APPLICATION WINDOW CLOSURE IN RESPONSE TO EVENT IN PARENT WINDOW

Field of the invention

The present invention relates to the field of Internet technologies and the protection of user's privacy when using an application that enables a user to browse confidential or sensitive information. More specifically the present invention relates to a method for effecting the closure of an application window in response to an event associated with its parent window.

Background of the invention

Consumers use the services provided by the World Wide Web to perform confidential transactions such as personal banking, share dealing, buying goods on-line or launching other services from within a portal environment.

A user can access these services over a network communicating with servers located throughout the world for obtaining information. The information is stored on servers and delivered to the user's application from a server by sending files or data packets to the requesting client application from the network server resources.

When performing such transactions on line consumers are concerned about the privacy and security of their information. In particular the use of shared computers by a number of different users at the same location increases these concerns as confidential information is on display in open application windows for other users to view.

When a number of users share the same computer it is possible that a user may not log off or shut down their application correctly and leave sensitive, confidential or personal information on display in open application windows for a subsequent user to view. The information contained in the open application windows could be used by another user to gain unauthorized access to another persons' bank accounts, share-dealing accounts or credit card details. This is a particular problem for network applications accessing confidential information on-line. The security concerns can mostly be mitigated by explicit session management.

Hypertext Transfer Protocol - HTTP, which defines the ways in which network applications interact with network servers; performs session management. HTTP is a 'request-reply' protocol in which the client sends a

request message to the server containing the URL of the required resource. The server looks up the pathname and, if it exists, sends back the file's content in a reply to the client. HTTP allows for content negotiation and authentication.

Authentication involves a user entering a password; on first attempt to access a password protected area, the server reply contains a challenge applicable to that resource, for example a digital signature. When it receives the challenge, the client prompts the user to type a name and password and submits associated credentials with subsequent requests.

However the need to establish and close a connection for every request-reply exchange is expensive, both in overloading the server and in sending too many messages over the network. Persistent connections are used that remain open over a series of request reply exchanges between client and server. A persistent connection can be closed by a client or server at any time by sending an indication to the other participant. Servers may be configured to close a persistent connection when it has been idle for a period of time. If an application over a network has been idle for some time then the connection will time out and the user will be required to re-enter their authentication details into the application.

Session management can therefore be useful to stop an unauthorized user gaining access to a variety of different applications that a user may be logged on to, but session management does not provide management of windows and consequently an unauthorized user will be able to view any confidential information on display in an open application window.

Where an explicit log off has not occurred in the application window there are a number of design implications that can be considered when designing the user interface with regards to the management of windows. Broadly these fall into the following categories which are; not launching any further application windows, only launching a single application window at a given time in an application window that exhibits modal behavior, or closing the application's home page window by using the application's File - Exit route or an operating system provided function such as the 'X' icon or task manager.

One method of providing the management of windows in an application is to use an interpreted client side programming language to provide the required functionality. Using an interpreted client side programming language such as JavaScript, the management of windows is limited to

creating new application windows with defined properties of height, width, color and style. A client side programming language such as JavaScript uses what are known as 'modal windows' to obtain user feedback as to whether to perform specified operations. A typical modal window contains one or more buttons yes, no, cancel or okay and a question prompting the user, for example 'Do you wish to close this application?'. A modal window will not go away until a user clicks on the yes, no, cancel or okay button.

Modal application windows that are provided by client side code have their limitations, for example when using the *showModalDialog()* function in JavaScript. The *showModialDialog()* function creates a separate pop up application window displaying the html page of a given URL. The application window allows the user to address the data of that application window before returning to the parent application window.

The *showModalDialog()* function is Internet Explorer® browser specific and provides no support for the Netscape® browser or the W3C Document Object Model. The *showModalDialog()* function creates a completely standalone application window which has no scriptable relationship with the launching application window. On the Macintosh® operating system platform it is not possible to set the width, height and color properties.

The *onFocus()* function in the parent window enables the selection of the parent application window and captures the event generated by the *onFocus()* function. The *onFocus()*function carries out a check to determine whether the launched application window exists. The *onFocus* event handler executes the specified JavaScript code or function on the occurrence of an *onFocus()* generated event. The *onFocus* event occurs when a window, frame or form element is given focus by the user. The *onFocus()* function works in the Internet Explorer® browser and the Netscape 4.xx® browser, the Macintosh ® and the LINUX® platforms.

A disadvantage of this function is the inability to minimize the modal window in the Microsoft Windows® operating system. The function can be fatal in the Netscape 6® browser. When the close button on the parent application window is selected and the modal application window is open the parent window is closed leaving the modal application window open.

The repeated calling of the *self.focus()*function in a child application window, allows a timer to start when a document is loaded into the application window. The timer rapidly and repeatedly calls the

self.focus() function on the launched window. A disadvantage of implementing the *self.focus()* function is that it is modal to all other applications and the function cannot access other launched application windows, because there does not exist a scriptable relationship between the application window and the modal window. The *self.focus()* function can also close the launched application window via the operating system function calls; thereby leaving the launched window open, which potentially could contain a transactional application in an invalid session.

The *onFocus()* or *onClick()* function in the parent window calls the *onOpenedWindow()function*. Focusing or clicking on the parent application window calls a check for the existence of a launched application window. If the check returns true then the launched application window is brought forward by *windowname.focus()* function. The *windowname.focus()* function has the disadvantage that it does not produce consistent results in the Internet Explorer 4® browser.

Compatibility problems are a major consideration when creating applications using an interpreted client side language. Different applications support different levels of HTML and the amount of JavaScript functionality that an application can support depends on the interpreter in the application.

Another method of closing opened application windows is by a system provided function such as the 'X' icon on the application task bar, the task manager or by the use of the File/Exit on the menu bar. Although this closes the opened application window it provides no automation of the close action of any network or non-network application window launched from within the initial application window and relies on the user to perform the action.

Disclosure of the invention

In accordance with the present invention there is now provided a method for effecting the closure of an application window in response to an event associated with its parent application window comprising the step of, polling on an indicator associated with the parent of the application window and in the absence of said indicator accessing a system provided function causing the closure of the application window. The present invention provides for the automatic closure of a child application window in response to an event associated with the parent of the application

window. The present invention closes the child application window when the parent application window no longer exists. Each launched application window takes on the role of a parent and a child with the control of the application window associated with the individual launched application windows. By associating window control with the individual launched applications windows, the launched application is able to detect whether the application it was launched from exists or not.

Preferably the method provides for the polling of an indicator of the parent of the application window for the existence of the indicator to determine whether the application window is not to be closed. The present invention provides for the data within the application window to be refreshed or reloaded without closing the application window or subsequent launched application windows that are still required by the user.

Preferably the method provides for an event, which is navigating to a resource outside of the application's environment. If a resource is navigated to outside of the application's environment, the application window will close.

Preferably the method provides for monitoring the status of the indicator of the parent of the application window. Preferably the method provides for an application window to operate in a network environment.

According to a second aspect of the present invention, a computer program product comprising computer program code stored on a computer readable storage medium, when executed on a computer, the program code embodied therein effects the closure of an application window in response to an event occurring in the parent of the application window.

According to a third aspect of the present invention, there is provided a system for effecting the closure of an application windows in response to an event associated with its parent application window, the system comprising means for polling on an indicator associated with the parent of the application window and in the absence of said indicator accessing a system provided function thereby causing the closure of the application window.

Preferably the system provides means for polling on said indicator of the parent of the application window for the existence of said indicator to determine whether the application window is not to be closed. Preferably the system provides means for detecting the occurrence of an

event, wherein the event is navigating to a resource outside of the application's environment.

Preferably the system provides a means for, monitoring the status of the indicator of the parent of the application window. Preferably the system provides means for the parent of the application window to be refreshed without destroying data within the application window.

Preferably the system provides means for the parent of the application window to be closed, without closing a required application window.

The present invention when operating in a network environment allows for the inherent problems of network page delivery latency. When an application window is reloading or refreshing data within its window, the invention comprises a means for polling on the existence of an indicator to determine the indicators existence. If it does not exist, the system pauses for m seconds before checking for the existence of the indicator. This will allow for the data to be refreshed within the application window without closing any dependent application windows. This achieves a further advantage of not destroying the data structure of the application's window that is being refreshed. Unlike a traditional desktop system the flags exist throughout the user's session and the data is not lost. The present invention therefore does not need to keep a list of what application windows are open and what event handlers are in use at any given time.

The present invention further provides an advantage in that the described method, computer program product and system operates across a variety of different applications and is not application specific.

Brief description of the drawings

The invention will now be described by way of example only, with reference to the accompanying drawings, in which:

Fig 1 illustrates a portal environment as accessed in a network or non-network environment as found in the prior art;

Fig 2 illustrates multiple applications in a portal environment as found in the prior art;

Fig 3 shows a flow diagram illustrating the application window closing method of the present invention;

Fig 4 illustrates the relationship between multiple application windows as described by the present invention.

Detailed description of the invention

Referring to fig. 1, a portal environment 1, accessed by a network application 2 over a network 3, contains a plurality of hypertext links to other services 4 for example banking 5, insurance 6, investment 7, loans 8, pensions 10 and savings 11.

Clicking on the service's hypertext link can access a service 4. The application sends a request message 13 through the network 3 to the network server 15 containing the URL 12 of the required resource. The network server 15 looks up the pathname of the requested resource 12 and the network server 15 sends back the file contents in a reply message 14 through the network 3 to the application 2. The requested resource 12 is loaded into an application window in the portal environment 1.

A plurality of services can be accessed and loaded into a plurality of launched application windows whilst retaining access to the portal home page in the initial application window.

Referring to fig. 2, Internet banking is accessed by selecting the 'sign on' hypertext link 20 from the Internet banking section of the portal home page 21. A new application window 22 is launched and displayed together with the portal homepage 21. Confidential information is required to be input into the application's form fields 25 to identify the user to the application. Once a log-on has successfully completed a second application window 23 is launched, displaying a list of options pertaining to the manage accounts window, as a selection of hypertext links 26. To manage a current account, the current account 26 hypertext link is selected and a third application window 24 is launched. Further options are displayed in the newly launched application window. In the example of fig. 2, by way of an example only, a total of four nested application windows are open on the computer screen. Each application window is dependent on the application window that it was launched from.

The present invention is equally applicable to both network environments and non-network environments, where the required information is contained on a server located on a network or stored on a user's computer disk storage system and accessed off-line.

Referring now to fig. 3, the portal home page application is launched 31, displaying the hypertext links to other services within or outside of the portal environment. A variable called '*KeepOpen*' is instantiated in the launched portal homepage window 32. A dependent application window is launched 33 by selecting a hypertext link from the portal homepage 1.

The <body> tag in the html page in a launched application window contains an *onLoad()* function. The *onLoad()* function calls an initial function. The initial function carries out a check to ascertain if the data currently loaded into the application's parent window contains a variable named for example '*KeepOpen*' 34.

If the initial function returns a value of '*false*' 35, the initial function ceases checking whether the variable exists for m seconds 36. This allows for the parent application to be refreshed by a user or another application.

After m seconds the initial function resumes its check on the '*KeepOpen*' variable 37. If the initial function finds that the '*KeepOpen*' variable exists, the initial function returns a value of true 38. The initial function stops checking the existence of the variable for n seconds 39. After n seconds the initial function resumes its check on the 'KeepOpen' variable.

The initial function continues through this process until the initial function returns a value of 'false' 35. The initial function pauses checking on the 'KeepOpen' variable for m seconds 36 to allow for the parent application being refreshed. After m seconds the initial function resumes its check on the 'KeepOpen' variable. If the 'KeepOpen' variable does not exist then a value of 'false' is returned 41 and the child window is closed 42.

The present invention allows for the inherent problems of network page delivery latency when an application is reloading or refreshing data within its window. The present invention provides a method for polling on the existence of an indicator to determine the indicators existence. If it does not exist, the method pauses for m seconds before checking the existence of the variable. This process allows for the data within the application window to be refreshed without closing any dependent application windows. This achieves a further advantage of not destroying the data structure of the application's window that is being refreshed,

unlike a traditional desktop system the flags exist throughout the users session. The present invention therefore does not need to keep a list of what application windows are open and what event handlers are in use at any given time.

Referring now to fig. 4 of the present invention, a plurality of application windows can be launched from within a portal environment. Each application window is dependent on the application that it was launched from.

When closing the finance application window 51 by navigating to a resource outside of the portal environment or clicking on a close button, the personal bank account application window 52 and the current account application window 53 will be closed. The current account application window 53 is dependent on the personal bank account application window 52. The personal bank account application window 52 is dependent on the finance application window 51. Each window is dependent on the window that it was launched from as the logic runs in the dependent window. Each launched application window takes on the role of a parent and child, but only if a further application window is launched from within the parent application window therefore creating a child application window.

When closing the current account application window 43, only the current account application window 53 will close, as there does not exist in this example a dependent window.

When closing the portal home page window 50, the finance application window 51, the personal bank accounts application window 52 and the current account application window 53 will close.

CLAIMS

1. A method for effecting the closure of an application window in response to an event associated with its parent application window, the method comprising the step of:

polling an indicator associated with the parent of the application window and in the absence of said indicator accessing a system provided function causing the closure of the application window.

2. A method as claimed in claim 1, further comprising the step of:

polling on said indicator associated with the parent of the application window for the existence of said indicator to determine whether the application window is not to be closed.

3. A method as claimed in claim 2, wherein polling of said indicator further comprises the step of;

monitoring the status of said indicator associated with the parent of the application window.

4. A method as claimed in claim 3, wherein said monitoring of the status of said indicator of the parent of the application window provides for the parent of the application window to be refreshed without destroying the data within the application window.

5. A method as claimed in claim 1, wherein said event is navigating to a resource outside of the application's environment.

6. A method as claimed in claim 1, wherein said application window operates in a network environment.

7. A method as claimed in claim 6, wherein one or more application windows are launched in a network environment.

8. A computer program product comprising computer program code stored on a computer readable storage medium, which when executed on a computer, the program code embodied therein effects the closure of an application window in response to an event associated with its parent application window as claimed in claim 1.

9. A system for effecting the closure of an application windows in response to an event associated with its parent application window, the system comprising :

means for polling on an indicator associated with the parent of the application window and in the absence of said indicator accessing a system provided function causing the closure of the application window.

10. A system as claimed in claim 9, wherein the system further comprises;

means for polling on said indicator associated with the parent of the application window for the existence of said indicator to determine whether the application window is not to be closed.

11. A system as claimed in claim 9, wherein said event is navigating to a resource outside of the application's environment.

12. A system as claimed in claim 11, wherein polling of said indicator further comprises:

means for monitoring the status of said indicator of the parent of the application window.

13. A system as claimed in claim 12, wherein said means for monitoring the status of said indicator associated with the parent of the application window allows for the parent of the application window to be refreshed without destroying data within the application window.

14. A system as claimed in claim 12, wherein said means for monitoring the status of said indicator of the parent of the application window to be closed, provides a means for the parent application window to be closed without closing a required application window.

15. A system as claimed in claim 9, wherein said application window operates in a network environment.

16. A system as claimed in claim 15, wherein one or more application windows are launched in a network environment.