US 20080167124A1

(19) United States

(12) Patent Application Publication

Korchemniy et al.

(10) Pub. No.: US 2008/0167124 A1

(43) Pub. Date: Jul. 10, 2008

(54) SYSTEM AND METHOD FOR ADDING IN-GAME FUNCTIONALITY

(76) Inventors: **Alex P. Korchemniy**, Sandpoint, ID (US); **Charles F. Manning**, Sandpoint, ID (US); **Jason Sanchez**, Sandpoint, ID (US)

Correspondence Address:
CALFEE HALTER & GRISWOLD, LLP
800 SUPERIOR AVENUE, SUITE 1400
CLEVELAND, OH 44114

(21) Appl. No.: **11/969,531**

(22) Filed: **Jan. 4, 2008**

Related U.S. Application Data

(60) Provisional application No. 60/878,822, filed on Jan. 5, 2007.

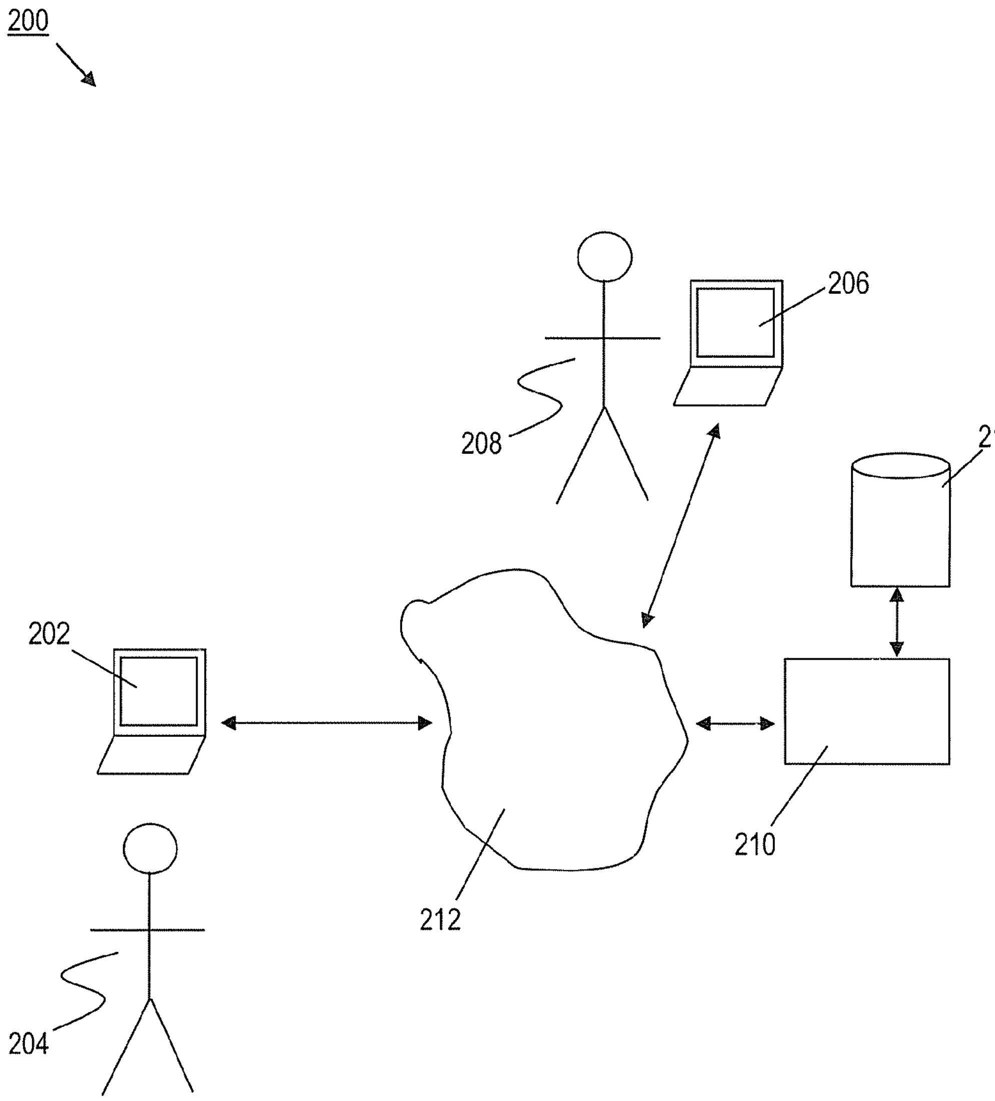
Publication Classification

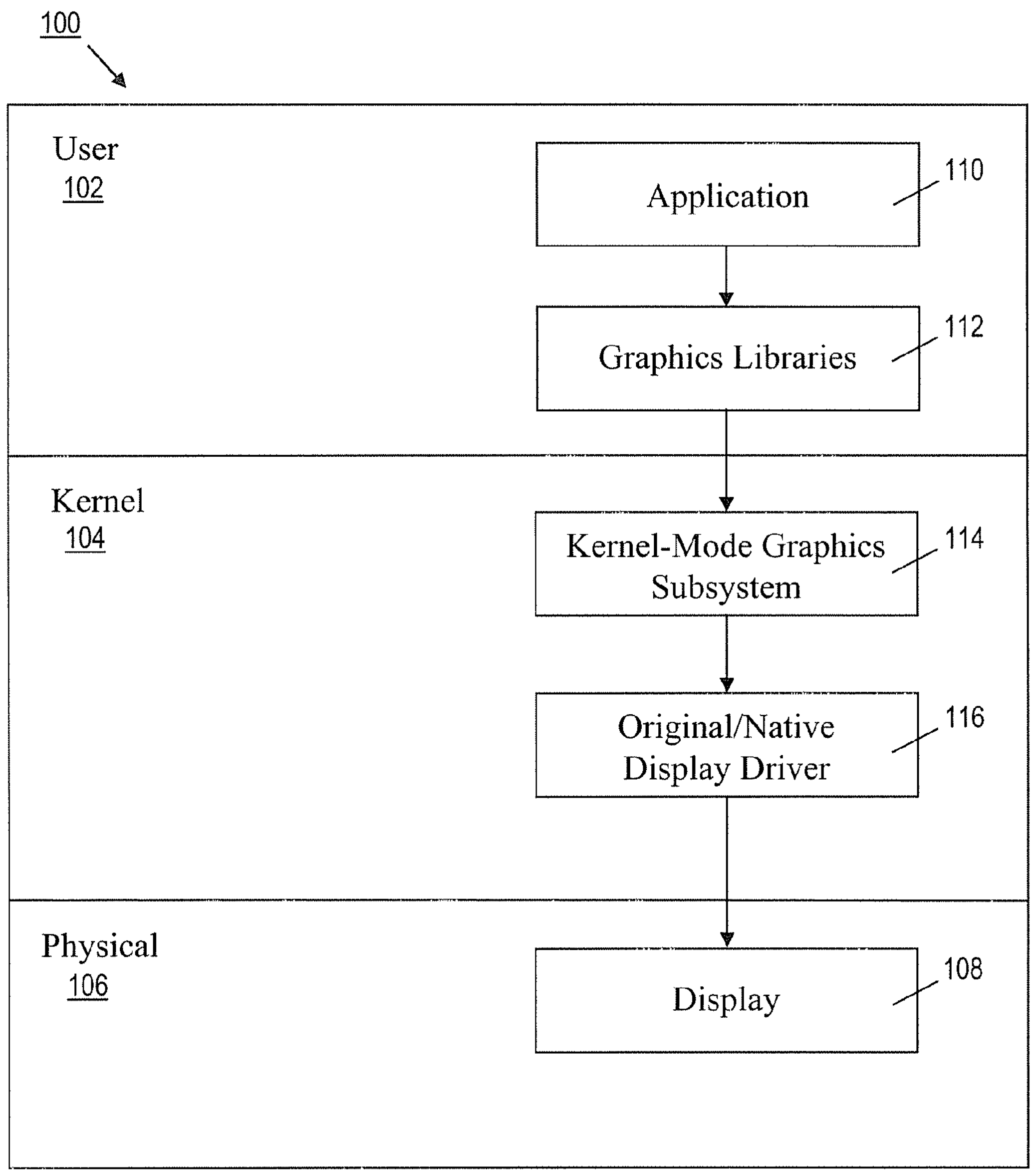
(51) **Int. Cl.**
A63F 9/24 (2006.01)

(52) **U.S. Cl.** **463/31**

(57) **ABSTRACT**

A system and method for adding, accessing and/or using software modules within the context of an active computer game are provided. The system and method include overlaying graphics/video on top of the game.





PRIOR ART

FIG. 1

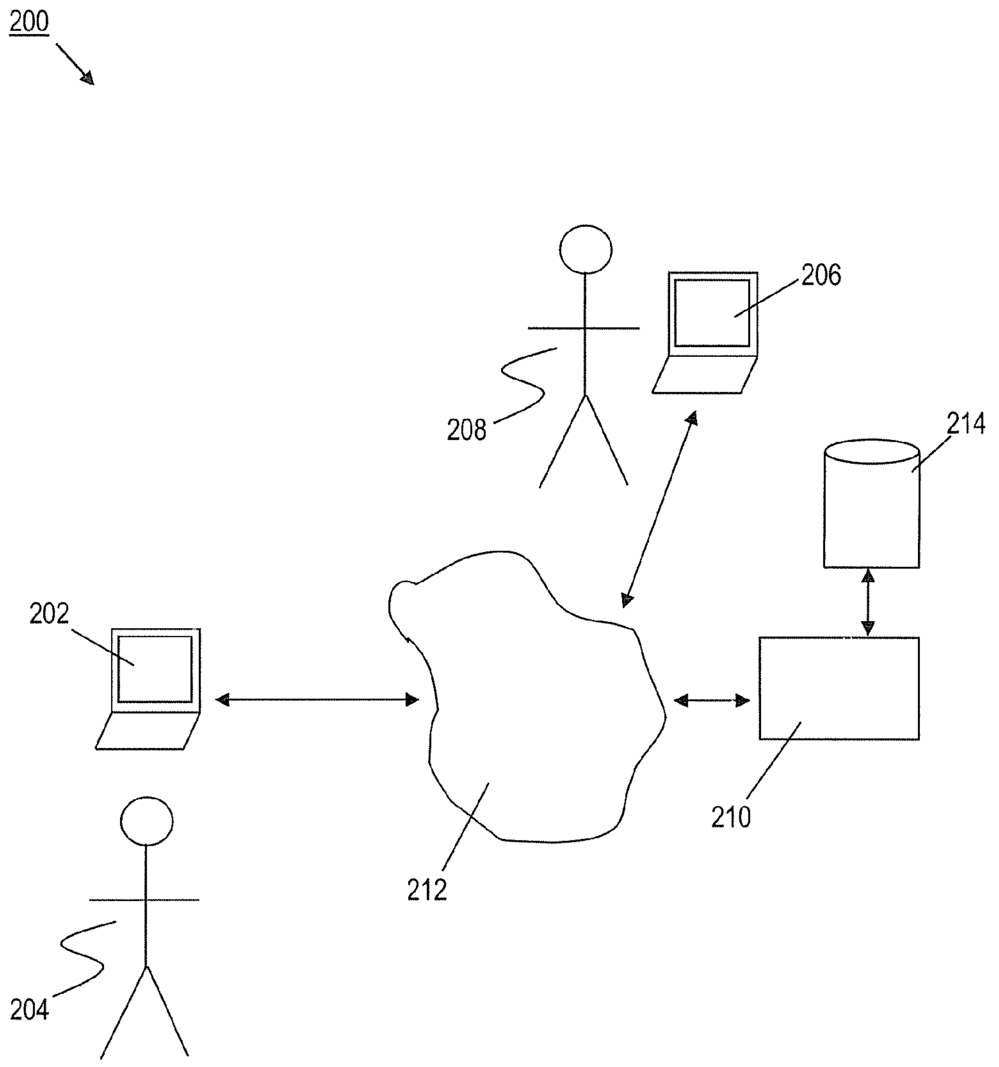


FIG. 2

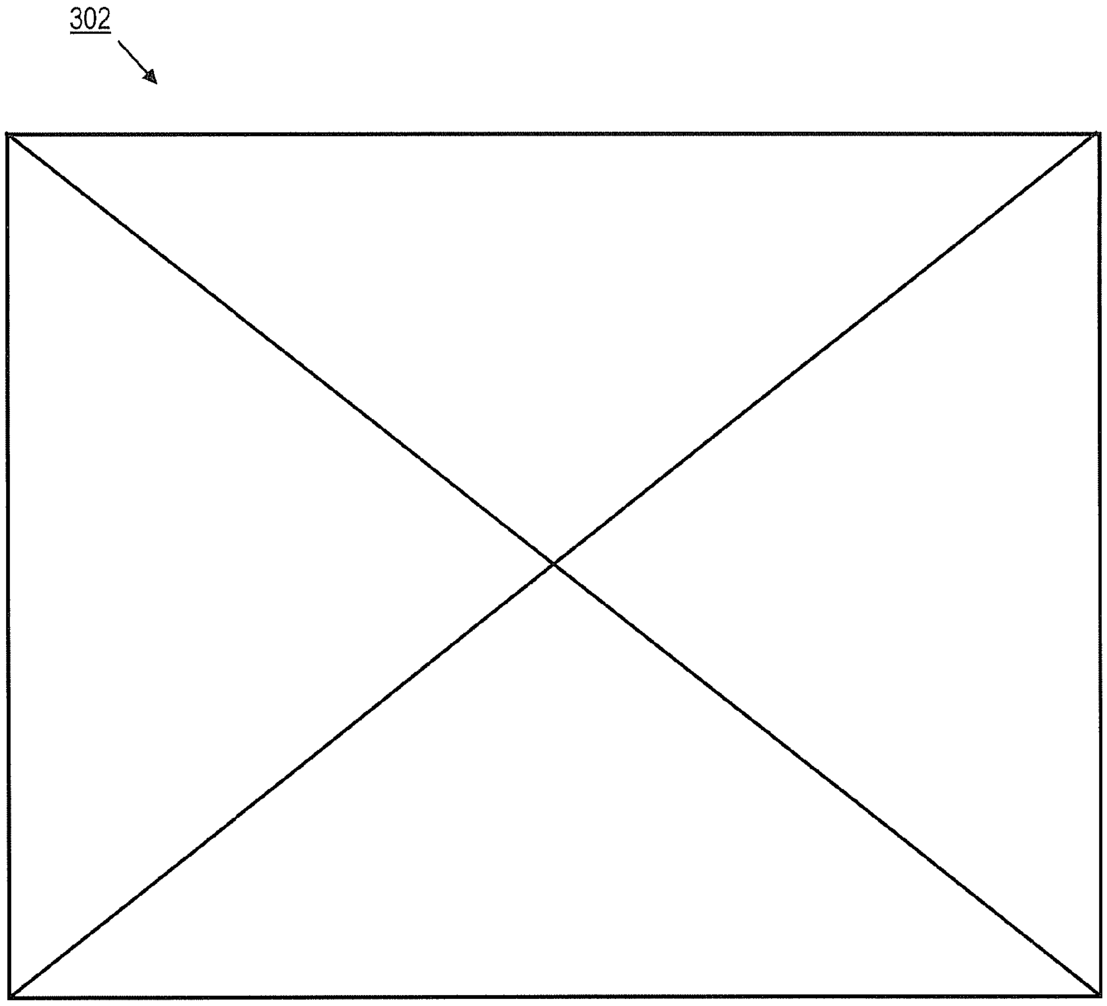


FIG. 3

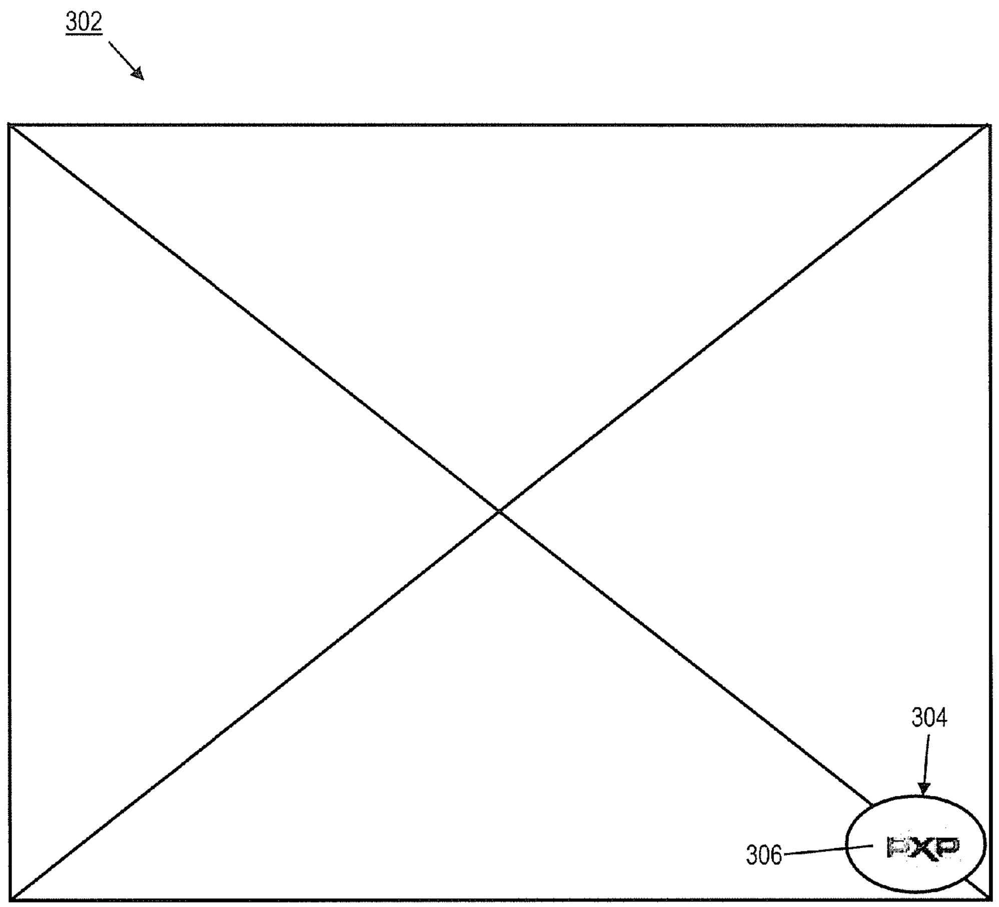


FIG. 4

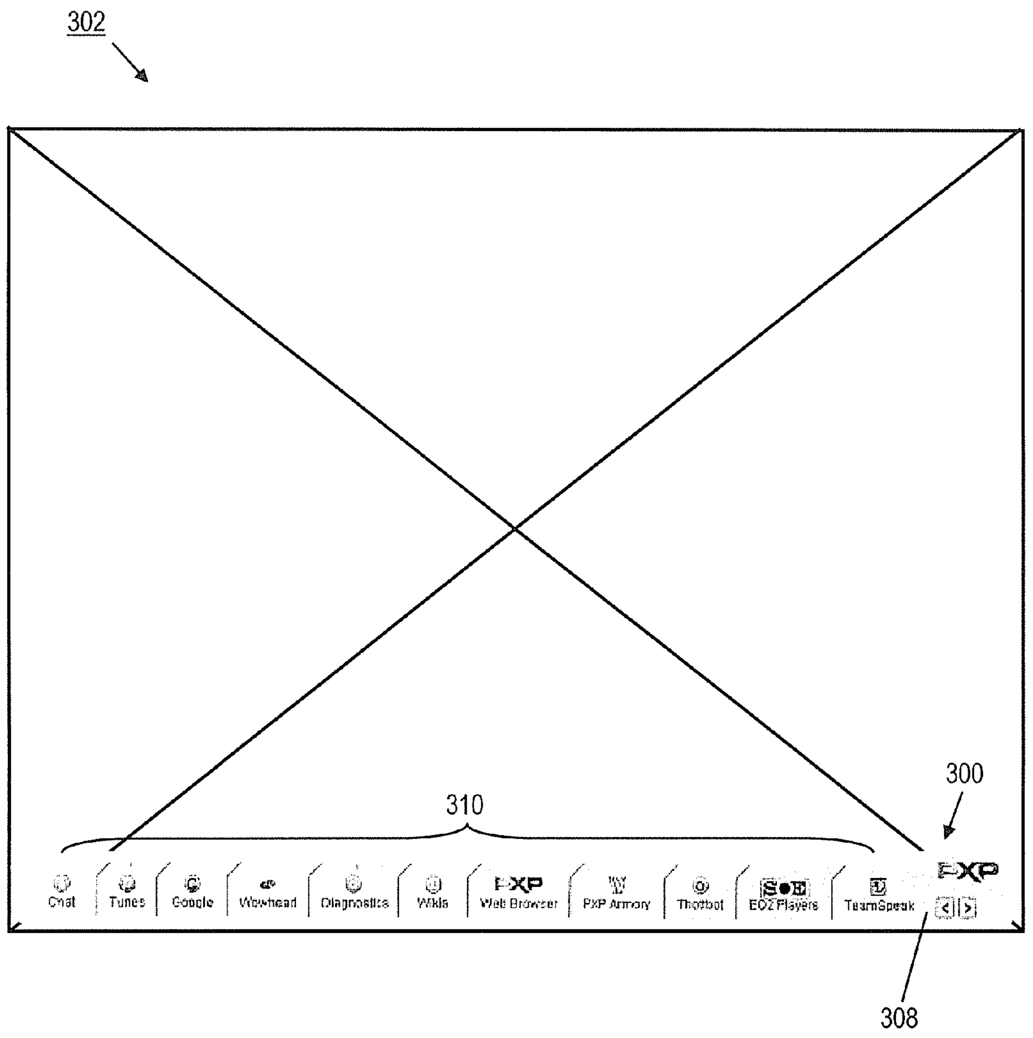

302
310
300
308
Chat
Tunes
Google
Wowhead
Diagnostics
Wikis
PXP Web Browser
PXP Armory
Thottbot
EQ2 Players
TeamSpeak
PXP

FIG. 5

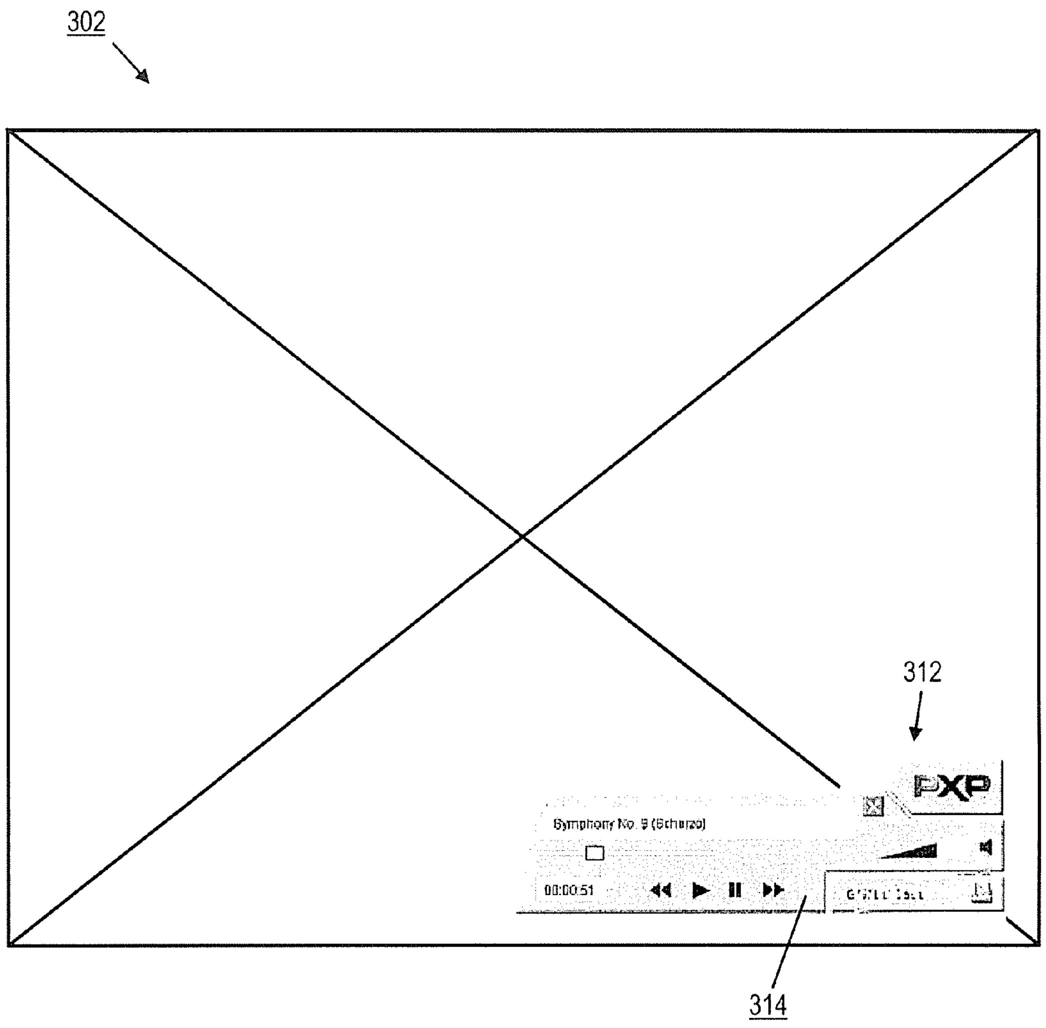


FIG. 6

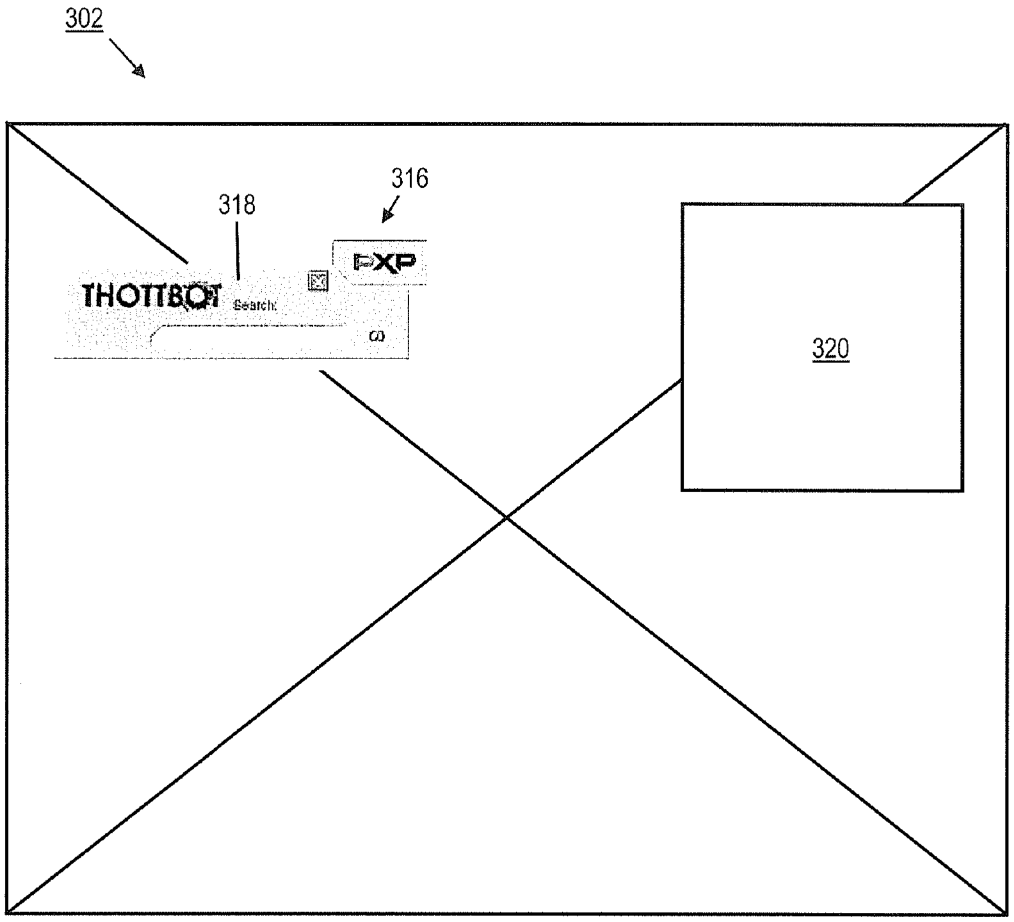


FIG. 7

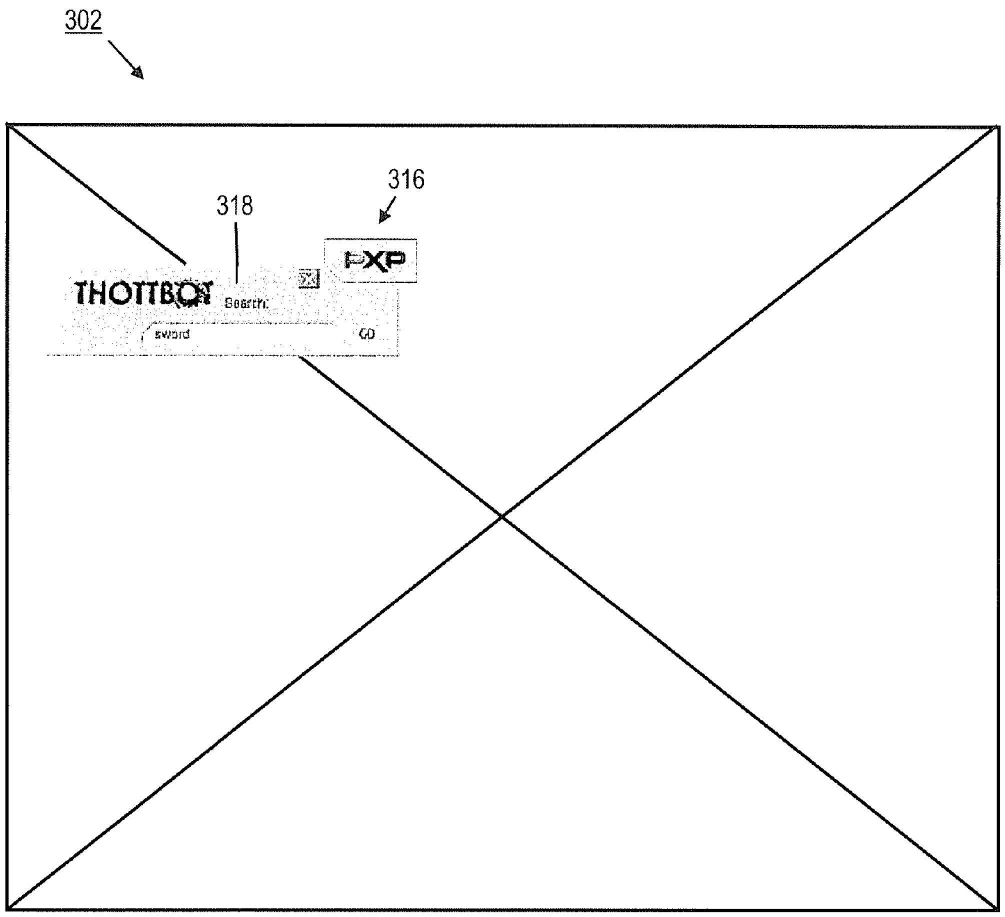
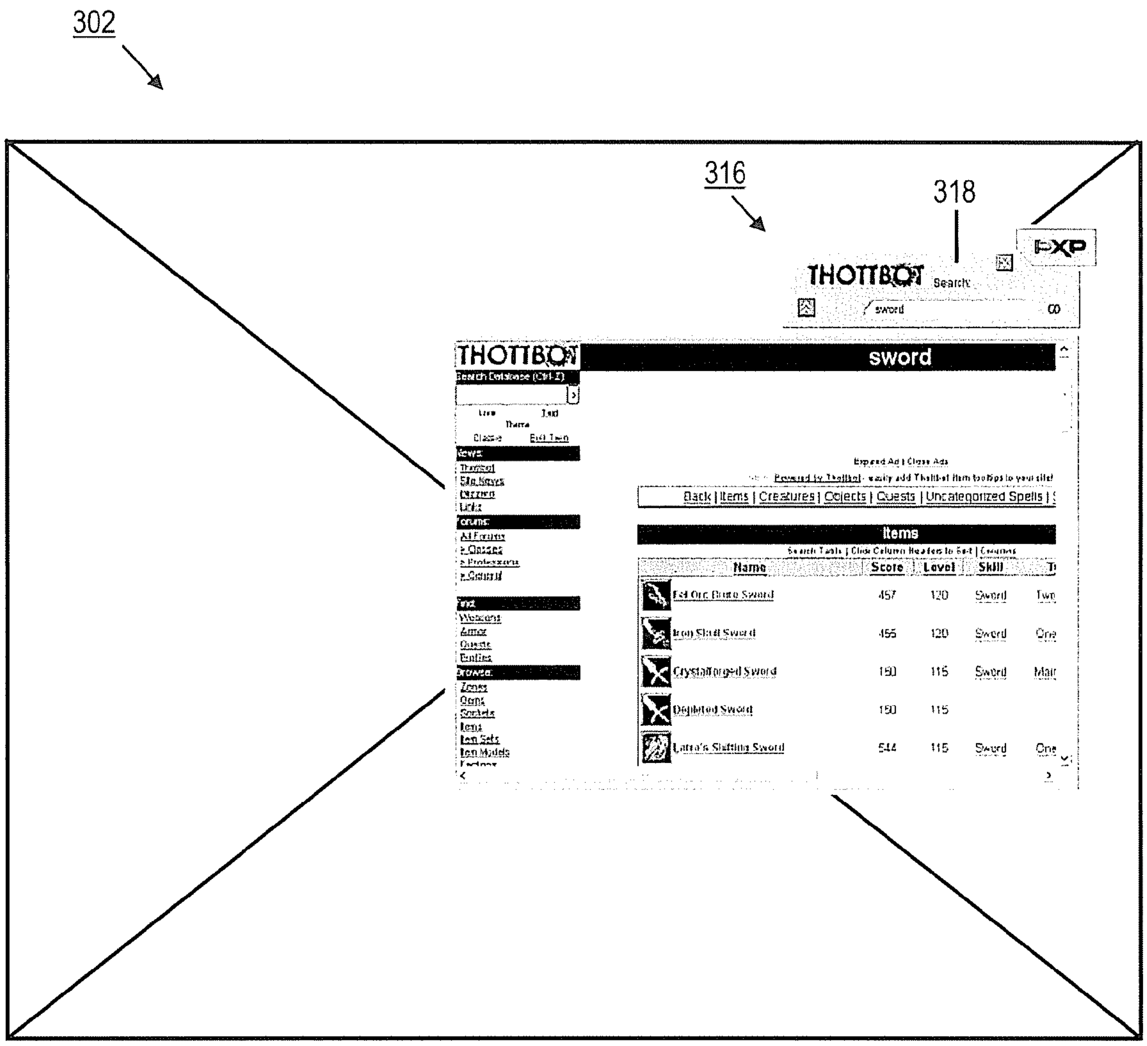


FIG. 8



THOTTBOT **sword**

Search Database (Ctrl-Z)

News:
Thottbot
Site News
Dizzmo
Links

Forums:
All Forums
+ Classes
+ Professions
+ General

Find:
Weapons
Armor
Quests
Entities

Browse:
Zones
Gems
Sockets
Items
Item Sets
Item Models

Expand Ad | Close Ads

Powered by Thottbot - easily add Thottbot item tooltips to your site!

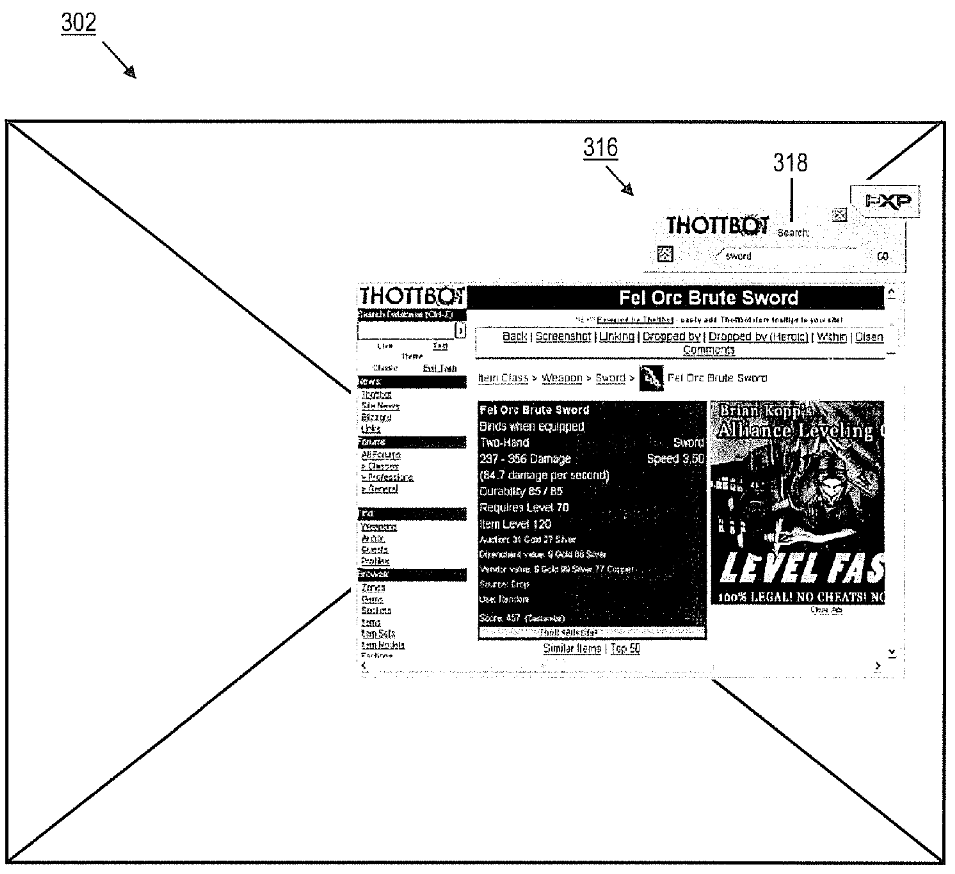
Back | Items | Creatures | Objects | Quests | Uncategorized Spells |

Items

Search Tools | Click Column Headers to Sort | Columns

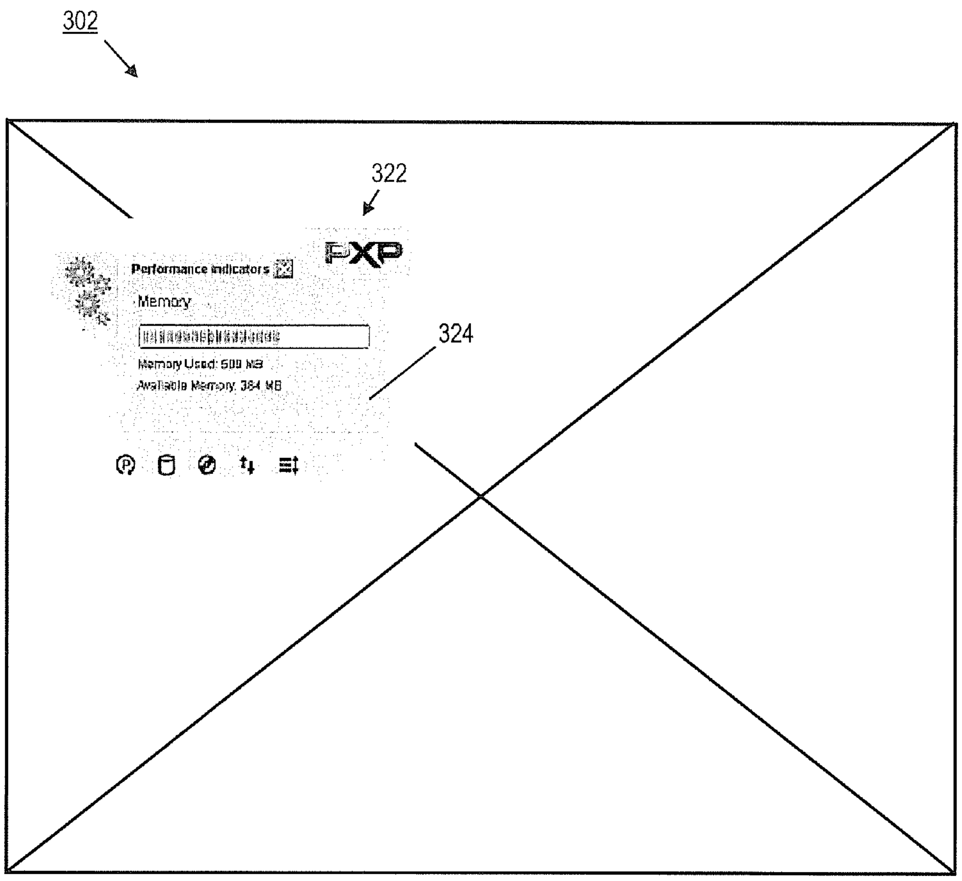
Name	Score	Level	Skill	T
Fel Orc Brute Sword	457	120	Sword	Two
Iron Skull Sword	455	120	Sword	One
Crystalforged Sword	150	115	Sword	Mair
Depleted Sword	150	115		
Latro's Shifting Sword	544	115	Sword	One

FIG. 9



Fel Orc Brute Sword
Binds when equipped
Two-Hand Sword
237 - 356 Damage Speed 3.50
(84.7 damage per second)
Durability 85 / 85
Requires Level 70
Item Level 120
Auction: 31 Gold 27 Silver
Disenchant value: 9 Gold 88 Silver
Vendor value: 9 Gold 99 Silver 77 Copper
Source: Drop
Use: Random
Score: 457 (Desirable)

FIG. 10



Memory Used: 500 MB

Available Memory: 384 MB

FIG. 11

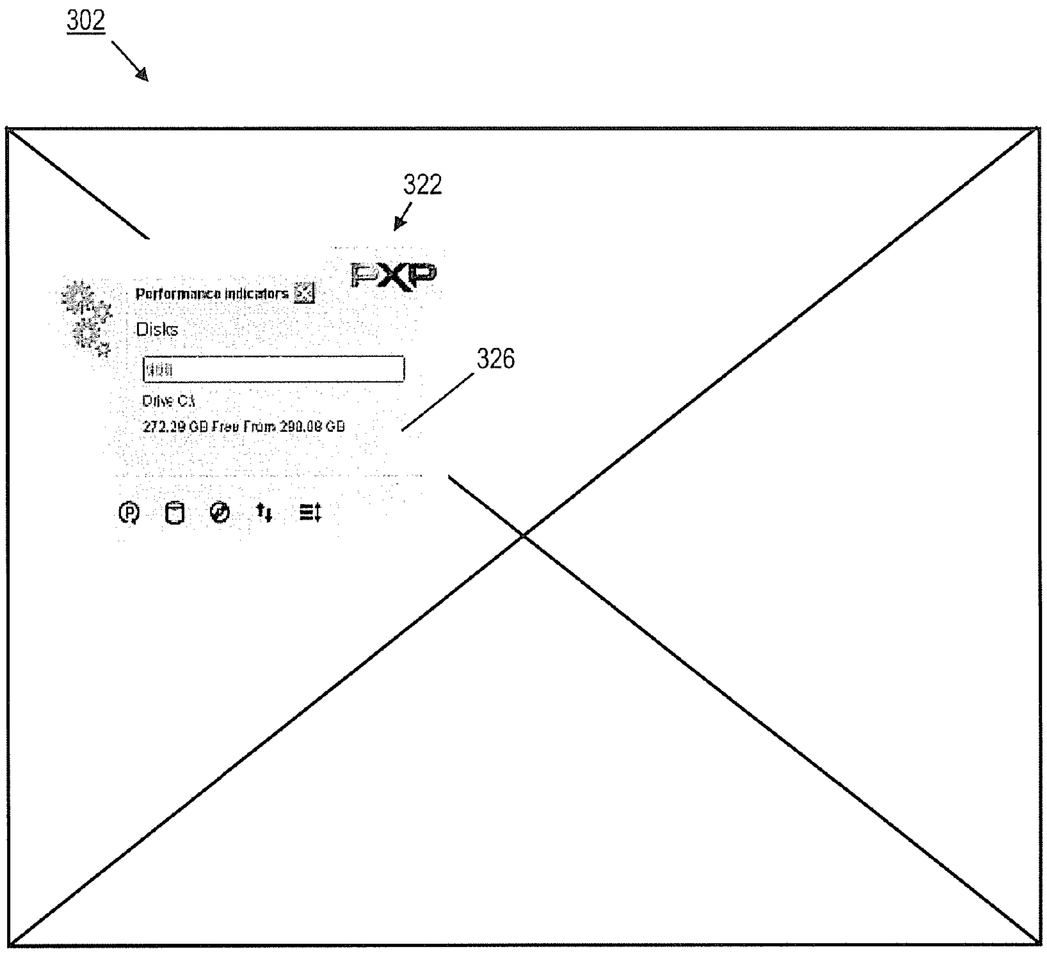

302
322
326
PXP
Performance indicators
Disks
Drive C:\
272.29 GB Free From 298.08 GB

FIG. 12

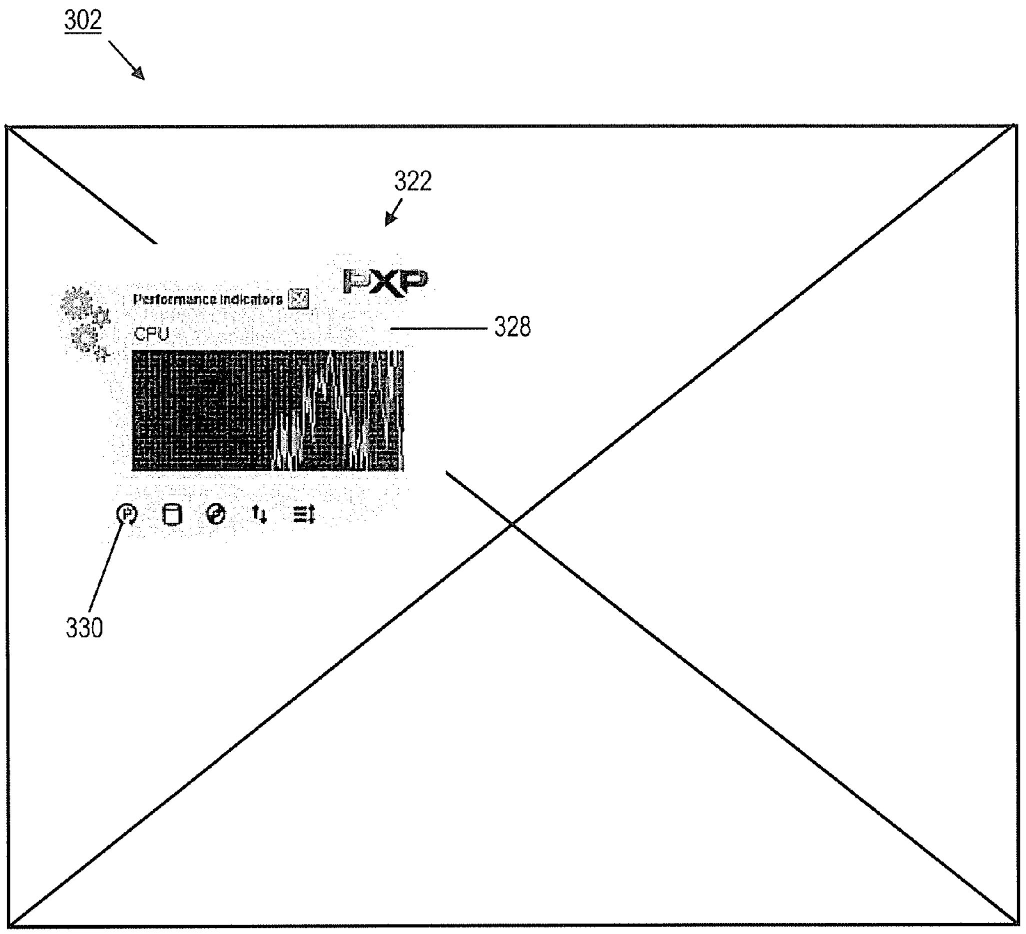
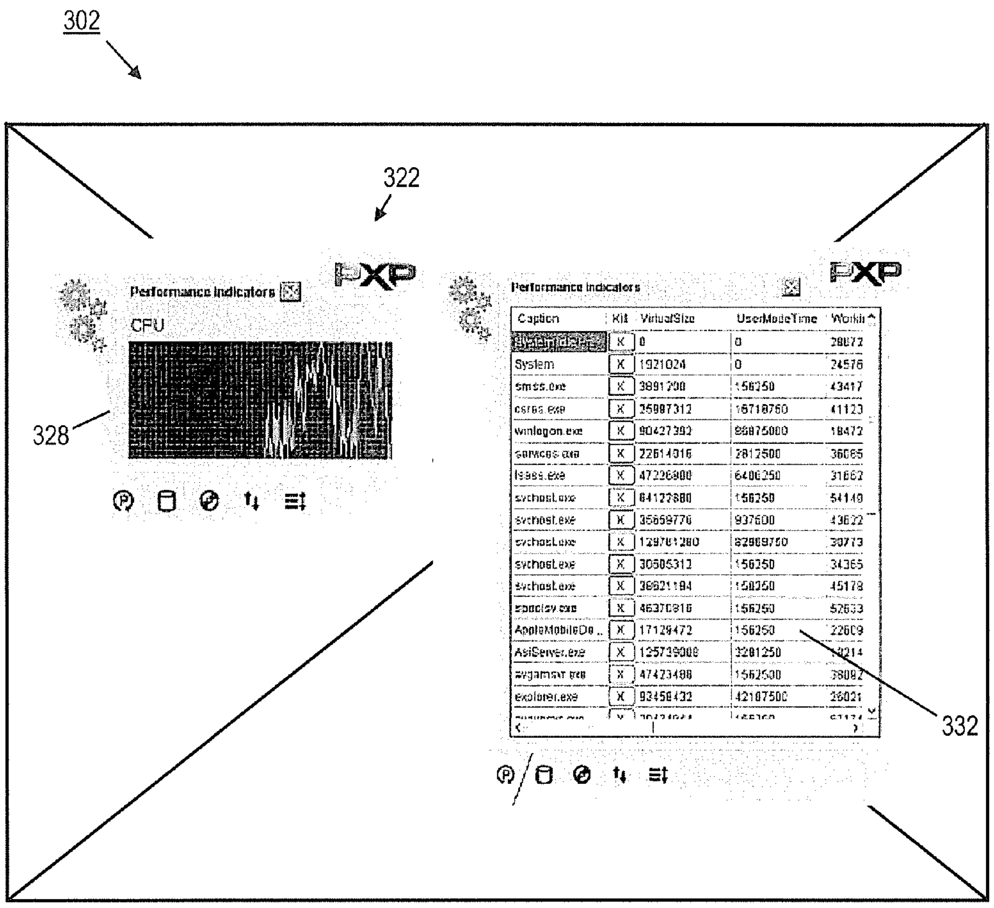
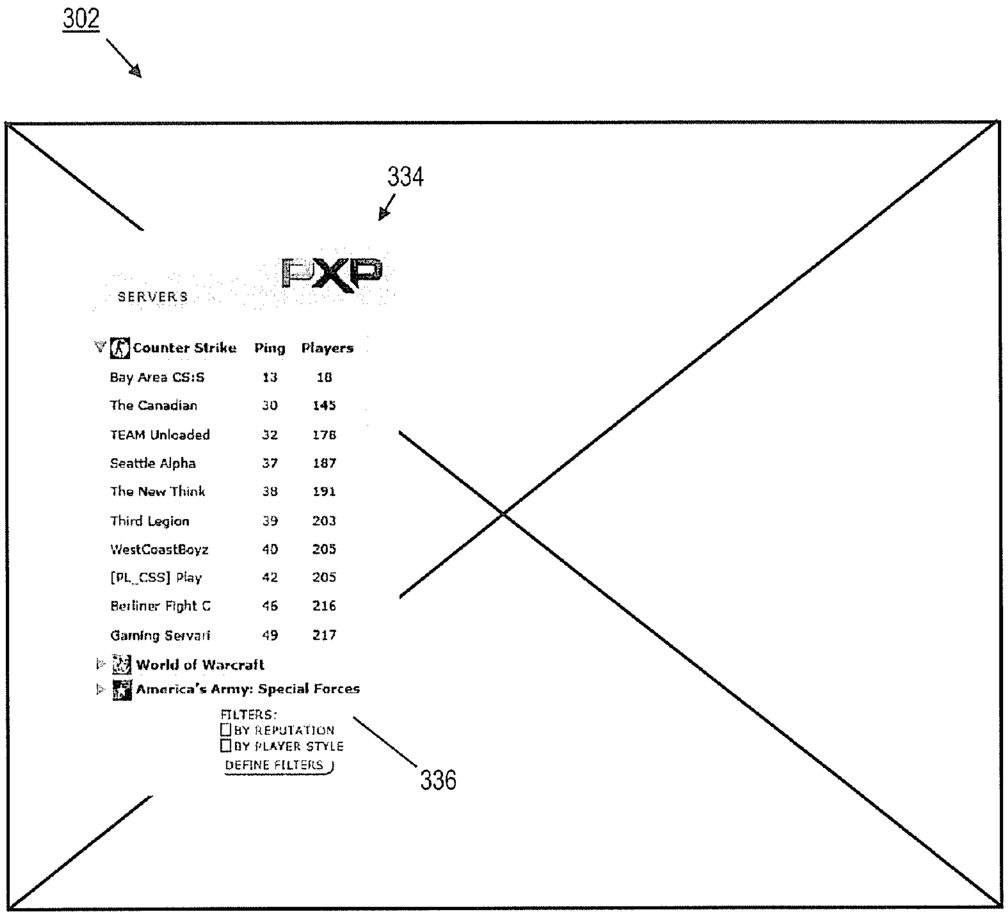


FIG. 13



Caption	Kill	VirtualSize	UserModeTime	Workii
System Idle Pro	X	0	0	28672
System	X	1921024	0	24576
smss.exe	X	3891200	156250	43417
csrss.exe	X	35987312	16718750	41123
winlogon.exe	X	90427392	86875000	18472
services.exe	X	22614016	2812500	36065
lsass.exe	X	47226880	6406250	31662
svchost.exe	X	64122880	156250	54149
svchost.exe	X	35659776	937500	43622
svchost.exe	X	129761280	82968750	30773
svchost.exe	X	30605312	156250	34365
svchost.exe	X	36821184	156250	45178
spoolsv.exe	X	46370816	156250	52633
AppleMobileDe...	X	17129472	156250	22609
AsiServer.exe	X	125739008	3281250	[illegible]214
avgamsvr.exe	X	47423488	1562500	38082
explorer.exe	X	93458432	42187500	26021

FIG. 14

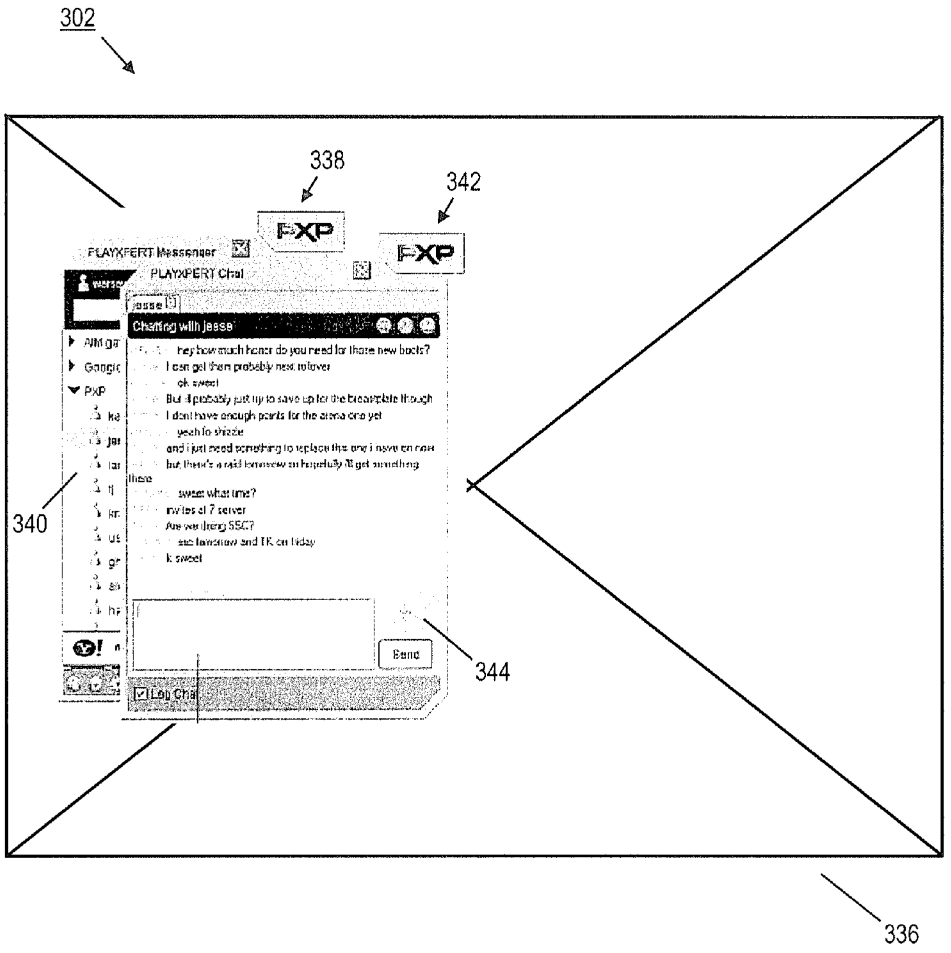


Counter Strike	Ping	Players
Bay Area CS:S	13	18
The Canadian	30	145
TEAM Unloaded	32	176
Seattle Alpha	37	187
The New Think	38	191
Third Legion	39	203
WestCoastBoyz	40	205
[PL_CSS] Play	42	205
Berliner Fight C	46	216
Gaming Servari	49	217

World of Warcraft
America's Army: Special Forces

FILTERS:
☐ BY REPUTATION
☐ BY PLAYER STYLE
DEFINE FILTERS

FIG. 15



hey how much honor do you need for those new boots?
I can get them probably next rollover
ok sweet
But ill probably just try to save up for the breastplate though
I dont have enough points for the arena one yet
yeah fo shizzle
and i just need something to replace the one i have on now
but, there's a raid tomorrow so hopefully ill get something there
sweet what time?
invites at 7 server
Are we doing SSC?
ssc tomorrow and TK on friday
k sweet

FIG. 16

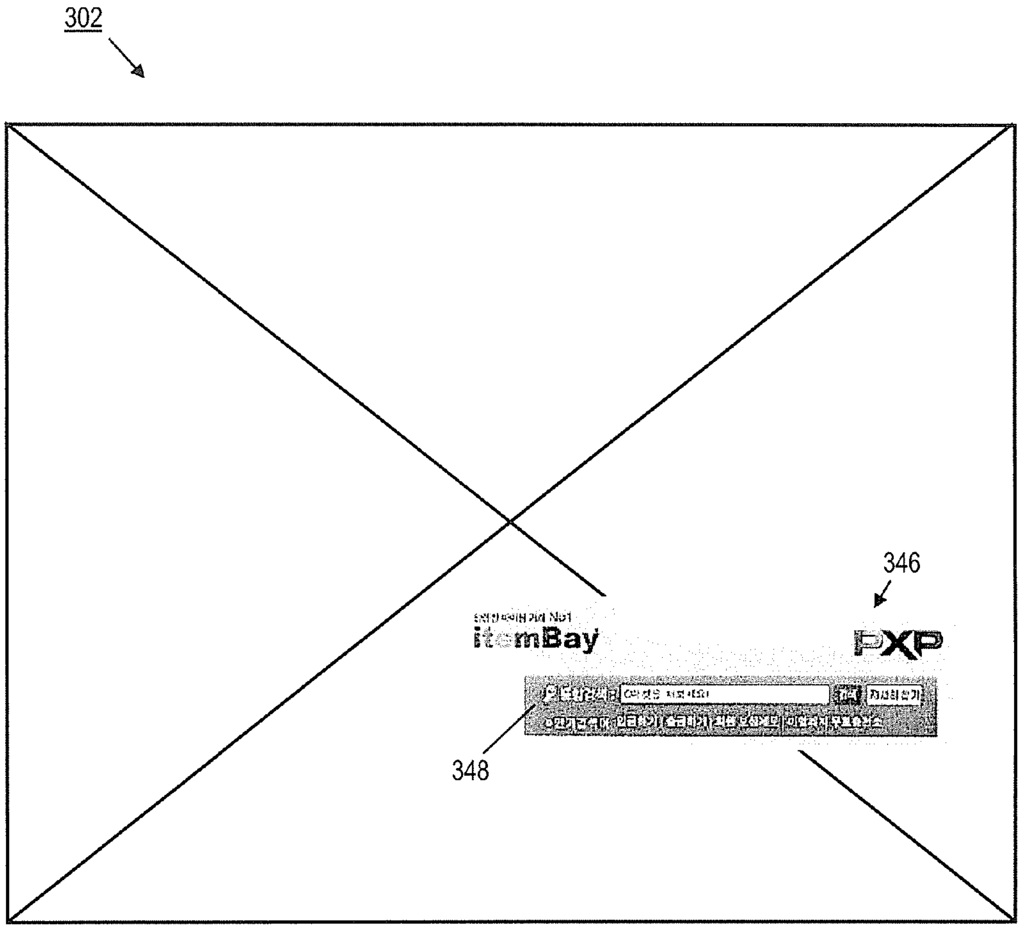


FIG. 17

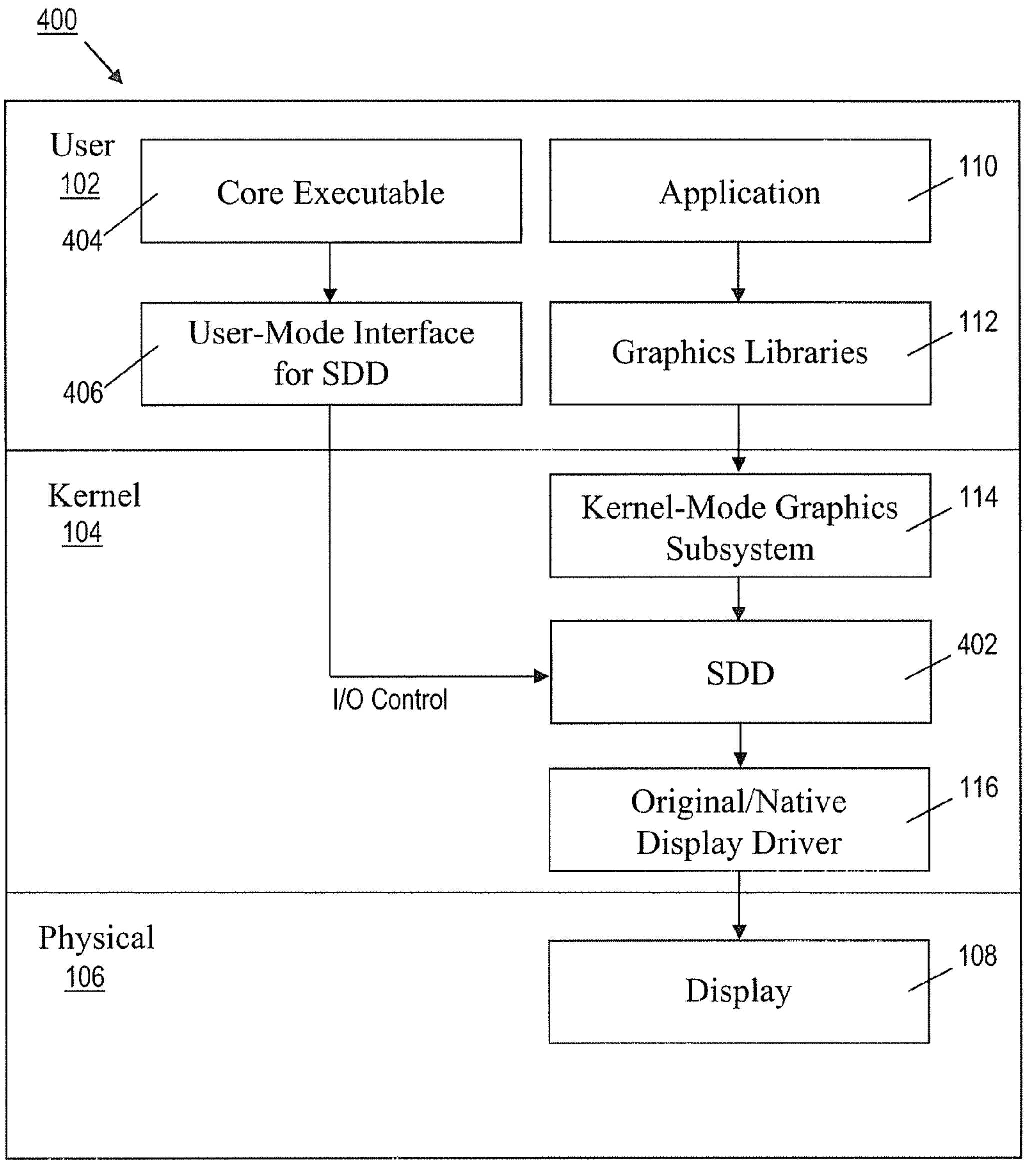

400
User
102
Core Executable
404
Application
110
User-Mode Interface for SDD
406
Graphics Libraries
112
Kernel
104
Kernel-Mode Graphics Subsystem
114
SDD
402
I/O Control
Original/Native Display Driver
116
Physical
106
Display
108

FIG. 18

SYSTEM AND METHOD FOR ADDING IN-GAME FUNCTIONALITY

RELATED APPLICATION

[0001] The present application is being filed as a non-provisional patent application claiming priority under 35 U.S.C. § 119(e) from, and any other benefit of, U.S. Provisional Patent Application No. 60/878,822 filed on Jan. 5, 2007, the entire disclosure of which is herein incorporated by reference.

FIELD

[0002] The invention relates generally to software modules and, more particularly, to a system and method for adding, accessing and/or using software modules within the context of an active computer game, as well as overlaying images and video on the game.

BACKGROUND

[0003] Video games include software that is installed on a computer. An operating system (OS) of the computer can then execute (i.e., "run") the game. In general, the operating system running the computer game causes a graphics card/processor of the computer to display images and video on a display (e.g., an LCD monitor) of the computer. The operating system running the computer game can also cause an audio card/processor of the computer to output sounds and/or music through speakers of the computer. An input device (e.g., a keyboard and/or a mouse) of the computer can be used to allow a user to interact with the computer game.

[0004] Some computer games support on-line play, i.e., exchanging data with other computers (e.g., a client, a server) over a network (e.g., the Internet, a LAN) so that a first user at a first computer can play the game with (e.g., cooperatively or competitively) a second user at a second computer. Furthermore, a Massively Multiplayer Online Game (MMOG) is one type of on-line computer game that is capable of supporting hundreds or thousands of players simultaneously and is played over the Internet. Accordingly, a computer game may provide built-in functions that facilitate the on-line playing of the particular computer game.

[0005] For example, the computer game may have a built in chat function wherein the first user can type a text message or speak a voice message that is then transferred over the network and displayed/played to the second user, and vice versa. In this manner, the first and second users can communicate with one another in the on-line game that they are both playing.

[0006] As another example, the computer game may have a built-in server browser wherein a list of servers that are hosting an on-line session of the computer game are displayed to the first user and/or the second user. Information on each session, such as the type of session (e.g., deathmatch, capture the flag), the number of users present in the session, the current latency (e.g., ping) of the session, etc. can be displayed in the server browser. The server browser allows the first user and/or the second user to join a desired session (e.g., by clicking on its name).

[0007] These built-in functions are limited to the specific game in which they reside, are relatively limited in scope and fail to address many of the issues that arise in the context of on-line gaming. For example, on-line games often give rise to communities of players (e.g., groups, clans, guilds, etc.) that like to play together and may assist one another in and out of the context of a game. As a result, a community of players will often generate a web page for purposes of recruiting members, sharing information among members, etc. The conventional computer games do not have a built-in function that members of a community can use to visit their own web page. Instead, the members must exit out of the game (e.g., in the WINDOWS, a registered trademark of Microsoft, Inc., operating system, change the focus from the game screen/window to another window running a web browser) to navigate to the web page.

[0008] One conventional program that attempts to expand in-game functionality is XFIRE (a registered trademark of XFIRE, Inc.). XFIRE is a computer application that allows a user to send and receive messages from within an active computer game screen/window. XFIRE is not limited to a single game but instead supports messaging in many different games. XFIRE evolved to include other functions such as a clan tool that allows users to register clans and associate users with the clans.

[0009] Another conventional program that attempts to expand in-game functionality is PLAYLINC (a registered trademark of Super Computer International, Inc.). PLAYLINC is a computer application that allows a user to send and receive messages from within an active computer game/screen window. PLAYLINC is not limited to a single game but instead supports messaging in many different games. PLAYLINC supports additional functions such as server browsing, buddy tracking and server hosting.

[0010] These conventional programs, however, have drawbacks. For example, both XFIRE and PLAYLINC fail to intrinsically support inter-network messaging from within a computer game. Users of XFIRE must send and receive messages via the XFIRE network. Users of PLAYLINC must send and receive messages via AOL (America On-Line) or ICQ (also owned by AOL) user accounts. Thus, users are unable to use (messaging) networks/accounts, which they may have previously used for years, to send messages in XFIRE or PLAYLINC.

[0011] Furthermore, neither XFIRE nor PLAYLINC provides a full range of in-game functions to facilitate on-line gaming and the communities that result therefrom. For example, neither XFIRE nor PLAYLINC includes a plugin model which among other things visualize extended interactive tools within a game, for readily extending the functions provided therein.

[0012] Consequently, there is a need in the art for a system and method that supports inter-network in-game messaging, adds in-game functions to facilitate on-line gaming and the communities that result therefrom, and is readily extensible to provide additional interactive tools that are visualized within a game.

[0013] Additionally, in expanding in-game functionality, it may be useful to overlay images and video on top of an active game, which is hereinafter referred to as "video overlay." Conventional techniques for performing video overly on a game, such as those used by XFIRE and PLAYLINC, however, suffer from numerous drawbacks.

[0014] FIG. **1** shows a conventional system **100** for performing video overlay on top of a game running under an operating system of a computer, wherein the system **100** includes a user layer **102**, a kernel layer **104** and a physical layer **106**. Applications, subsystems, etc. in the user layer **102** (i.e., user-mode) have limited access to system resources (e.g., central processing unit (CPU), system memory, external devices). Applications, subsystems, etc. in the kernel layer **104** (i.e., kernel-mode) have unrestricted access to the system resources. A primary purpose of the kernel layer **104** is to manage and prioritize use of the system resources and allow other programs to run and use these system resources. The physical layer **106** includes the physical system resources, such as a keyboard (not shown), a display device **108** (e.g., an LCD monitor) and speakers (not shown).

[0015] An application programming interface (API) must be provided by the developer of an application **110** for performing the video overlay (such as XFIRE) to a publisher of a game on which the application will perform video overlay. The publisher of the game must conform to the API to give the application **110** direct access to the graphics context of the game.

[0016] The application **110** replaces/bypasses a graphics library **112** provided by the operating system (e.g., OpenGL32.dll) in order to intercept graphics calls before they are sent to a graphics subsystem **114** of the kernel layer **104**. In particular, the game's executable files are modified (i.e., patched) in memory to reroute graphics calls to the application **110**. After the game's graphics have been rendered by a graphics device (e.g., a video card) of the computer using a graphics driver (e.g., the display device driver **116**) associated with the video card, the video overlay is performed using a graphics operation (i.e., a bit blit (BitBlt) operation). The bit blit operation combines several bitmap patterns into one using a raster operator.

[0017] As noted above, this conventional video overlay technique has many drawbacks. For example, performance penalties arise from performing the BitBlt operation, which is non-optimal and requires the system **100** to wait until the video commands have been rasterized by the video card. During a standard rendering operation in a game, the game renders to one or more back buffers while presenting a single front buffer to the user. The game attempts to flip these buffers at a regular frequency, although the frequency is sometimes defined by a refresh rate of the display device **108**. Game developers design games to maximize the time interval between a present function call and a flip function call during normal operations, in order to allow the video card adequate time to process the next scene. While the video card is rendering the scene, games are designed to perform game logic processing. By attempting to perform post rendering changes, the system **100** deprives the game of CPU time for processing game logic by blocking the processor until the video card has finished the rendering.

[0018] Additionally, compatibility barriers exist in the conventional system **100** because the system **100** relies on game publishers conforming to an API. For example, XFIRE requires that game publishers expose access points for their games. As a result, XFIRE does not work in all games but only in those games for which the publisher chose to support XFIRE.

[0019] Further still, security penalties arise in the conventional system **100**. For example, the practices of patching a game's executable files and replacing or otherwise bypassing a graphics library provided by an operating system are often performed by individuals interested in cheating in a game. Technologies like PUNKBUSTER (a trademark of Even Balance, Inc.) have been developed to prevent such individuals from modifying a game. Since the practice of modifying the graphics subsystem is the same for both game hackers/cheaters and the system **100**, the system **100** may be identified as a cheating tool resulting in its users being expelled from the games they are playing. Users who are expelled from game play are likely to be dissatisfied with the system **100**.

[0020] Consequently, there is a need in the art for a system and method for overlaying images and video on an active application, which don't give rise to the performance penalties, compatibility barriers and/or security penalties described above.

SUMMARY

[0021] In view of the above, it is an exemplary aspect to provide a system and a method for adding functions that are usable within an active computer game.

[0022] It is another exemplary aspect to provide a user interface for accessing the added functions from within the active computer game.

[0023] It is still another exemplary aspect to provide a system and method for delivering advertisement information in an active computer game.

[0024] It is yet another exemplary aspect to provide a system and method for performing video overlay within an active application, such as a computer game.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] The above aspects and additional aspects, features and advantages will become readily apparent by describing in detail exemplary embodiments thereof with reference to the attached drawings, wherein like reference numerals denote like elements, and:

[0026] FIG. **1** is a diagram of a conventional system for performing in-game video overlay.

[0027] FIG. **2** is a diagram of a system for adding in-game functionality, according to one exemplary embodiment.

[0028] FIGS. **3-17** are drawings of a screen for describing an implementation of an exemplary user interface for the system of FIG. **2**.

[0029] FIG. **18** is a diagram of a system for performing in-game video overlay, according to one exemplary embodiment.

DETAILED DESCRIPTION

[0030] While the general inventive concept is susceptible of embodiment in many different forms, there are shown in the drawings and will be described herein in detail specific embodiments thereof with the understanding that the present disclosure is to be considered as an exemplification of the principles of the general inventive concept. Accordingly, the general inventive concept is not intended to be limited to the specific embodiments illustrated herein.

[0031] A system **200** for adding functionality to a game running on a first computer **202** of a first user **204** and/or a game running on a second computer **206** of a second user **208**, according to one exemplary embodiment, is shown in FIG. **2**. The system **200** includes a server **210** (or multiple servers) that communicates with a client program (not shown) installed and running on the first computer **202** and/or the second computer **206**. The server **210** communicates with the client program over a network **212** (e.g., the Internet). The server **210** may be connected to a data store **214** (or multiple data stores), such as a relational database. The server **210** includes a set of web services that can exchange data with the client program to facilitate server-managed aggregation of data such as user registration data, clan management data, reputation management data, etc. The server **210** may also include a Jabber-based chat interface for connecting to other client programs of other users and other chat programs such as MSN, Yahoo!, Gmail, etc. In this manner, the server **210** facilitates messaging/chatting between two or more users (e.g., the first user **204** and the second user **208**), wherein the user can send and receive messages while actively playing a game running on their respective computers (e.g., the first computer **202** and the second computer **206**).

[0032] The client program running on the computer (e.g., the first computer **202**) of a user (e.g., the first user **204**) provides a user interface **300** for the user to access and use the additional functionality from within an active game running on the computer. See also FIGS. **3-17**. In particular, the user interface **300** is overlaid upon a game screen or display **302** of the active game, as described below, when the user wants to access and use the additional functionality. In one exemplary embodiment, the client program interfaces with the graphics card, processor, driver, etc. of the computer running the game in order to overlay information on the game screen/window and process user interaction with the overlaid information. In this manner, the client program can overlay information in any game without requiring permission from the publisher of the game and without modifying the game itself.

[0033] The additional in-game functionality is provided through one or more software modules, hereinafter referred to as widgets. Each widget provides additional functionality that can be accessed by the user, without requiring the user to leave a game that the user is currently playing. Leaving the game, for example, refers to exiting out of the game or changing from a window/screen in which the game is running to a different window or area (e.g., a desktop).

[0034] An exemplary user interface **300** for use in the system **200** will be described with reference to FIGS. **3-17**.

[0035] A representation of a screen or display **302** of a computer running a game (e.g., WORLD OF WARCRAFT, which is a registered trademark of Blizzard Entertainment, Inc.) is shown in FIG. **3**. In one exemplary embodiment, the visual output of the game (e.g., application **110**) fills the entire display **302**. Accordingly, in describing the general inventive concept, the visual output of the game is merely represented by an "X" spanning the four corners of the display **302**. A portion of the display **302** is designated as a hot spot **304**. In one exemplary embodiment, the size of the hot spot **304** is relatively small when compared to the overall size of the display **302** of the computer. If the hot spot **304** is activated for a predetermined period of time, an activation icon **306** appears on the game display **302**, as shown in FIG. **4**. In one exemplary embodiment, the hot spot **304** is activated by holding a mouse pointer within its borders.

[0036] Once the activation icon **306** is overlaid onto the display **302** of the game running on the computer, the user can select the activation icon **306** to cause the user interface **300** to be overlaid onto the display **302** of the game in place of the activation icon **306**, as shown in FIG. **5**. In one exemplary embodiment, the activation icon **306** is selected by double clicking on it using an input device (e.g., a mouse).

[0037] As seen in FIG. **5**, the user interface **300** extends from a lower right corner of the in-game display **302** and includes a menu bar **308** with various (textual) menu choices **310**. It will be appreciated that the user interface **300** can have any shape or size and can be located anywhere on the in-game display **302**. Similarly, the menu choices **310** can be graphical in addition to, or instead of, textual. Furthermore, the user can configure preferences for the various widgets, such as a size for the widget, a location (on the in-game display **302**) to place the widget, a transparency level of the widget, etc.

[0038] Each menu choice **310** corresponds to a widget or a sub-menu (not shown) with additional menu choices **310**. The user can select a desired one of the menu choices **310**, for example, by clicking on it using a mouse. It will be appreciated that other means of accessing the user interface **300** and/or selecting/activating a widget are encompassed by the general inventive concept. For example, a hotkey (e.g., the "C" key) can be associated with a particular widget (e.g., a "CHAT" widget), such that if the user presses the C key, the CHAT widget is launched.

[0039] If the user selects the "TUNES" menu choice **310**, a corresponding "TUNES" widget **312** is loaded. The "TUNES" widget **312** adds the in-game functionality of playing music from an external source (e.g., C.D., .MP3 file) within the game. As shown in FIG. **6**, a "TUNES" interface **314** of the "TUNES" widget **312** is overlaid on the display **302** of the game and replaces the user interface **300**. The "TUNES" interface **314** facilitates the selecting and playing of the music from the external source.

[0040] If the user selects the "THOTTBOT" menu choice **310**, a corresponding "THOTTBOT" widget **316** is loaded. The "THOTTBOT" widget **316** adds the in-game functionality of accessing a third-party repository of information extracted from and/or relating to the WORLD OF WARCRAFT MMOG (see, www.thottbot.com) from within the game. As shown in FIG. **7**, a "THOTTBOT" interface **318** of the "THOTTBOT" widget **316** is overlaid on the display **302** of the game and replaces the user interface **300**. The "THOTTBOT" interface **318** facilitates navigating the THOTTBOT database, for example, inputting a query to be used to obtain relevant information from the THOTTBOT database. See FIGS. **8-10**. In this manner, the user can look up the location of an item needed to complete a quest in the game without leaving the game, even though the information on the location of the item is stored in a remote database only accessible via the Internet.

[0041] An electronic advertisement **320** is also shown overlaid onto the game display **302** in FIG. **7**. In one exemplary embodiment, such advertisements will appear periodically (when the client program is running on the computer) and disappear after a predetermined period of time. As in the case of a widget, the user can interact with the advertisement without leaving the game.

[0042] Thus, with the system **200**, interactive ads can be integrated into a game without modifying the game. The ads can be presented in a manner that does not distract from the game play experience, such as by overlaying billboards with ads next to a road on the game display **302** in a driving game. The ads can be update periodically, whereas if they were embedded in the game itself they could become stale as the game ages. The ads overlaid on the game display **302** can be selected based on game play achievements or actions and/or external criteria (e.g., current date).

[0043] If the user selects the "DIAGNOSTICS" menu choice **310**, a corresponding "DIAGNOSTICS" widget **322** is loaded. The "DIAGNOSTICS" widget **322** adds the in-game functionality of accessing a repository of information extracted from and/or relating to the computer on which the game (and the client program) are running, from within the game. See FIGS. **11-14**. As shown in FIG. **11**, a "memory" interface **324** of the "DIAGNOSTICS" widget **322** is overlaid on the display **302** of the game and replaces the user interface **300**. The "memory" interface **324** provides details on the memory of the computer (e.g., the total system memory, the available system memory, the system cache and the used system memory). As shown in FIG. **12**, a "disks" interface **326** of the "DIAGNOSTICS" widget **322** is overlaid on the display **302** of the game and replaces the user interface **300**. The "disks" interface **326** provides details on the disk-based storage devices of the computer (e.g., total size, free space). As shown in FIG. **13**, a "processes" interface **328** of the "DIAGNOSTICS" widget **322** is overlaid on the display **302** of the game and replaces the user interface **300**. The "processes" interface **328** provides details on the processes currently running on the computer (e.g., the percentage of CPU capacity being used by user processes and system processes). A portion **330** of the "processes" interface **328** can be selected by the user to show details **332** (e.g., names) of the processes currently running on the computer, as shown in FIG. **14**.

[0044] If the user selects the "SERVERS" menu choice **310**, a corresponding "SERVERS" widget **334** is loaded. The "SERVERS" widget **334** adds the in-game functionality of obtaining a list of servers hosting sessions for one or more different on-line games (e.g., WORLD OF WARCRAFT), from within the game. As shown in FIG. **15**, a "SERVERS" interface **336** of the "SERVERS" widget **334** is overlaid on the display **302** of the game and replaces the user interface **300**. The "SERVERS" interface **334** facilitates displaying and filtering (e.g., by player style, user reputation) the list of servers. Additionally, the user can use the "SERVERS" interface **334** to select one of the servers (e.g., via a single mouse click) in order to join the game being hosted by the selected server.

[0045] If the user selects the "FRIENDS" menu choice **310**, a corresponding "FRIENDS" widget **338** is loaded. The "FRIENDS" widget **338** adds the in-game functionality of obtaining a list of users that have been marked as friends of the user currently playing the game, from within the game. As shown in FIG. **16**, a "FRIENDS" interface **340** of the "FRIENDS" widget **338** is overlaid on the display **302** of the game and replaces the user interface **300**. The "FRIENDS" interface **340** facilitates displaying the list of friends (e.g., the friends can be grouped by clans, etc.). Furthermore, clicking on the name of a particular friend in the "FRIENDS" interface **340** may cause a "CHAT" widget **342** to load, in a manner similar to if the user had selected the "CHAT" menu choice **310**.

[0046] The "CHAT" widget **342** adds the in-game functionality of establishing a chat session for exchanging messages between the parties involved in the chat session, from within the game. As shown in FIG. **16**, a "CHAT" interface **344** of the "CHAT" widget **342** is overlaid on the display **302** of the game and can be on the display **302** at the same time as the "FRIENDS" interface **338**. In one exemplary embodiment, the "CHAT" interface **344** can be positioned anywhere on the display **302** by the user (e.g., by dragging with a mouse). The "CHAT" interface **344** facilitates the creation of messages to be sent to other users and the display of messages received from other users.

[0047] FIG. **17** shows an "ITEMBAY" widget **346** that adds the in-game functionality of establishing a connection to the third-party ITEMBAY web site (www.itembay.com), from within the game. The ITEMBAY web site allows players of various MMOGs to exchange actual physical currency (e.g., U.S. dollars) for virtual items or currency used in the respective MMOG. As shown in FIG. **17**, an "ITEMBAY" interface **348** of the "ITEMBAY" widget **346** is overlaid on the display **302** of the game. The "ITEMBAY" interface **348** facilitates use of the ITEMBAY services from within the game, and shows the use of a non-English language.

[0048] Other widgets can be used to achieve additional in-game functionality. Furthermore, in one exemplary embodiment, a software development kit (SDK) and/or an application programming interface (API) will be made available so that third-parties can create widgets to further extend the additional in-game functionality provided by the client program.

[0049] As noted above, conventional techniques (e.g., as shown in system **100**) for performing video overlay in a game suffer from drawbacks that make such techniques unsuitable for use in the system **200** for adding in-game functionality. Accordingly, the general inventive concept encompasses a system and method for performing video overlay on top of an application (e.g., a game) running under an operating system of a computer. In one exemplary embodiment, a system **400** for performing video overlay on top of a game running under an operating system of a computer includes a user layer **102**, a kernel layer **104** and a physical layer **106**, while the operating system is the WINDOWS operating system.

[0050] Device drivers are software that allow higher-level programs (e.g., the application **110**) to interact with a computer hardware device (e.g., the display **108**), such as through graphics libraries **112** in the user layer **102** and a graphics subsystem **114** in the kernel layer **104** (see FIG. **18**). In the WINDOWS operating system, display device drivers do not adhere to the same model that non-display device drivers follow, i.e., the filter model. For example, file system driver commands (e.g., in packets) are sent down a chain of filter drivers, which are managed by the operating system, and wherein each filter driver in the chain can complete, defer or modify an I/O packet. In addition to and distinct from deferring a packet, a filter driver can pass the packet down to the next filter driver in the chain.

[0051] On the other hand, display device drivers are designed according to a function pointer model. In WINDOWS, a GDI (graphics device interface) subsystem **114** is a kernel-level subsystem that interacts directly with a display device driver (e.g., display device driver **116**) to represent graphical objects and transmit them to output devices such as a monitor (e.g., represented by the display **108** in FIG. **18**). Conversely, user-level services, such as those provided by the graphics library **112**, do not interact directly with a display device driver. The GDI subsystem **114** is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. In the function pointer model, the GDI subsystem **114** queries the initial entry point of the display device driver image and the initial entry point is responsible for returning a table of other entry points. The GDI subsystem **114** then proceeds to call these other entry points directly for various purposes. Thus, in the function pointer model, there is no opportunity to implement OS-managed chaining. Additionally, in WINDOWS, display device drivers (e.g., display device driver **116**) have a very restricted set of system services that they are allowed to use, which further complicates implementation of a non-native video overlay system. Display device drivers do not typically have access to standard WINDOWS NT (new technology) system services, such as registry services and driver communication methods, and have only provisional access to memory mapping services. The NT system services represent routines, which ran entirely in the kernel mode. Further still, buffered I/O, which is the only mode that is directly supported by the GDI subsystem **114**, requires the use of an escape code. Other modes, i.e., "direct" and "neither," are only indirectly available to display device drivers.

[0052] In view of the above, the system **400** includes a supplanting display driver (SDD) **402** located in the kernel layer **104** that supplants the existing display device driver **116**, such that the display device driver **116** is subordinate to the SDD **402**. In one exemplary embodiment, a special section name (e.g., ".SDDN") is added in the image of the SDD **402** at compile time, wherein a dynamic linker can look for the special section name. Driver images (e.g., according to a Portable Executable (PE) file format as set out in the Microsoft Portable Executable and Common Object File Format Specification, revision 8.0 dated May 16, 2006, the entire disclosure of which is herein incorporated by reference) have a specific format, wherein all code and data in the image is stored in a number of sections. Each of the sections has a name, an address, a length and page protection characteristics that describe whether the section can be read, written and/or executed once loaded by a loader service.

[0053] Upon identifying the special section name, the dynamic linker writes a vendor's display device driver name (as a global variable) directly into the special section. The WINDOWS Registry is modified to set the display device driver to be the dynamically linked SDD image. The SDD image refers to the SDD binary executable as it appears in memory after having been loaded by the loader. For example, the relevant registry keys in WINDOWS can include the "InstalledDisplayDrivers" multistring value of the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000 key. One of ordinary skill in the art will appreciate that different keys might need to be adjusted (e.g., any one of the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class†{4D36E968-E325-11CE-BFC1-08002BE10318}\nnnn keys). In order to determine which key, in particular, must be modified, the "EnumnDisplayDevices" function is called, and the "DeviceKey" member of the resulting DISPLAY_DEVICE data structure is used to determine the specific path to the device key that must be opened. Thereafter, the "MatchingDeviceId" key is opened to extract a four-digit number that when combined with the class path provides the full device key to be modified.

[0054] The SDD **402** operates by using a callback thunking system, which gives the SDD **402** the opportunity to control the behavior of the original/native graphics system (e.g., original display device driver **116**) being supplanted. A thunking system is a system wherein function calls are forwarded directly to some other system, without any required inspection or processing of the contents of the function calls. Thus, the thunking system allows the contents of the function calls to be viewed, if desired, while dramatically improving performance when the system is inactive. Once the GDI subsystem **114** loads the SDD **402** and calls an SDD entry point, the SDD **402** loads the original vendor display device driver **116** (e.g., using a custom written Portable Executable (PE) loader, written in compliance with the PE specification). Assuming the role of the GDI subsystem **114**, the SDD **402** records all entry points of the original display device driver **116**. In return, the SDD **402** reports a modified set of entry points to the GDI subsystem **114**.

[0055] In order to implement video overlay services, the SDD **402** can use the function groups listed below. For these functions, the SDD **402** records the original entry points and reports altered points to the GDI subsystem **114**, thereby creating a thunk, which as noted above is a function that generally calls another function. Because it is not necessary to monitor most other entry points, the SDD **402** simply reports these entry points to the GDI subsystem **114** without modifications.

[0056] At runtime, the SDD **402** monitors various functions (see, e.g., the WINDOWS Driver Kit (WDK), RTM Build 6000.16386, released on Nov. 3, 2006, the entire disclosure of which is herein incorporated by reference; see also the WDK documentation accessed at http://www.microsoft.com/whdc/DevTools/WDK/WDKdocs.mspx on Jan. 2, 2007, with a date of last updating of Dec. 12, 2007, the entire disclosure of which is herein incorporated by reference). The WDK details all of the functions that a display device driver written for the WINDOWS 2000/XP display driver model is expected to implement. The functions monitored by the SDD **402** can include: PDEV creation (wherein a PDEV is a logical representation of the physical device); PDEV enabling/disabling; driver load/unload; DirectDraw enable/disable requests (wherein DirectDraw is part of Microsoft's DirectX API and is used to render graphics in applications); Direct3D context creation/destruction (wherein Direct3D is part of the DirectX API and is used to render three dimensional graphics in applications); DirectX exclusive mode notifications (wherein DirectX is a collection of APIs for handling tasks related to multimedia, especially game programming and video, in WINDOWS); surface creation within Direct3D; and all Direct3D drawing operations.

[0057] In order to display a video overlay, the SDD **402** monitors, alters, and injects new tokens into the DirectX command stream. As documented by Microsoft in its WDK, DirectX operates by encapsulating a series of commands in predefined tokens and sending them to the video hardware that executes the rendering operation in compliance with the DirectX specification. For example, the SDD **402** uses members of the D3DHAL_DP2COMMAND enumeration to specify commands to DirectX's Direct3D subsystem. One or more D3DHAL_DP2COMMAND structures are parsed by the Direct3D subsystem, wherein each structure specifies either a primitive to draw or a state change to process.

[0058] A desired video overlay is drawn by creating a Direct3D command stream in memory. Since DirectX allows for a variety of rich graphics scenes to be constructed, the SDD **402** is agile and can take into account the current state of the video hardware before constructing a new stream to inject. First, the stream constructed by the SDD **402** requires some state transitions to be drawn properly. For example, the following states should be properly set before video overlay rendering starts: cull mode; Z-enable; alpha blending; source blending; destination blending; alpha testing and separate alpha testing; Z-function; fog; vertex colors; clipping; lighting (e.g., specular, ambient, diffuse); stencil; multi-sample aliasing; scissor testing; viewport; all matrix transformations; flexible vertex format; vertex shaders; pixel shaders; and all texture stage state information (see, e.g., the WDK). At the end of the video overlay operation, the states of the video hardware (i.e., the state recorded by the display device driver

116) must be restored to the previous state to avoid altering the operations of other applications (e.g., the application **110**) in user mode.

[0059] In order to inject additional command stream tokens, the SDD **402** implements a command stream monitor that interprets each token in accordance with the DirectX specification (see, e.g., the DirectX SDK, version 9.21.1148 published on Oct. 26, 2007, the entire disclosure of which is herein incorporated by reference before the tokens are sent to the display device driver **116**. The token stream includes handles to vertex buffers, index buffers, n-dimensional textures, and tokens that alter the state of the display device driver **116**. It is necessary to determine when to inject the SDD-created overlay stream into the rendering sequence. The SDD **402** is able to make this determination by detecting when DirectX is finished rendering a scene of the application **110**, and is about to start the rendering of a next scene of the application **110**. In a standard double buffered system, two or more separate buffers are used in parallel, such that while a first buffer is read a second buffer is written, and while the second buffer is later read the first buffer can be written. Using a double-buffered system, DirectX selects one of two or more buffers for presenting a current scene (i.e., displaying the contents of the buffer to the user in the form of pixel data on the display device **108**), while one or more of the unselected buffers are used for rendering a next scene. When DirectX selects one of the other buffers in the primary display chain, the SDD **402** knows that the current scene is finished and that it can render the overlay on top of the previously active buffer. The system **400** waits for another member of the flipping chain to become active, as opposed to simply waiting for the selection of any other surface for rendering, since some applications will use secondary contexts to render dynamic elements of the application. A flipping chain is any ordered sequence of buffers containing at least two buffers, in which a front buffer is currently being presented and at least one back buffer is being actively modified (e.g., for later presentation).

[0060] In the system **400**, the command stream monitor also records all state transition information in a complete record known as a state block, which is stored in the SDD **402** as a global variable. The state block is used to determine the state of the display device driver **116** prior to rendering the video overlay command stream and to properly restore the state of the display device driver **116** once video overlay rendering for the current frame is complete.

[0061] Since display device drivers are often designed with low tolerance for error and since the SDD **402** supplants the original display device driver **116** using a method that the original display device driver **116** is unaware of, small deviations can cause system failures. Accordingly, in one exemplary embodiment of the system **400**, the SDD **402** adheres not only to the documented DirectX behaviors, but to undocumented behaviors as well. For example, during surface creation, the state of output fields (i.e., fields the display device driver **116** is expected to fill in) in all structures passed to the display device driver **116** is undefined according to the DirectX specification; however, these states are frequently default-initialized to values that can be determined diagnostically. Some display device drivers depend on these structures being initialized this way and fail when they are not. Consequently, the system **400** performs the default initialization of the states.

[0062] In the system **400**, the SDD **402** resides in the kernel layer **104**, while a core executable **404** and a user interface for the SDD **406** reside in the user layer **102**. The user interface **406** enables an application to interact with the SDD **402**. The system **400** can be implemented as a client program running on a computer (e.g., the first computer **202**) to provide the user interface **300** that allows a user (e.g., the first user **204**) to access and use additional functionality, such as the widgets described above, from within an active game running on the computer. Operation of the SDD **402**, the core executable **404** and the user interface for the SDD **406** is transparent to the application **110** (e.g., the active game).

[0063] The above description of specific embodiments has been given by way of example. From the disclosure given, those skilled in the art will not only understand the general inventive concept and its attendant advantages, but will also find apparent various changes and modifications to the structures and methods disclosed. For example, although exemplary embodiments described herein refer to the WINDOWS operating system, one of ordinary skill in the art will appreciate that the general inventive concept can be applied to other operating systems. It is sought, therefore, to cover all such changes and modifications as fall within the spirit and scope of the general inventive concept, as defined in the claims, and equivalents thereof.

1. A system for overlaying image data, the system comprising:
- a first application running under an operating system;
- a second application running under the operating system;
- a first device driver located in a kernel of the operating system;
- a second device driver located in the kernel of the operating system; and
- a display device,
- wherein the first application is operable to use the first device driver to display a first image on the display device; and
- wherein the second application is operable to use the second device driver to display a second image within at least a portion of a first image displayed on the display device.

2. The system of claim **1**, wherein the second device driver is operable to modify a command stream being sent to the first device driver by a graphics subsystem located in the kernel of the operating system.

3. The system of claim **2**, wherein the operating system is a version of Microsoft Windows.

4. The system of **2**, wherein the graphics subsystem is operable to interact directly with the first device driver.

5. The system of claim **1**, wherein the first application is a video game.

6. The system of claim **5**, wherein the first application is a multiplayer online game.

7. The system of claim **1**, wherein the second image is an advertisement.

8. The system of claim **1**, wherein the second application is operable to add a function that is accessible from within the first application without modifying the first application.

9. The system of claim **8**, wherein the function is at least one of a chat module for sending and receiving data over a network; a music function for playing music from a source other than the first application; an information function for accessing information from a source other than the first application; a diagnostics function for accessing information on a computer on which the first application is running; a server function for displaying at least one server providing an on-line feature for the first application; a friends function for displaying at least one user having been previously associated with a user of the first application; and an e-commerce function for accessing a sales or auction site.

10. The system of claim **8**, wherein the first application and the second application are installed on a first computer,

wherein a second computer is operable to exchange data with the first computer over a network, and

wherein the data is associated with the function that is accessible from within the first application.

11. A method of overlaying image data, the method comprising:

using a first device driver located in a kernel of an operating system to display a first image on a display device; and

using a second device driver located in the kernel of the operating system to modify a command stream being sent to the first device driver.

12. The method of claim **11**, wherein the command stream is being sent to the first device driver by a graphics subsystem located in the kernel of the operating system.

13. The method of claim **11**, wherein the first device driver displays a composite image on the display device based on the modified command stream, and

wherein the composite image includes the first image and a second image within at least a portion of the first image.

14. The method of claim **13**, wherein the first image is associated with a first application running under the operating system and outside the kernel, and

wherein the second image is associated with a second application running under the operating system and outside the kernel.

15. The method of claim **14**, wherein the first application is a video game.

16. The method of claim **14**, further comprising using the second application to add a function that is accessible from within the first application without modifying the first application.

17. The method of claim **16**, wherein the function is at least one of a chat module for sending and receiving data over a network; a music function for playing music from a source other than the first application; an information function for accessing information from a source other than the first application; a diagnostics function for accessing information on a computer on which the first application is running; a server function for displaying at least one server providing an on-line feature for the first application; a friends function for displaying at least one user having been previously associated with a user of the first application; and an e-commerce function for accessing a sales or auction site.

18. An article of manufacture comprising a computer-readable medium tangibly embodying instructions readable by a computer for performing a method of overlaying image data, the method comprising:

using a first device driver located in a kernel of an operating system to display a first image on a display device;

installing a second device driver in the kernel of the operating system; and

using the second device driver to modify a command stream being sent to the first device driver.

19. The article of manufacture of claim **18**, wherein the command stream is being sent to the first device driver by a graphics subsystem located in the kernel of the operating system.

20. The article of manufacture of claim **18**, wherein the first device driver displays a composite image on the display device based on the modified command stream, and

wherein the composite image includes the first image and a second image within at least a portion of the first image.

21. The article of manufacture of claim **20**, wherein the first image is associated with a first application running under the operating system and outside the kernel, and

wherein the second image is associated with a second application running under the operating system and outside the kernel.

22. The article of manufacture of claim **21**, further comprising using the second application to add a function that is accessible from within the first application without modifying the first application.

* * * * *