the site for faster subsequent processing. SiteStorm also makes use of the structure of the site. This it can either build itself (using the directory and link structure of the site and clustering in step 2 below) and/or use a user selected structure as shown in Fig. 15b. Normally, the latter would be in conjunction with a site management tool.

SiteStorm then performs the following steps:

1. SiteStorm examines each page in the SOURCE and determines keywords and other indexing items for the page (Fig. 15c). This stage uses lists of common words to remove "noise" words and may also use standard indexing techniques such as stemming.
2. If required SiteStorm uses these keywords to build clusters of SOURCE pages (Fig. 15b).
3. SiteStorm uses the clustering and other structure of

the site to classify and weight keywords using standard weighting techniques (Fig. 15c). If a keyword/indexing item is common to many pages in a cluster/related part of the site, then it is allocated to the cluster/directory/index page and given a lower weighting/priority for the individual page. This is because these keywords are deemed to denote the topic as a whole, as opposed to the specific sub-topic of the page. For multi-level hierarchies/clusters, this process is continued for higher levels also. Where keywords are determined for a whole cluster/group of pages, then they may also be added (with low weight) to all pages within that cluster/group (Fig. 15c). This is to ensure that a page with a single image and a title "The Guillotine" will have different suggested pages when it is found in the context of an office equipment site than it would if in a site on the French Revolution.

4. The keywords are used to initiate searches in the TARGET (web searches and other repositories). Depending on the nature of the TARGET the keywords and other index items will be transformed appropriately to make valid "queries" or searches. The returned pages are the initial candidate RESULT set of documents (Fig. 15d).

5. The RESULT documents are allocated to the closest matching page(s) in the SOURCE. However, where a RESULT document is equally close to several pages within the same cluster/group, it is allocated to the cluster rather than the individual documents (Fig. 15d).

6. Local matching of RESULT documents against their related SOURCE pages may lead to the removal of some pages.

7. The RESULT set may also grow. Hyperlinks may be followed for web documents leading to new documents

which can be matched locally against keywords. These additional pages can then be allocated as in step 5. Also, if the set of RESULT documents associated with a particular SOURCE page all contain similar words, then these may be added as tentative additional keywords for that SOURCE page and the process from step 2 on may be repeated using this larger set of keywords.

8. Parts of this process are continually repeated and reviewed in the background as pages in the SOURCE are updated (leading to different keywords) and as the TARGET repositories change.

The appropriate RESULT documents are then listed against each SOURCE document (Fig. 15e). Depending on the kind of site display and on user options, this list may also contain the RESULT documents for associated clusters/groups, suitably marked.

Various modifications may be made to the embodiments hereinbefore described without departing from the scope of the invention. For example, although the software interface agent is described as being usable with a computer, it will be understood that the software interface agent may be used with other control units, such as a TV remote control, a cellphone of the like. The results obtained by the program may be indicated to the user visually as in the case of a graphic icon or audibly by a beep, or by a tactile indication in the case of a cellphone, that the results had been obtained.

It will be appreciated that a number of similar products may be obtained based on the same concept and using different combinations of Qbits which are settable by a user. For example, a software interface agent may include a four function on-screen calculator so that when the user selects a number, the aQtiveDesk software interface agent suggests adding it to the calculator. Thus, as the user moves around a set of documents selecting numbers, these are automatically added to a

running total. The calculator also recognises arithmetic expressions so that when the user selects an expression in a document, the aQtiveDesk software interface agent suggests passing it to the calculator to evaluate. The calculator may be a stand alone calculator or may enhance the basic aQtiveDesk software interface agent. The user can customise the calculator keypad to include its own or third party functions by selecting appropriate Qbits to perform numeric functions, such as business or scientific calculations.

Other software interface agents may be specifically directed to web-based products. For example, a software interface agent to let the user look ahead to summary of pages and a further software interface agent which manages access to local and remote mirrors of frequently used sites. Other software interface agents may provide interactive visualisations of web sites and also guide users round key features of web sites. All of the products use a common server-site web map on activated sites which is linked to summaries, providing update information and providing data visualisations. Such software interface agents will include some of the features described above with reference to the aQtiveDesk and Brainstorm implementations.

The users of the software could be any person or organisation who use a computer to access information or resources, such information or resources usually residing on the computer used by the user and/or on a network that the computer may be connected to and/or on the internet. There is a large potential market for internet use. There is a large potential market for corporates who wanted to use the software on their intranet and/or extranets, providing the advantage to users that it can suggest the "right thing at the right time" (see 'Customer Code' example in Appendix B). Because the software can suggest things to the user, there is a market for it as an advertising and marketing tool,

allowing marketers and advertisers to offer targetted content to people. Websites and portals can utilise this feature to retain users by suggesting aspects of their site to users. The software technology can have many alternative instantiations, each offering a different targetted product to a specific market, but in every case the software acts as a broker between the information and the user, providing faster and easier access to that information for the user.

Advantages of the invention are that use of the status/event analysis concept enables the creation and use of observeable methods and observeable data. This means that it makes it easier to program or produce systems which observe and react to the electronic and physical environment. The use of self-describing objects allow the typed inspection of methods, data and particular events. Because of the modular nature of the program elements, the Qbits, collections of many applications can be provided which share common components and these common components can be combined to form specific customised applications, either by a provider or by an end-user. The use of modular components allows a wide product range to be implemented, although it may be provided in three main categories: web-based products; end-user desktop products as well as corporate productivity products. The software interface agent can be used synchronously or asynchronously, although asynchronous use is considered to be essential for networked applications. In addition, the software interface agent has the advantage of being used in real-time, that is when the user is actually working on a document or is actively using a control unit and receives a response from the software interface agent that the user recognises as being related to their current work, such response time typically but not necessarily being less than ten seconds. This timing is achieved on any computer system that fulfils or exceeds the minimum

requirements of 386 processor or equivalent, 4MB RAM, running any operating system typically but not exclusively Windows (3.1, 95, 98, NT), Macintosh OS, Linux, Solaris, Unix. In addition, the software interface agent can also be used in non real-time; that is, after completion of a task, the interface agent can peruse a work product so that when the user then next activates the control unit or computer, a list of prompts or suggestions can be provided. The software interface agent has the advantage of being suppliable on a floppy disk or CD drive or it can be embodied into a specific chip or integrated circuit within a control unit or a computer or by an internet download. A further advantage is that the user can set or define the components which are required for a particular interface agent, thus giving a wide variety of applications. Other advantages are that the software interface agent can react in ways that are appropriate to a user's current actions and/or data and the software interface agent can integrate data from a wide variety of sources, both from the user's computer and from a local or global network. A further advantage is that a software interface agent does not interrupt the user, even in real-time activities, and provides a non-intrusive prompt that the user may wish to take advantage of to further extend enquiries or investigations.

APPENDIX A

This appendix includes the Java code for a simplified version of the SumIt! Qbit. It consists of three parts:

MiniNumberList

A Java class for the type of data recognised.

MiniFigureRecogniser

A java class for the recogniser Qbit written in the recogniser pattern. It looks at the user's copied text, checks whether it is a list of numbers and if so converts them into a MiniNumberList object.

MiniSumIt

A Java class for the service Qbit written in the service pattern for copying back data into the clipboard.

```
//-------------------------------------------------------------
//------- MiniNumberList

package com.aqtive.user.minisumit;

import java.util.Vector;
import java.util.StringTokenizer;

public class MiniNumberList
{
    protected double[] figures;
    public DoubleList(double[] figures)
    {
        this.figures = figures;
    }
    public MiniNumberList(String str)
    {
        Vector vfigures = new Vector();
        StringTokenizer toks = new StringTokenizer( str, "\t\n\r " );

        while ( toks.hasMoreTokens() )
        {
            String tok = toks.nextToken();
            if ( tok.length() == 0 )
                continue;
            try
            {
                vfigures.addElement( Double.valueOf( tok ) );
            }
            catch ( NumberFormatException e ) {}
```

```
        }
        figures = new double[ vfigures.size() ];
        for ( int i = 0; i < figures.length; i++ )
            figures[ i ] = ( (Double)vfigures.elementAt( i )
).doubleValue();
    }

    public int length()
    {
        return figures.length;
    }
    public double sum()
    {
        double res = 0.0;
        for ( int i = 0; i < figures.length; i++ )
            res += figures[i];
        return res;
    }
}

//-----------------------------------------------------------------

//-----------------------------------------------------------------
```

//------- MiniFigureRecogniser

```
package com.aqtive.user.minisumit;


import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
import com.aqtive.qbit.*;
import com.aqtive.qbit.pattern.*;

public class MiniFigureRecog implements Recogniser1Pattern
  {
  public final static Type outRecognise = Type.forClass(
MiniNumberList.class );
  public Listen listenRecognise; // Recognise is a listenable node

    /**
     *  aQtive Desk sets this node if the clipboard data is
     *  recognised as a String (basic text type)
     */
  public void setTryRecognise( String string )
    {
        MiniNumberList nl = new MiniNumberList(string);
        if ( nl.length() > 1 )  // singe numbers or non-numbers not
recognised
        {
            listenRecognise.set( Data.newData( nl ) );
                //  aQtiveDesk will take this data as 'recognised'
                //  and pass it to other Qbit recognisers or services
                //  that match this type
```

```
                // In particular, the service MiniSumIt will be
activated

            }
        }
    }

//-------------------------------------------------------------------

//-------------------------------------------------------------------
//-------- MiniSumIt

package com.aqtive.user.minisumit;


import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
import com.aqtive.qbit.*;
import com.aqtive.qbit.pattern.*;
import com.aqtive.desktop.Config;


public class MiniSumIt implements Service1Pattern
    {
    public final static Type outIcon = Type.forClass( Image.class );
    public Data getIcon()  // icon displayed by aQtiveDesk for this
service
        {
        return Data.newData( Config.getImage(
"com/aqtive/user/minisumit/minisumit.gif" ) );
        }
    public final static Type outHelp = Type.forClass( String.class );
    public Data getHelp()
        {
        return Data.newData( "Sum the numbers" );
        }

    public final static Type outProvide = Type.forClass( String.class
);
    public Listen listenProvide;  // Provide is a listenable node

        /**
         *  aQtive Desk sets this node if the clipboard data is
         *  recognised as a MiniNumberList
         */
    public void setTryProvide( MiniNumberList figures )
        {
            double val = figures.sum();
            int len = figures.length();
            listenProvide.set( Data.newData( ""+val ), "sum" );
            listenProvide.set( Data.newData( ""+(val/len) ), "average" );
                    // aQtive Desk will put these in the copy back menu
for MiniSumIt
        }
```

]

//----------

5

APPENDIX B

XML Qbits are an easy way to specify simply Qbits. This appendix includes the definitions of four complete example XML Qbits:

Pet Lovers Heaven

Recogiser for fixed keywords and associated service taking user to fixed web page.

HCI Book Search

A Qbit service that links to a web search engine.

Customer Code

A Qbit recogniser for customer codes as may be found in a corporate Intranet and associated service linking into relevant Intranet resources.

Surelynot Date

Regular expression recogniser and associated service for date-based web service..Demonstrates multiple menu items for service and URL using component parts of a recognised type (day, month, year).

XML Qbits can be combined with Java Qbits in the aQtiveDesk embodiment. XML Qbits can be used to produce recognisers or services.

Many of the Qbits in the commercial version of aQtiveDesk (onCue) are written using the XML Qbit API.

```
//-------------------------------------------------------------
//------- Pet Lovers Heaven
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE Qbit PUBLIC
"-//aQtive//Qbit Specification V1.2Beta//EN"
"http://www.aQtive.com/dtd/qbit_v1.2Beta.dtd">
<!-- Keyword Qbit Template -->
<!-- Copyright aQtive limited 1999 -->
<Qbit>
```

```
<!-- Recogniser Bits -->
<Recogniser>
<Name>Pet Lovers Heaven Recogniser</Name>
<Recognises>
<Match Type="Keywords">dog, goldfish, rat</Match>
</Recognises>
<!-- N.B. Recogniser "Is" should match Service "For" -->
<Is Type="User" Name="Keywords for Pet Lovers Heaven">
<Field Name="Matched" Expand="Yes">$0</Field>
</Is>
</Recogniser>
<!-- Service Bits -->
<Service>
<Name>Pet Lovers Heaven</Name>
<Icon>user/images/rat.gif</Icon>
<Author>Paula Elizabeth Tomkins</Author>
<Help>A page with all sorts of things about pets</Help>
<For Type="User" Name="Keywords for Pet Lovers Heaven">
<URL Label="pet lovers page">http://www.surelynot.com/pet-lovers/pet-
lovers.html</URL>
</For>
</Service>
</Qbit>
//-----------------------------------------------------------------

//-----------------------------------------------------------------
//------- HCI Book Search
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE Qbit PUBLIC
"-//aQtive//Qbit Specification V1.2Beta//EN"
"http://www.aQtive.com/dtd/qbit_v1.2Beta.dtd">
<!-- HCI Book Search -->
<!-- Author Alan Dix -->
<!-- Copyright aQtive Ltd 1999 -->
<Qbit>
<!-- Service Bits -->
<Service>
<Name>HCI Book Search</Name>
<Icon>user/images/eye-icon.gif</Icon>
```

```
<Author>Alan Dix</Author>
<Help>Search for references to this topic in the HCI book</Help>
<For Name="com.aqtive.qbits.general.SomeWords" Type="Java Class">
<URL Expand="Yes" Label="search the HCI
book">http://www.hcibook.com/hcibook/search/dosearch.cgi?query=$0</UR
L>
</For>
</Service>
</Qbit>
//-----------------------------------------------------------------

//-----------------------------------------------------------
//-------- Customer Code
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE Qbit PUBLIC
"-//aQtive//Qbit Specification V1.2Beta//EN"
"http://www.aQtive.com/dtd/qbit_v1.2Beta.dtd">
<!-- Regular Expression Qbit Template -->
<!-- Author Alan Dix -->
<!-- Copyright aQtive Ltd 1999 -->
<Qbit>
<!-- Recogniser Bits -->
<Recogniser>
<Name>Customer Code Recogniser</Name>
<Recognises>
<Match Type="RegExp">^\s*(C[A-Za-z][0-9]{4,6})\s*$</Match>
</Recognises>
<!-- N.B. Recogniser "Is" should match Service "For" -->
<Is Type="User" Name="Customer Code Type">
<Field Name="code" Expand="Yes">$1</Field>
</Is>
</Recogniser>
<!-- Service Bits -->
<Service>
<Name>Customer Code</Name>
<Icon>user/images/customer.gif</Icon>
<Author>Alan Dix</Author>
<Help>Look up the customer on ACME's Corporate Intranet</Help>
<For Type="User" Name="Customer Code Type">
```

```
<URL Expand="Yes" Label="look up customer code"
>http://internalserver.acme.com/intranetsearch.cgi?code=$0</URL>
</For>
</Service>
</Qbit>

—
//-----------------------------------------------------------------

//-----------------------------------------------------------------
//-------- Surelynot Date

<?xml version="1.0" standalone="no" ?>

<!DOCTYPE Qbit PUBLIC
"-//aQtive//Qbit Specification V1.2Beta//EN"
"http://www.aQtive.com/dtd/qbit_v1.2Beta.dtd">

<!-- Example Date Qbit -->
<!--Author: Alan Dix -->
<!--Copyright aQtive limited 1999 -->
<Qbit>
<!-- Recogniser Bits -->
<!-- recognises dates such as 12/12/99 3-7-1997 1-1/2000 01/01/2000
abc    -->
<Recogniser>
<Name>Example Date Recogniser</Name>
<Recognises>
<Match Type="RegExp">^\s*([0-3]?[0-9])[-/]([0-3]?[0-9])[-/]([0-
9]{2}|[0-9]{4})\s*$</Match>
</Recognises>
<!-- N.B. Recogniser "Is" should match Service "For" -->
<Is Type="User" Name="Surelynot Date Type">
<Field Name="DD" Expand="Yes">$1</Field>
<Field Name="MM" Expand="Yes">$2</Field>
<Field Name="YY" Expand="Yes">$3</Field>
</Is>
</Recogniser>
<!-- Service Bits -->
<Service>
```

```
<Name>Surelynot Date</Name>
<Icon>user/images/date.gif</Icon>
<Author>Alan Dix</Author>
<Help>Look up a date in surelynot date</Help>
<For Type="User" Name="Surelynot Date Type">
<URL Label="lookup the date - UK format"
Expand="Yes">http://www.surelynot.com/dates/date.cgi?day=${DD}&amp;amp;mo
nth=${MM}&amp;amp;year=${YY}</URL>
<URL Label="lookup the date - US format"
Expand="Yes">http://www.surelynot.com/dates/date.cgi?day=${MM}&amp;amp;mo
nth=${DD}&amp;amp;year=${YY}</URL>
</For>
</Service>
</Qbit>

//-----------------------------------------------------------------
```

CLAIMS

1. A method of providing active assistance to a user performing a computer type task by using a software interface agent with a control unit, said method comprising the steps of:

providing a software interface agent for monitoring the activities of a user, said agent recording data representative of the user's activities,

analysing the data obtained from said user's activities,

identifying events/data present in said user's activities and generating data corresponding to said identified events/data,

comparing said generated data with a database of information services or data accessible by said user's control unit, and

selecting at least one service corresponding to the analysed and identified data and displaying to a user a representation of said selected service, said representation being selectable by the user to access said selected service.

2. A method as claimed in claim 1 wherein said method is performed in real-time when a user is using the control unit.

3. A method as claimed in claim 1 wherein said method may be performed when the control unit is not being used on a stored work product of the user.

4. A method as claimed in any preceding claim wherein the method is usable with a user control unit which is conveniently a computer.

5. A method as claimed in any one of claims 1 to 3 wherein the method is usable with any similar device which incorporated an embedded computer processing unit, such as a microprocessor.

6. A method as claimed in any preceding claim wherein a graphical representation of the service is presented to the user.

7. A method as claimed in any one of claims 1 to 5 wherein an audible representation of the service is provided to the user.

8. A software interface agent for monitoring the activities of a user using a control device and for providing prompts based on a review of data from the user's activities to the user to allow the user to decide to accept said prompts, said software interface agent comprising,

clipboard program means for monitoring the activities of a user, data recognition program means for comparing user input data with a user definable database of recognition program elements, and for providing a corresponding data recogniser output data,

user service program means coupled to at least one user service for receiving said recogniser output data and for activating said service, obtaining the results of said services, and

indication means for providing to the user an indication that a service has been performed, the user being capable of interacting with the indication means to obtain the results of the service performed.

9. A software interface agent as claimed in claim 8 wherein said indication means is a graphical display means.

10. A software interface agent as claimed in claim 8 wherein said indication means is an audible message provider.

11. A software interface agent as claimed in claim 9 or 10 wherein said computer program is operable in a host personal computer, laptop, palmtop or the like.

12. A software interface agent as claimed in any one of claims 9 to 11 wherein the software interface agent is downloadable from the internet to a user's host machine.

13. A software interface agent as claimed in claim 9 or 10 wherein said computer program is operable in a control unit such as a web TV remote control, a telephone, a cellphone which have a processing unit with a memory for

receiving said program.

14. A software interface agent as claimed in any one of claims 8 to 13 wherein said at least one service is a web browser program which is launched when said recogniser output data is received.

15. A software interface agent as claimed in claim 14 wherein the computer program includes further program elements for calculating what has been copied to the clipboard program means from the user's activities.

16. A software interface agent as claimed in any one of claims 8 to 15 wherein the service program means includes data types corresponding to a plurality of different services, said data type being generated by said data recognition program means.

17. A software interface agent as claimed in any one of claims 8 to 16 wherein each recogniser program element has two nodes, a first node being settable and corresponding to the input of the recogniser, a second node being listenable and corresponding to the output of the recogniser.

18. A software interface agent as claimed in claim 17 wherein said service program means has a first settable node, a listenable node for receiving data items received at said clipboard, and a callable node returning the results of a service.

19. A software interface agent as claimed in any one of claims 8 to 18 wherein the graphical display means is a computer screen and a graphic window representing a desktop service is displayable, said window being clickable-on and being interactive with said user.

20. A software interface agent as claimed in claim 19 wherein said window displays a plurality of services icons found by said software interface agent, each said services being actuatable by said user by clicking on a selected icon.

21. A software interface agent as claimed in claim 20 wherein said icons may include search engines for browsing the worldwide-web, user desktop items such as CD

encyclopaedias and dictionaries and graphical display programs.

22. A software interface agent as claimed in any one of claims 8 to 21 wherein the software interface agent operates in real-time while the user is using the computer.

23. A software interface agent as claimed in any one of claims 8 to 21 wherein the software interface agent may not operate in real-time; it may operate on a user's work product after it has been created.

24. A software interface agent as claimed in any one of claims 8 to 23 wherein software interface agent data recognition program means and said user service program means may operate in synchronous or asynchronous versions.

25. A software interface agent as claimed in claim 24 wherein said software interface agent supports both synchronous and asynchronous operation and converts automatically between said versions.

26. A software interface agent as claimed in claim 23 or 24 wherein said asynchronous versions are used in networked environments.

27. A computer program product comprising:

a computer usable medium having computer readable code means in said medium for monitoring the activities of a user operating a control device and for providing to the user prompts or suggestions to allow the user to perform further activities using said control device, said computer program product having:

computer readable program code means for monitoring the activities of a user and for receiving data from the user's activities, computer readable program code means for comparing the data from said user's activities with a user definable database of data recognition elements and for providing outputs from said data recognition elements,

computer readable program code means for receiving said data recognition element outputs and for executing a

service function and obtaining the results of said service,

computer readable program and code means for causing said control unit to indicate to the user when the results of said service are available to the user to access.

28. A computer program product as claimed in claim 27 wherein said computer program product is stored on a computer disk and said control unit is a PC-type, laptop or a palmtop computer.

29. A computer program product as claimed in claim 27 where in said computer program product is stored on any other form of electronically readable medium, such as a programmable read-only-memory integrated circuit (PROM) which may be located in a computer, TV control unit, cellphone or the like.

30. A computer program product as claimed in any one of claims 27 to 29 wherein the computer program product is created at a user's location by downloading the software interface agent from the internet.

31. A computer program product as claimed in any one of claims 27 to 30 wherein the computer program product includes computer readable code means for causing the computer to display a window or icon on a computer screen to indicate the results on said service, said icon being clickable-on by a user to display the results of the service.

32. A control unit having a software interface agent loaded in the form of a computer program product, said computer program product having a computer usable medium having a plurality of executable program code means for monitoring the activities of a control unit user, executing at least one service based on data from the user's activities and for obtaining results of said service, and for providing an indication to the user that results have been obtained for inspection by said user.

33. A control unit as claimed in claim 32 wherein the control unit is a computer.

34. A control unit as claimed in claim 32 or 33 wherein the computer program product is a disk; either a floppy disk or a CD or delivered over the internet.

35. A control unit as claimed in any one of claims 32 to 34 wherein the program code means includes display code means for displaying icons in a window display to the user, said icons being representative of said services performed by said software interface agent.

36. A control unit as claimed in any one of claims 32 to 34 wherein the indication to the user is provided by an audio signal from audio signal generating means.

37. A control unit as claimed in any one of claims 32 to 36 wherein said software interface agent operates in real-time as the user performs activities with the control unit or computer.

38. A control unit as claimed in any one of claims 32 to 37 wherein the software interface agent supports synchronous and asynchronous operation.

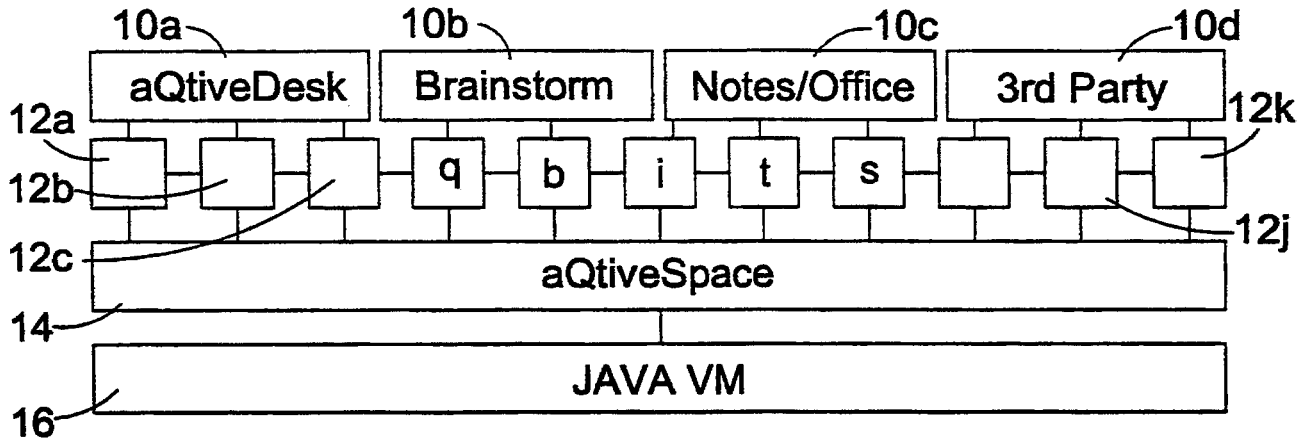

10a
10b
10c
10d
aQtiveDesk
Brainstorm
Notes/Office
3rd Party
12a
12k
12b
q
b
i
t
s
12j
12c
aQtiveSpace
14
JAVA VM
16

Fig.1

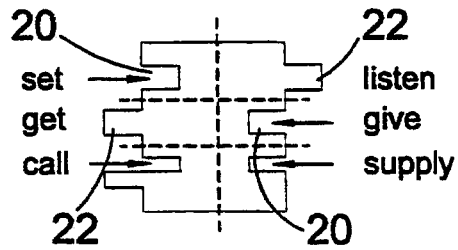

20
22
set
listen
get
give
call
supply
22
20

Fig.2

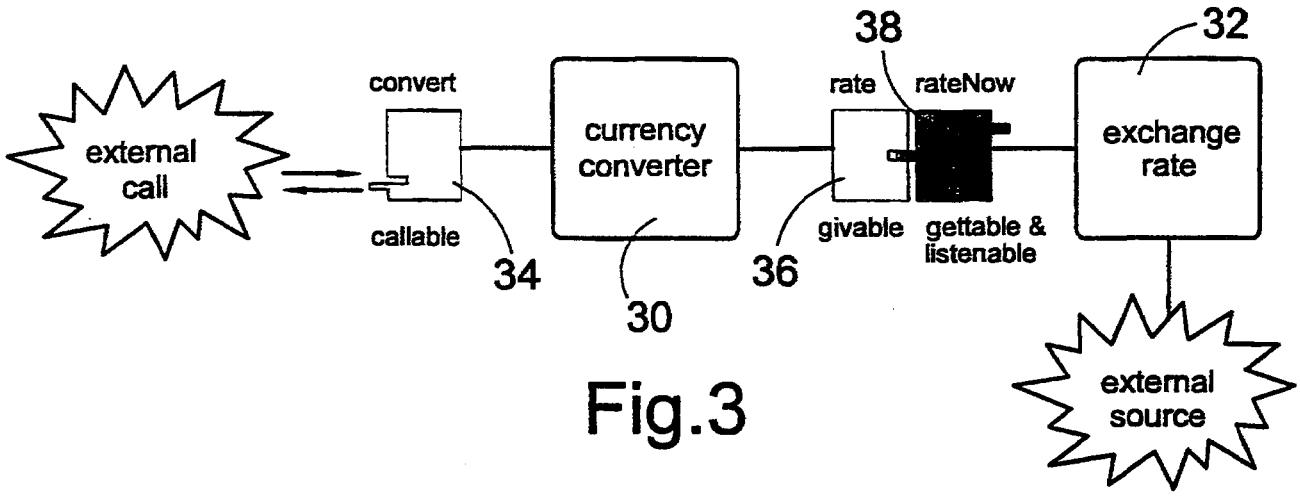

external call
convert
callable
34
currency converter
30
38
rate
rateNow
givable
36
gettable & listenable
32
exchange rate
external source

Fig.3

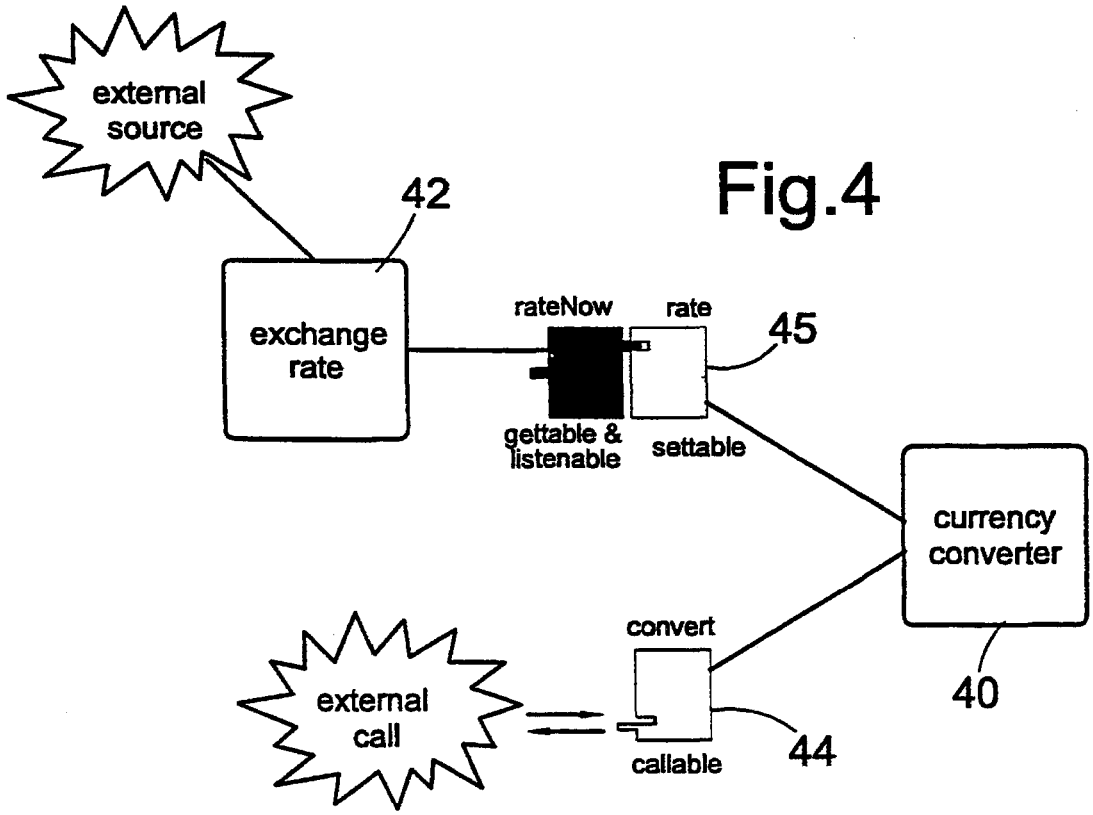

external source
42
exchange rate
rateNow
rate
45
gettable & listenable
settable
currency converter
40
convert
external call
callable
44

Fig.4

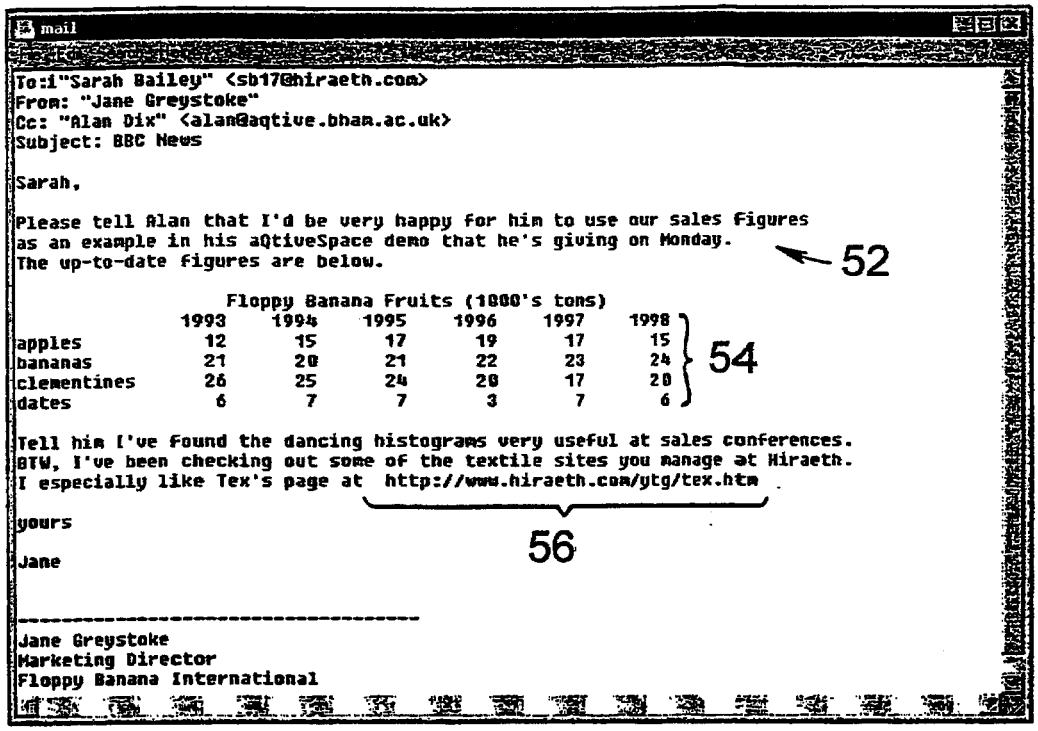
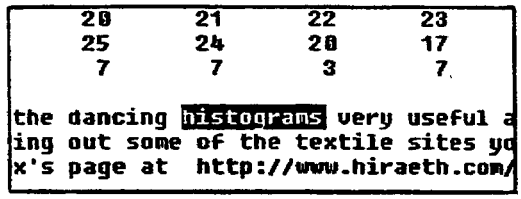


Fig.5



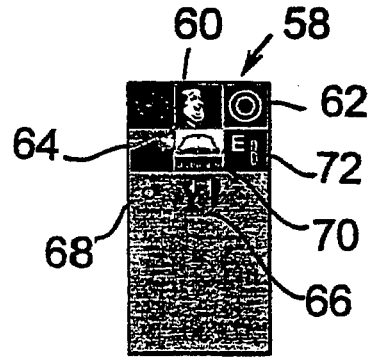
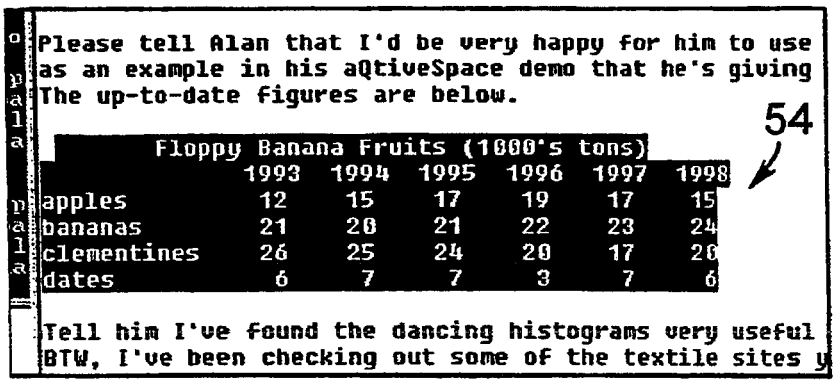


Fig.6



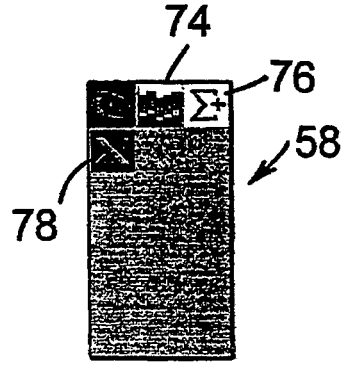


Fig.7

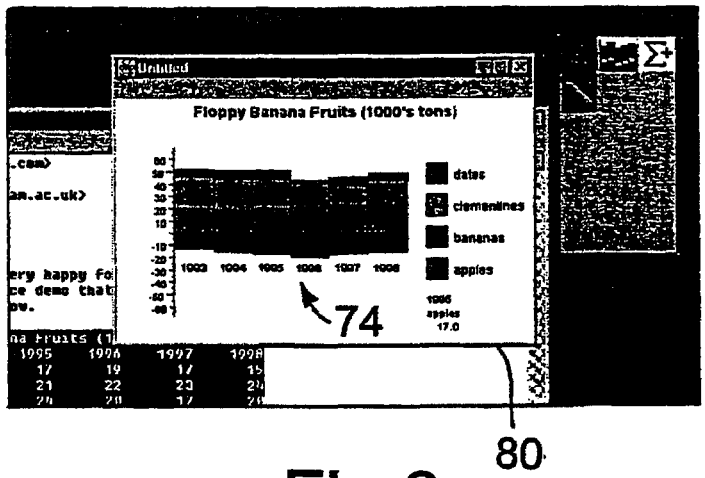

Floppy Banana Fruits (1000's tons)
dates
clementines
bananas
apples
74
80

Fig.8

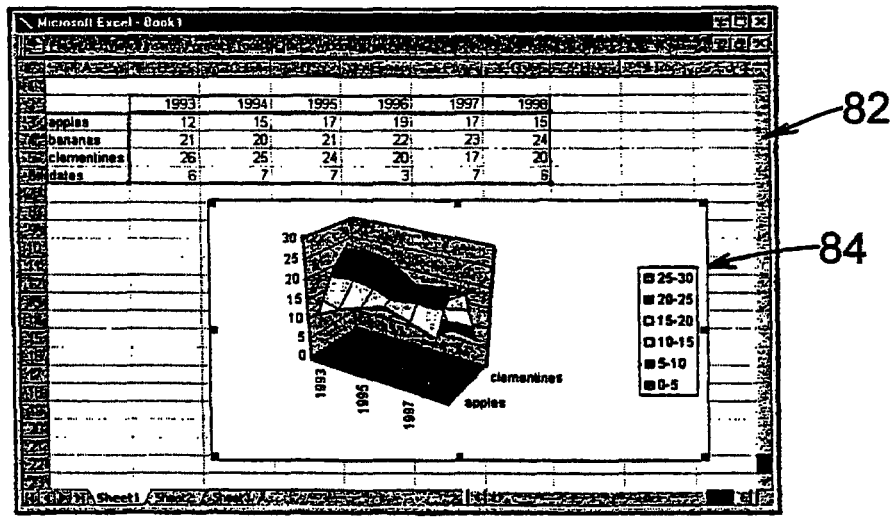

1993 1994 1995 1996 1997 1998
apples 12 15 17 19 17 15
bananas 21 20 21 22 23 24
clementines 26 25 24 20 17 20
dates 6 7 7 3 7 6
25-30
20-25
15-20
10-15
5-10
0-5
82
84

Fig.9

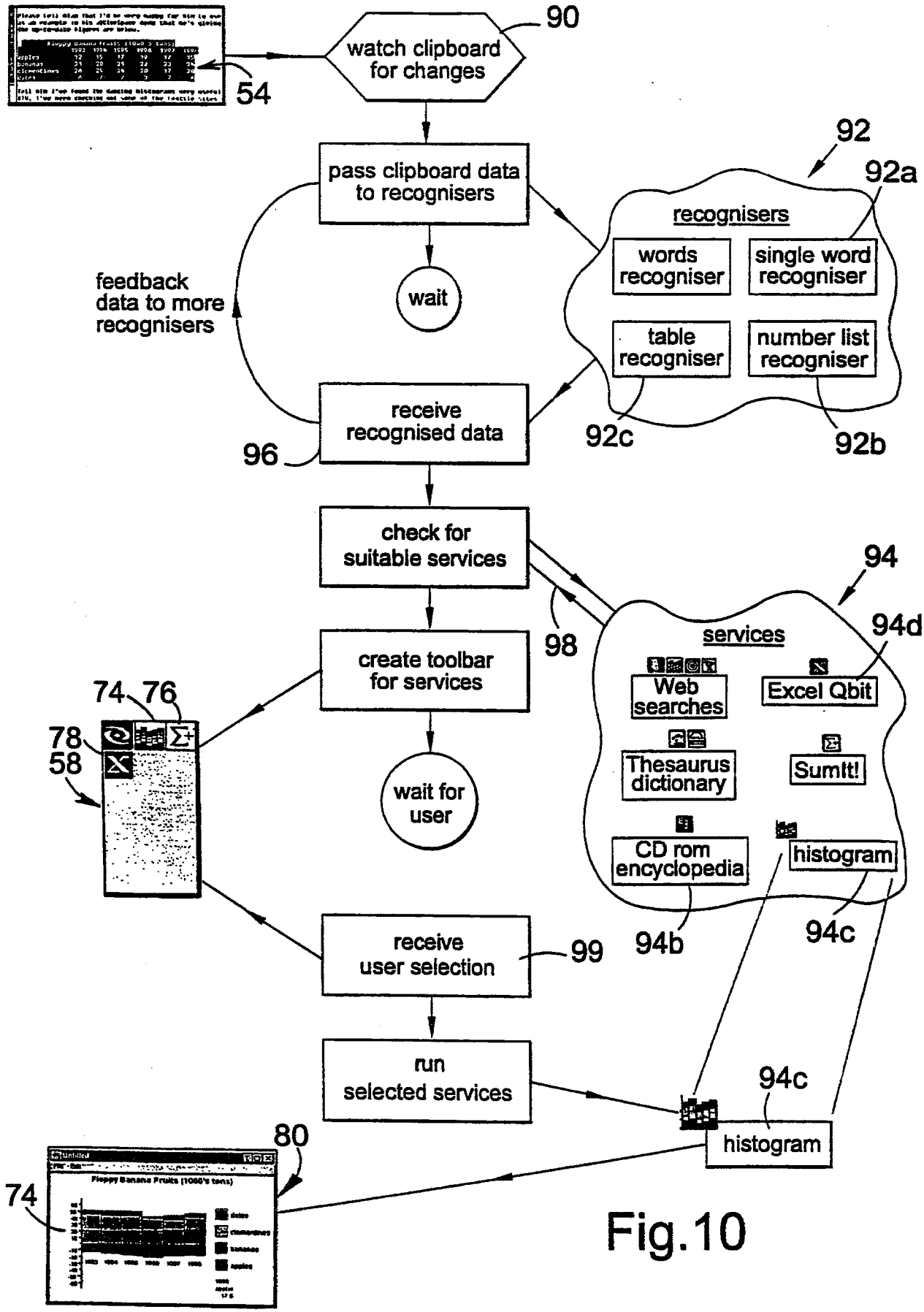

watch clipboard for changes
90
54
pass clipboard data to recognisers
92
92a
recognisers
words recogniser
single word recogniser
table recogniser
number list recogniser
92c
92b
feedback data to more recognisers
wait
receive recognised data
96
check for suitable services
94
94d
services
98
create toolbar for services
Web searches
Excel Qbit
Thesaurus dictionary
SumIt!
CD rom encyclopedia
histogram
94b
94c
74
76
78
58
wait for user
receive user selection
99
run selected services
94c
histogram
80
74

Fig.10

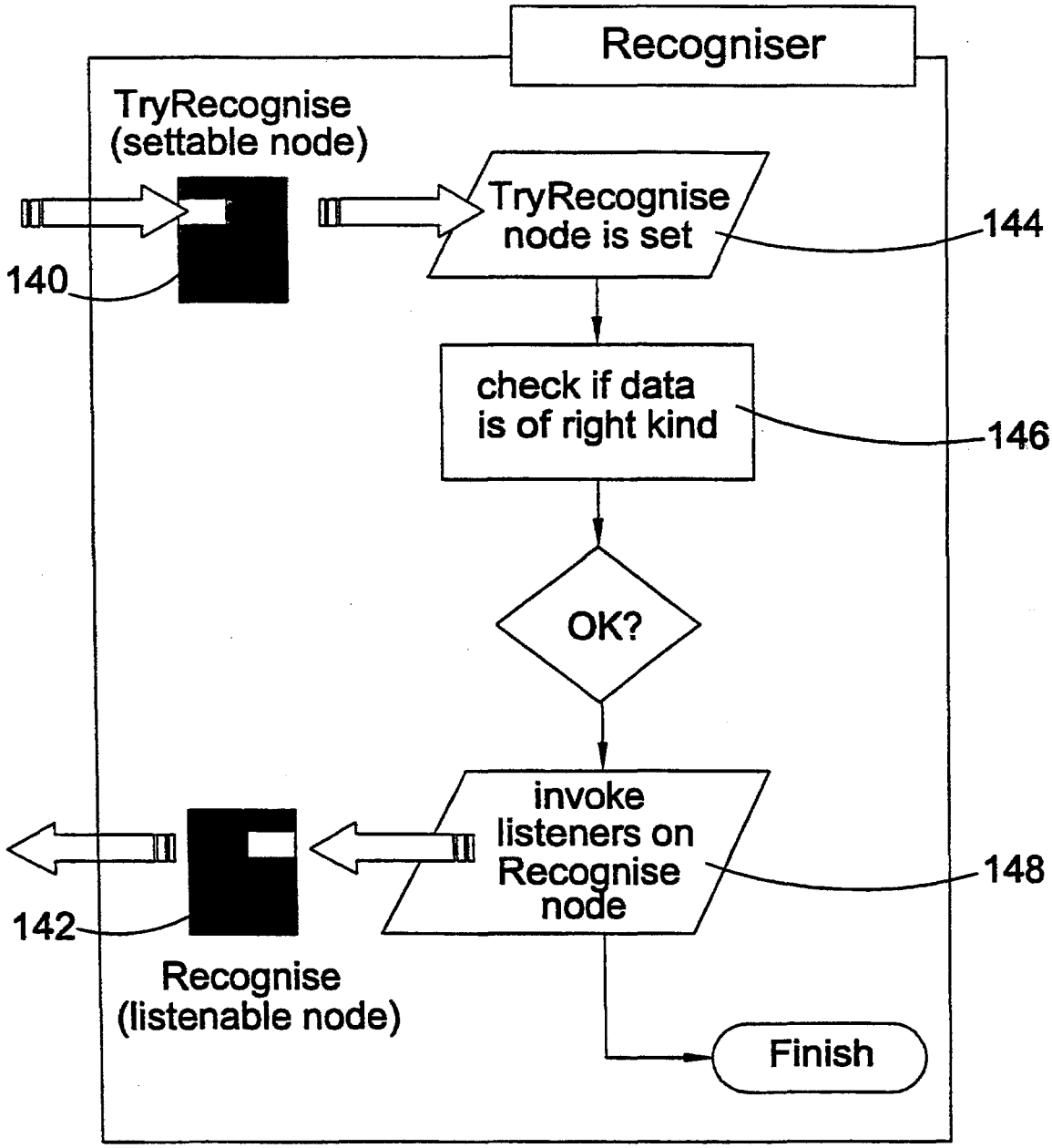

Recogniser
TryRecognise
(settable node)
TryRecognise
node is set
144
140
check if data
is of right kind
146
OK?
invoke
listeners on
Recognise
node
148
142
Recognise
(listenable node)
Finish

Fig.11

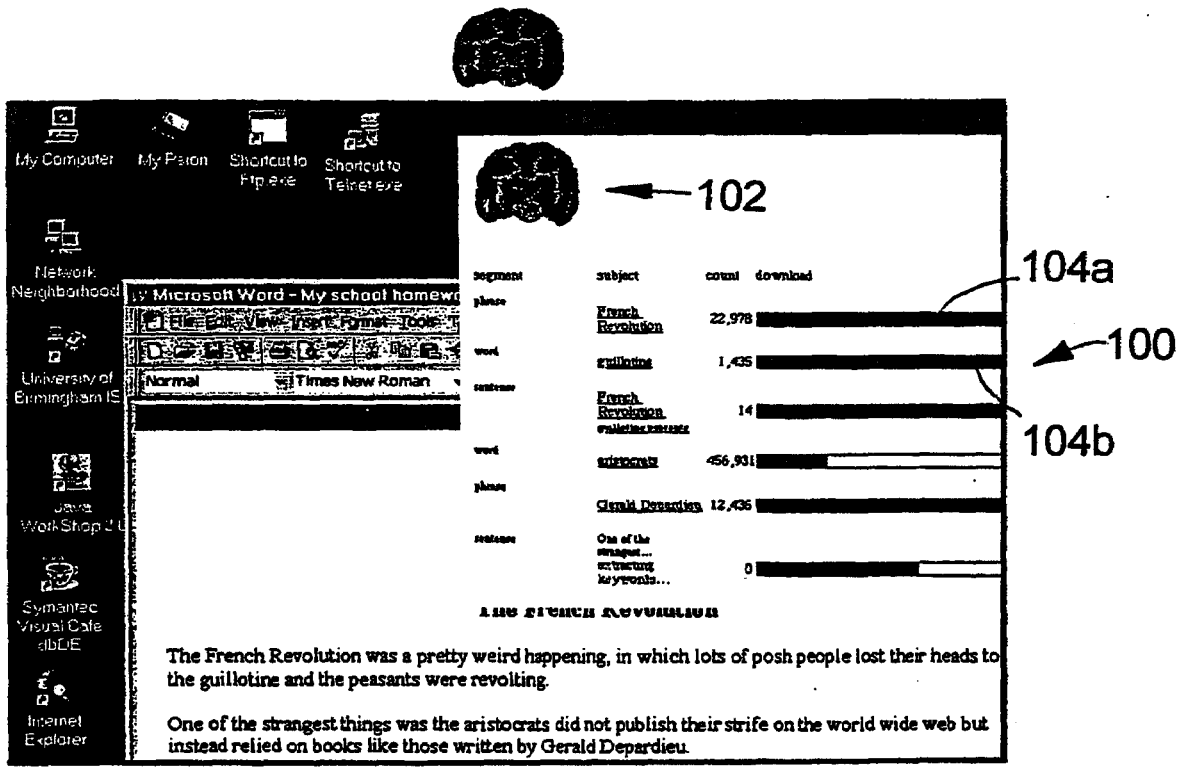

102
104a
100
104b
The French Revolution was a pretty weird happening, in which lots of posh people lost their heads to the guillotine and the peasants were revolting.
One of the strangest things was the aristocrats did not publish their strife on the world wide web but instead relied on books like those written by Gerald Depardieu.

Fig.12

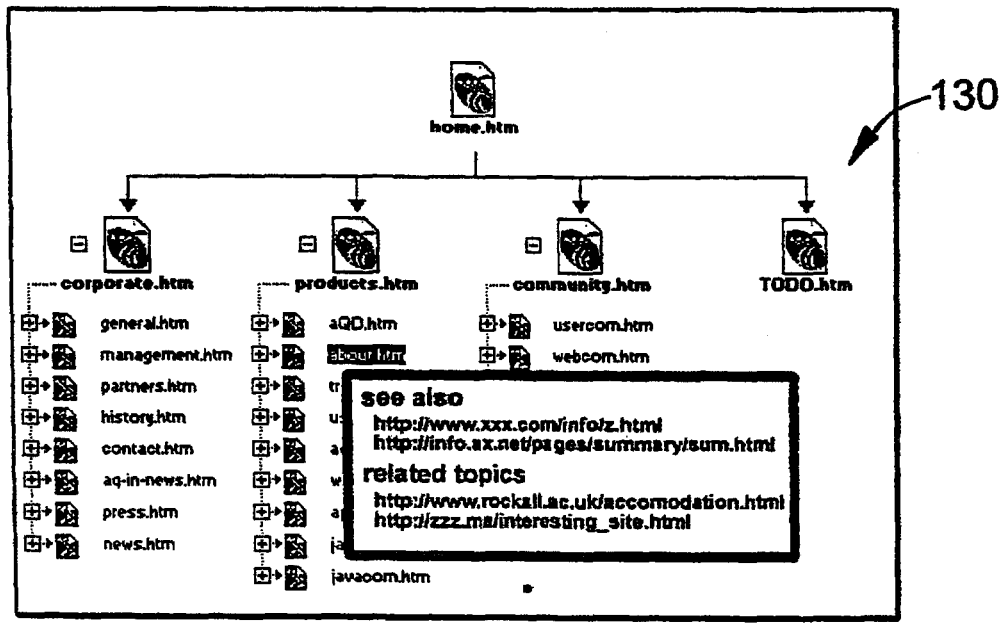

130
home.htm
corporate.htm
products.htm
community.htm
TODO.htm
general.htm
management.htm
partners.htm
history.htm
contact.htm
aq-in-news.htm
press.htm
news.htm
aQD.htm
usercom.htm
webcom.htm
javacom.htm
see also
http://www.xxx.com/info/z.html
http://info.ax.net/pages/summary/sum.html
related topics
http://www.rockall.ac.uk/accomodation.html
http://zzz.ma/interesting_site.html

Fig.14

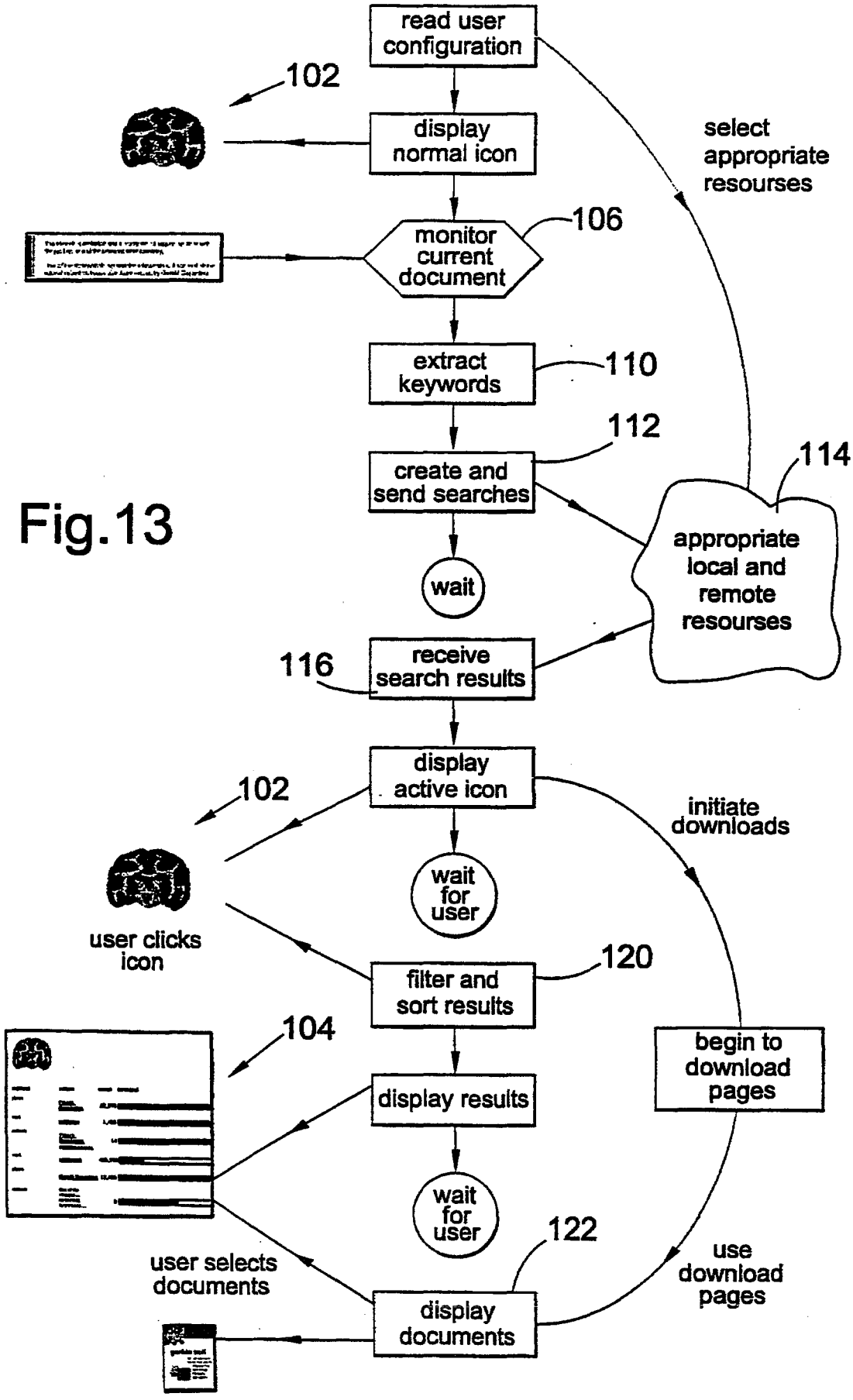

read user configuration
102
display normal icon
select appropriate resourses
monitor current document
106
extract keywords
110
112
create and send searches
114
appropriate local and remote resourses
wait
116
receive search results
display active icon
102
initiate downloads
wait for user
user clicks icon
filter and sort results
120
104
begin to download pages
display results
wait for user
122
user selects documents
use download pages
display documents

Fig.13

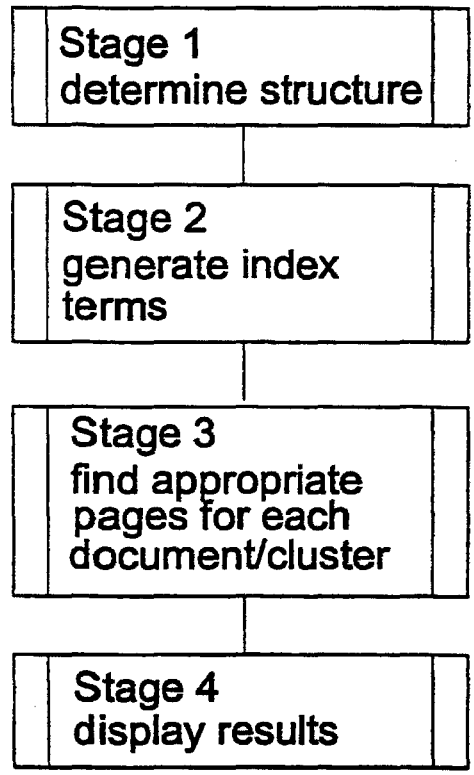


Fig.15a

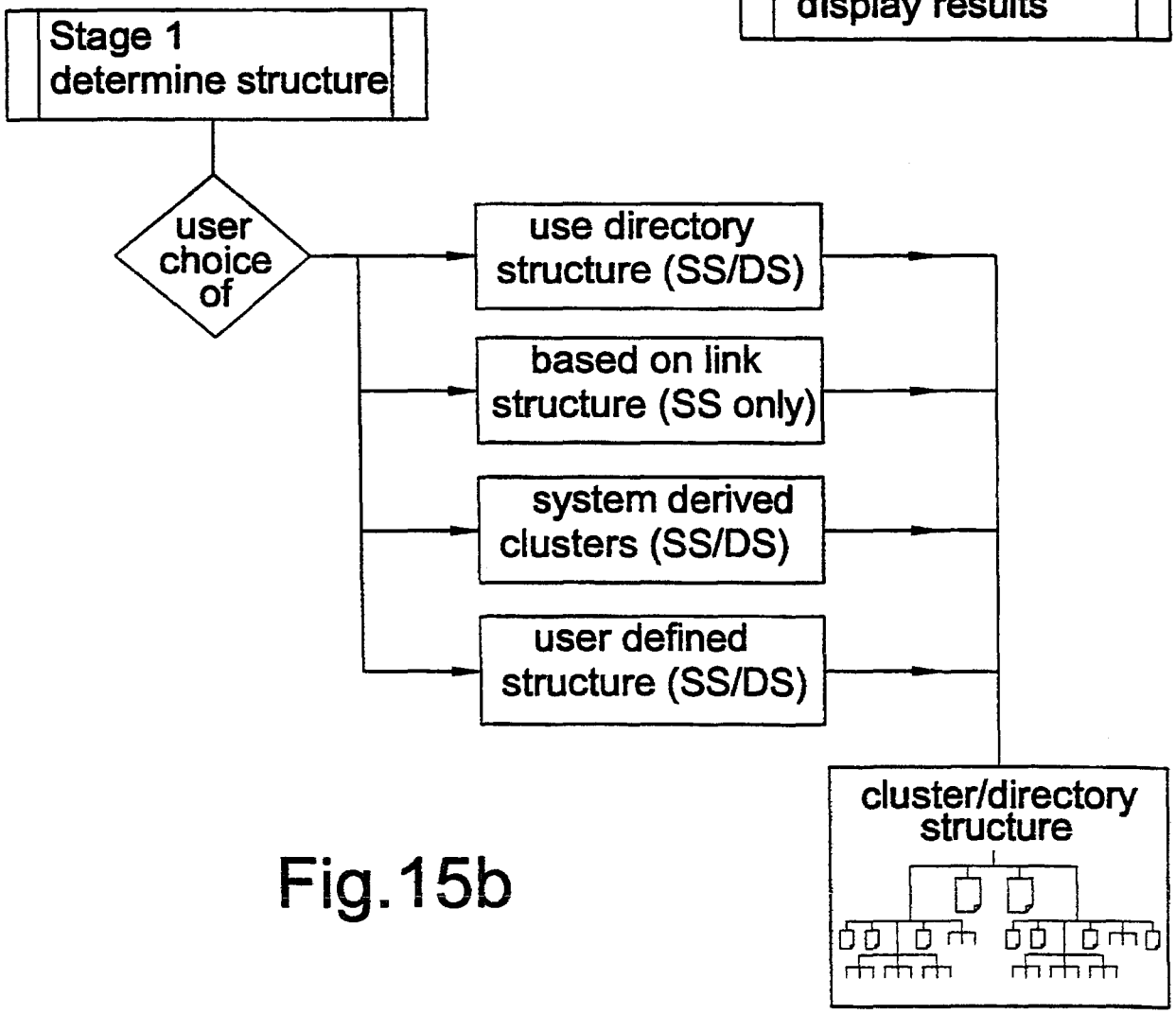


Fig.15b

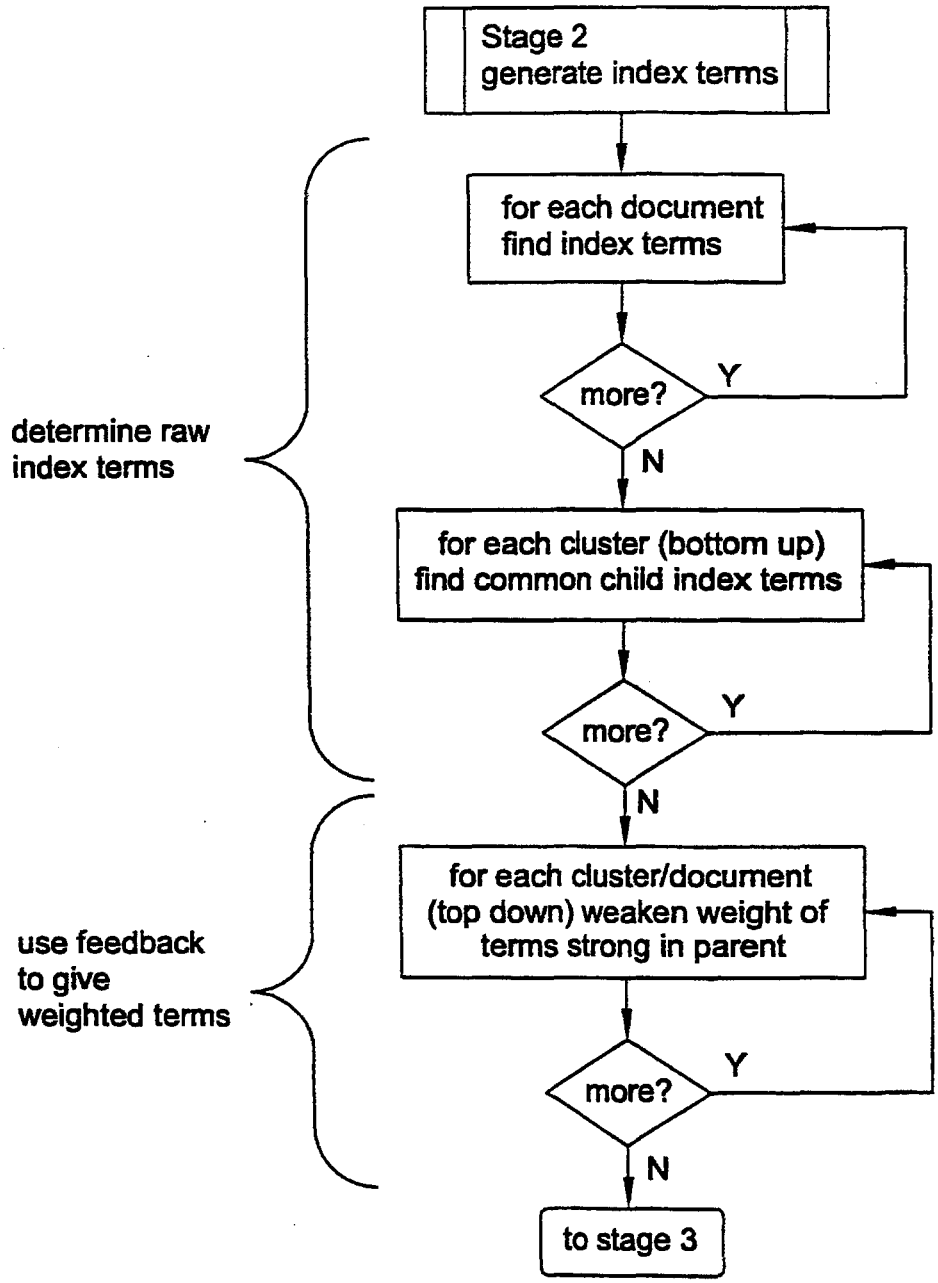

Stage 2
generate index terms
for each document
find index terms
more?
Y
N
determine raw
index terms
for each cluster (bottom up)
find common child index terms
more?
Y
N
use feedback
to give
weighted terms
for each cluster/document
(top down) weaken weight of
terms strong in parent
more?
Y
N
to stage 3

Fig.15c

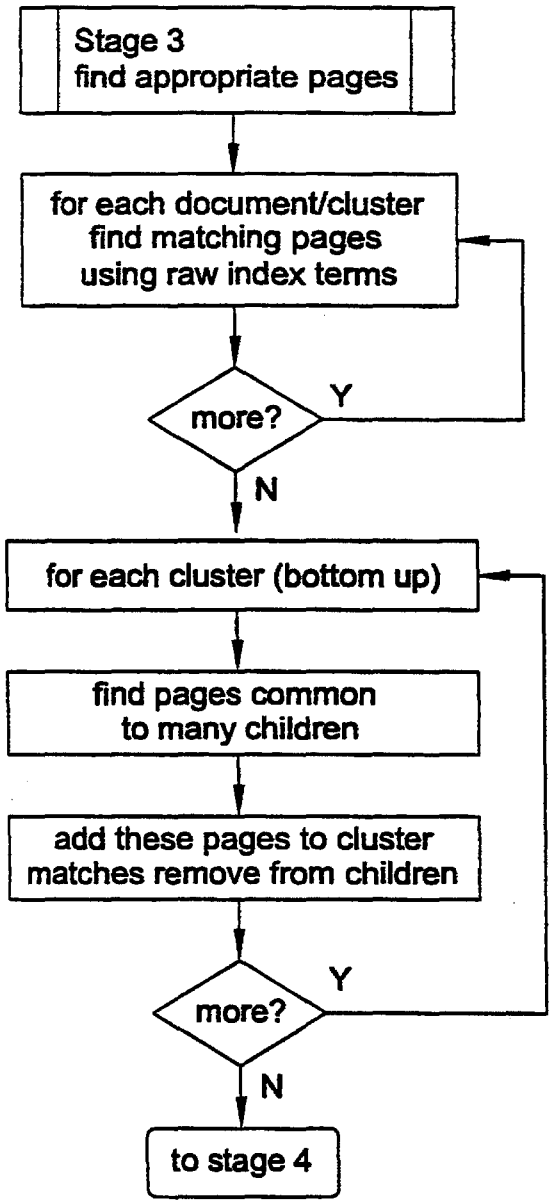


Fig.15d

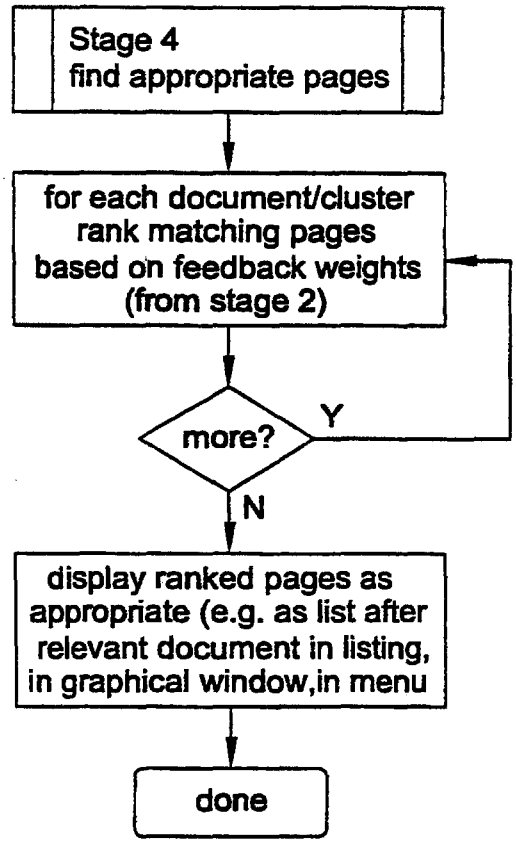


Fig.15e