OBJECT RENDERING FROM A BASE COORDINATE

Inventor: Michael Fleming, San Leandro, CA (US)

Correspondence Address:
CARR & FERRELL LLP
2200 GENG ROAD
PALO ALTO, CA 94303 (US)

Appl. No.: 12/231,994
Filed: Sep. 5, 2008

Related U.S. Application Data
Division of application No. 11/227,272, filed on Sep. 14, 2005.

Provisional application No. 60/661,757, filed on Mar. 14, 2005.

Publication Classification
Int. Cl.
G09G 5/00  (2006.01)

U.S. Cl. ......................................................... 345/629

ABSTRACT
An operating platform- and device-neutral user interface is provided. Through the use of the disclosed user interface, device-particular nuances with regard to the rendering of information are overcome thereby allowing for greater pervasiveness of mobile device usage and reduction in development and management costs through the improvement and consistency of functionality and rendering of information.
OBJECT RENDERING FROM A BASE COORDINATE

CROSS-REFERENCE TO RELATED APPLICATIONS


BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention relates generally to the field of user interfaces. More specifically, the present invention relates to information rendering techniques that achieve substantially uniform rendering of information notwithstanding variances in the hardware and/or software of particular mobile devices.

[0004] 2. Description of the Related Art

[0005] Mobile data access devices make it simple and affordable to access corporate and personal data while out of the office. Software allowing for such access is becoming a standard feature on a variety of mobile devices and platforms: BREW, Pocket PCs, Smartphones, Symbian-based phones, PDAs and Internet browsers.

[0006] There are approximately 35 million workers that make up the ‘mobile workforce,’ that is, individuals who carry out all or substantial portions of their job away from a physical office setting. With the increasing number of on-the-go workers, electronic mail continues to be, arguably, the most important business application. As a result, this workforce—as well as the casual individual user—has an inherent need for mobile access to their electronic mail and other data.

[0007] Despite an ever-increasing need for access to electronic mail and data, costs of ownership for mobile data access remain a barrier. The issue is no longer whether mobile data access is a necessity but whether it can be deployed and managed in an effective manner.

[0008] While cost is an obvious concern in equipping the workforce with the means for accessing data on-the-go, the implementation, development, integration and management of mobile data access solutions are of paramount interest. Despite mobile devices becoming a staple in personal and commercial enterprise, rapidly evolving changes such as number portability, mergers in the telecommunications and software industry and the lack of any one particular technical standard in the mobile device technological space, make providing support for a wide-array of mobile devices an important, albeit difficult, issue with regard to accessing data from a mobile device. The lack of internal expertise, the immaturity of standards, the complexity of integration, device limitations and application development have all been explicitly recognized as barriers to adopting mobile devices for providing access to data while, for example, out of the office or away from a personal desktop computer.

[0009] Increased user-flexibility—user familiarity amongst a variety of different devices and/or platforms—may be provided by device-neutral software as is described in the present application. For example, a single application (e.g., a notepad or an e-mail application) could be run on various mobile devices. The user-flexibility proffered by device-neutral software helps to improve IT-familiarity and expertise in that IT personnel need only become familiar with one software application (or suite of applications) instead of a particularized application for each individual platform environment and/or mobile device. Such device and platform neutrality increases end-user adoption of mobile device technologies in their fullest sense thereby better ensuring a return on investment.

[0010] But as adoption and pervasiveness of mobile devices and operating platforms increase, so does technological fragmentation within the marketplace. That is, with the increasing availability of differing mobile devices and operating platforms, there is an increase in disjunct technologies and methodologies that evidence an increasing need for standardization. Until there exists an overarching technological standard adopted by or at least a significant portion of the marketplace, developing device- and/or platform-neutral applications, as are taught in the present application, for mobile devices makes application development and testing less of a colossal task for software engineers while ensuring higher quality and better overall design.

[0011] Device-neutral user interfaces, like those described in the present application, will play a critical role in mobile device development. Such interfaces must not only provide access to mission critical data but also deal with the realities of variations in screen size, pixel density, aspect ratio and screen use availability amongst devices; limited memory on a mobile device; limited processing power; general quirksiness between platforms; and, perhaps most noticeable to the end-user, the general lack of space for interacting with the mobile device (e.g., keyboard space for text-entry and display space for viewing data). A keyboard, mouse or even a stylus are normally not available for such interaction in a traditional wireless or mobile device. Not only is input difficult, so is viewing a display rendering information. This is especially true when the mobile device happens to also be a cellular telephone.

[0012] Engineers have previously been forced to deal with the fact that present-day prior art interfaces are not be suitable for more than one primary set of devices. For example, PDAs utilize a stylus and touch-screen whereas cellular phones may utilize a keypad and/or five-way navigation. If an engineer is satisfied with limiting an interface to a particular type of environment (e.g., platform or device), the engineer must still deal with the nuances of particular device manufacturers (e.g., a Palm PDA versus a Nokia cell phone) and, in some instances, particular device models (e.g., PDA 710x and Nokia 7110).

[0013] An engineer is still, in many instances, limited by the fact that he or she must pre-generate static interfaces or multiple permutations of the interface as they pertain to a particular device or platform family. This results in delays for
delivery of applications and increased costs in research and development, which inevitably result in increased costs for the end-user.

[0014] There is, therefore, a need in the art for a user interface that is neutral with regard to operating platform and device wherein one client interface will work on multiple platforms and devices.

[0015] There is a further need in the art for a user interface that will intelligently adjust to hardware and software limitations of a particular device or platform so that information displayed on the user interface will maintain a consistent and high-quality appearance amongst devices or platforms notwithstanding the particular limitations of a particular device or platform.

[0016] It should be noted, in the course of this disclosure, that while a device (e.g., hardware) and platform (e.g., software) are recognized as distinct albeit related entities, any reference to a device or a platform should be considered inclusive of both. Similarly, any reference to the neutrality of an interface, in general, should be interpreted as neutrality as to both a device and a platform.

[0017] Further, it should be noted that any disclosed device or platform-neutral user interface is not dependent on the presentation or transmission of communications data (e.g., electronic mail, calendar, SMS) or utilization of user data (e.g., data stored on a desktop).

SUMMARY OF THE INVENTION

[0018] The present invention advantageously provides a virtual platform neutral to physical device or software/hardware operating platform. The virtual platform comprises an abstraction layer that allows for portability across a variety of mobile devices and operating platforms, especially with regard to user interfaces. The virtual platform and abstraction layer and any related software allow for a user interface on a first device to appear and operate substantially similar to a user interface on a second device regardless of differences or limitations that may exist between the operating systems or physical nuances of the two devices. By providing a device-neutral user interface application, a user can move effortlessly between devices should, for example, the need for replacement or repair of a particular device arise or if the user possess multiple mobile devices (e.g., one device for personal use and a second device for work use).

[0019] Additionally, the neutrality of the interface application makes it possible for software developers and engineers to utilize one test suite for a variety of devices or platforms when introducing new features thereby reducing lag-time in delivering applications to market as well as research and development costs. For example, instead of developing five different interfaces for five different devices, one interface may be utilized across five different devices. These reductions in the time and cost of development and delivery inevitably translate into savings for the end-user and/or increases in profit and competitiveness for the application and/or device developer/manufacturer.

[0020] The present invention also provides a layout engine wherein graphics and/or text that are not immediately or wholly compatible with a particular device or platform in their native state can be dynamically altered prior to rendering so that they are ultimately rendered without significant layout errors or disruptions in the user’s viewing of the information.

Methodologies such as coordinate positioning of information and/or vector drawing are also provided.

BRIEF DESCRIPTION OF THE DRAWINGS

[0021] FIG. 1A illustrates an exemplary embodiment of a device platform comprising various operational layers and modules for interaction with a particular device client and as described in the present invention.

[0022] FIG. 1B illustrates a device platform comprising various operational layers and modules for interaction with a particular device client as may be found in the prior art.

[0023] FIG. 2A illustrates an exemplary embodiment of an abstraction layer and a balance of platform-specific code and platform-neutral code as may be found in a device- and/or platform-neutral interface such as that described in the present invention.

[0024] FIG. 2B illustrates a typical balance of platform-specific code and platform-neutral code as may generally be found in the prior art.

[0025] FIG. 3 illustrates an exemplary embodiment of an abstraction layer comprising various informational modules as described in the present invention.

[0026] FIG. 4 illustrates the differences in screen display ratio for two different mobile devices as found in the prior art.

[0027] FIG. 5 illustrates an exemplary embodiment of a virtual platform comprised of a shell program and an abstraction layer.

[0028] FIG. 6A illustrates an exemplary embodiment of a layout engine for controlling a device- and platform-neutral interface.

[0029] FIG. 6B illustrates an embodiment of the present invention wherein a rules engine is integrated with an abstraction layer.

[0030] FIG. 7 is an exemplary embodiment of graphic rendering expressed as a relationship between points on a grid.

[0031] FIG. 8A is an exemplary embodiment of hierarchical graphic rendering.

[0032] FIG. 8B is an exemplary embodiment of rendering information at different resolutions while maintaining substantial quality and appearance.

DETAILED DESCRIPTION OF AN EXEMPLARY EMBODIMENT

[0033] FIG. 1A illustrates an exemplary embodiment of a device including various operational layers and modules for interaction with the device. The present embodiment comprises a platform 110, abstraction layers 120, optional synchronization module 130, user interface 140, and client application 150.

[0034] Some embodiments of the present invention may comprise additional operational layers such as open or proprietary application program interfaces (APIs) that allow software engineers, programmers and even users of a particular platform and/or device to author or install applications that are compatible with the particular platform’s operating environment. A virtual platform and/or layout engine may be embodied in such an application. Some embodiments of the present invention may lack certain operational layers or modules, such as synchronization module 130. Such modules would be absent should a particular device or platform not require, for example, synchronization operations.

[0035] The platform 110 is the underlying hardware and/or software for a particular operating environment. The platform
110 also defines a particular operating environment in which software, hardware and other applications are developed. An example of platform 110 is the Nokia Series 40 Developer Platform. The Nokia Series 40 Developer Platform can utilize platform technologies such as Java™ J2ME. Another example of platform 110 is the Nokia Series 60 and Series 80 Developer Platforms. The Nokia Series 60 and 80 platforms can utilize C++ in addition to Java™ J2ME technologies. The Palm OS® Platform, as another example of platform 110, supports native programming in C and C++ languages as well as Java programming via third-party Java Virtual Machines. The present invention further envisions the future development of operating environments on a variety of platforms.

[0036] Abstraction layer(s) 120 provide basic functionalities and means for accomplishing various operating goals that allow for, in part, the interoperation of the platform 110 with the client application 150 as well as other operational layers such as user interface 140. The abstraction layer(s) 120 provide classes, interfaces, abstract methods and other facilities and resources intended to support various functions and software operations regardless of any particular platform 110 or implementation on any particular device. Abstraction layer(s) 120 may be open or proprietary and are composed of various information modules (e.g., FIG. 3).

[0037] Optional synchronization module 130 comprises the various operational instructions, functionalities and code necessary to allow a particular device or a program residing on such a device to communicate with an external data source, such as a desktop personal computer or enterprise server.

[0038] Communications allowing for a synchronization operation can be achieved in a variety of ways including a cable-to-handset synchronization mechanism whereby the device is physically coupled to a desktop personal computer to allow for the exchange and synchronization of data (e.g., electronic mail). Communications can also be achieved wirelessly whereby an enterprise server (e.g., a Microsoft Exchange Server) configured with appropriate software (e.g., SEVEN Server Edition from SEVEN Networks, Inc. of Redwood City, Calif.) coupled with access to a wireless gateway allows for access to electronic mail and other data by the device without any physical connection. Communications can also be achieved without intermediate server software or gateways (e.g., wirelessly).

[0039] Synchronization should be appreciated in the most general sense (e.g., as a communication event). For example, synchronization may comprise not only maintaining the consistency of data between two points (e.g., real time calendar data on a handheld device and a desktop computer) but also the duplication of data (e.g., received emails at a desktop forwarded to a handheld). Synchronization may also be utilized for the purpose of updating information (e.g., receiving updated software packages, patches and so forth).

[0040] While the optional synchronization module 130 may be necessary for synchronizing the client device and other external data source (e.g., a server), the presence of such a module is not meant to be interpreted as a prerequisite for the operation of a device-neutral user interface.

[0041] The user interface 140 comprises and/or is coupled to various modules and software components and source code to allow for the rendering and operation of a user interface on a variety of devices. The user interface 140 comprises or is otherwise coupled to libraries comprising elements and abstractions such as icons, cursors, scroll bars, sounds, animations, etc. and the necessary software and code to enable their use. In an embodiment of the present invention, the user interface 140 is neutral with regard to a particular device or operation environment. That is, a single interface can operate across a plurality of devices (e.g., Nokia, Kyocera and Treo) and/or environments (e.g., Nokia and PalmOS®) without the need to be reprogrammed for each of these particular devices and/or environment. That is, one user interface 140 fits a broad universe of devices and/or environments.

[0042] The client application 150 resides on any device coupled to a network (e.g., wirelessly) that allows for access to a server device or other computing entity, such as a second client device. Through the coupling of the device to, for example, a server, the user of the device may receive and transmit data such as electronic mail or access data stored at the server. It should further be appreciated that the present invention may also operate in a device that is not coupled or connected to any particular network or second device.

[0043] Small handheld devices are increasingly mobile. This mobility is often a direct result of integrating the handheld device with, for example, a cellular telephone although it is not necessary for the device and related client application 150 to be integrated with a cellular phone or any other particular device.

[0044] Mobile devices are often associated with a particular platform 110. For example, the aforementioned Nokia Series 40 Developer Platform is associated with the Nokia 6101 and 6102 model client devices as well as the Nokia 6020, 6235, 6235i and 6822 model client devices. The Nokia Series 60 Developer Platform, on the other hand, is associated with client devices such as the Nokia 6680, 6681, and 6682 model devices. Similarly, the Palm OS® Platform is associated with client devices such as Xplore™ G18, Kyocera 7135, and the Treo™ 650.

[0045] FIG. 1B illustrates various operational layers for user interaction and general operation within a particular device as may be found in the prior art. Such a prior art device may comprise the actual platform and various operational layers such as synchronization modules, APIs and so forth.

[0046] Prior art devices differ from a device utilized in the context of an embodiment of the present invention in that the client application, user interface and other applications are more integrated, interdependent and operationally incorporated (160) as compared to the present invention (170), which allows for increased flexibility and operability. The 'tightly wound' nature of the prior art is often the result of a general lack of portability of a user interface or any other software between various devices. That is, a particular application, including an interface, is written exclusively for a particular platform and exclusively for a particular device solely in conjunction with that platform. In order for a similar interface with similar functional offerings to operate on another device or platform, that interface must be re-authored in its entirety.

[0047] The exemplary device platform illustrated in FIG. 1A, on the other hand, evidences the ability to transport various functionalities from one platform or device to the next, especially with regard to the design of the abstraction layer 120 as is further discussed in the context of FIGS. 2A and 2B, below.

[0048] It should be noted that while FIG. 1A illustrates various operational layers as separate elements, this is not to suggest a necessary physical differentiation or a general lack of integration in an embodiment. Similarly, the integration of the client, user interface and abstraction layer (160) in FIG. 1B is not meant to suggest a literal, physical integration.
These illustrations are provided merely to aid in the perception of the 'tightly wound' and vertically integrated aspects of the prior art versus an embodiment of the present invention.

For example, and as previously described in the context of FIG. 1B, prior art devices and their related platform and software are generally unitary in nature and are not meant to allow for portability of features, such as a user interface. As such, the prior art code 200 is monolithic in nature and comprised predominantly of platform-specific and application-specific code 210 (e.g., code written for, and only for, a Nokia 6680 device and configured with software written for the Series 60 Developer Platform environment).

This particularized code, while allowing for the integration and operation of a particular device on a particular platform, inhibits the portability of any particular features from one device to another (e.g., a user interface) as may otherwise be provided for with more generalized device/platform-neutral code 220. Such device/platform-neutral code 220 may comprise code written in accordance with particular industry standards or specifications but that allows for the portability or interoperability of a specific and particular feature amongst devices. This neutral code 220 is minimally—if at all—present in prior art devices.

FIG. 2A illustrates an exemplary embodiment of an abstraction layer 250 and a blend of platform-specific code 260 and platform-neutral code 270 as may be found in a device-neutral user interface allowing for the consistent rendering of information amongst devices and platforms regardless of the particular operating environment.

An abstraction layer 250, as may be found in an embodiment of the present invention and as illustrated in FIG. 2A, exhibits a much ‘thinner’ layer of platform- or device-specific code 260. In some embodiments of the present invention, platform specific code may be entirely non-existent. Abstraction layer 250, with its thin layer of platform- or device-specific code 260 may be, generally, the type of abstraction layer 120 as described in FIG. 1A.

As the abstraction layer 250 comprises more platform- or device-neutral code 270, the portability or interoperability of particular features—including a user interface providing for uniform and consistent rendering of information amongst various devices—is increased in that a feature (e.g., an application or function) will operate on various platforms or devices due to its coding being dependent more on the generalized code 270 than with platform- or device-specific code 260 that limits or inhibits portability or interoperability.

FIG. 3 illustrates an exemplary embodiment of an abstraction layer 310 comprising various informational modules 320-350 as may be implemented in the abstraction layer 250 illustrated in FIG. 2A.

Informational modules 320-350 comprise routines and instructions as they pertain to various operational features of, for example, a particular platform 110 and/or client application 150 linked in the abstraction layer 310. These modules link the particular device to the particular platform.

For example, resource module 320 may comprise specific data or routines utilized in the operation of platform 110, client application 150 and/or device; for example: sleep mode, power on and off in addition to bitmaps, layouts and other libraries of information that are stored on the device or the means for accessing the same.

Graphics module 330 may comprise the information, instructions or knowledge with regard to utilizing specific files such as JPEGs, bitmaps or other graphic data that could be utilized by user interface 140 in its rendering of a user interface on a device. The graphics module 330 may retrieve these files from resource module 320.

Event module 340 may comprise a library of information, instructions or knowledge with regard to identifying actions or occurrences as may be detected by a particular program such as user actions (e.g., pressing a key) in addition to system occurrences (e.g., an internal calendar alarm) and how to translate them across various environments (e.g., as if they were executed in a native environment).

Sound module 350 may comprise the information, instructions or knowledge of how to play or emit various sounds (e.g., WAV files) to be generated in response to, for example, the occurrence of certain system events (e.g., system warnings concerning low battery power). Sound module 350 may retrieve that particular file from the resource module 320.

Abstraction layer 310, as it corresponds to abstraction layer 120 (FIG. 1A) and abstraction layer 250 (FIG. 2A) may comprise additional or fewer modules as is required by the particular platform 110 and/or device and/or client application 150. It should also be noted that while FIG. 3 illustrates various modules as separate elements, this is not to suggest the requirement of a physical differentiation or a general lack of integration in an embodiment of the present invention.

FIG. 4 illustrates the differences in screen display ratio for two prior art mobile devices, specifically a TREOTM 650 410 and a Nokia 6680 420. In the case of the TREOTM 650 mobile device 410, the screen display offers 320x320 pixel-width with 16-bit color; the display offers approximately 65,000 colors. In the case of the Nokia 6680 mobile device 420, the screen display offers 176x208 pixel-width with active matrix; the display offers approximately 262,144 colors.

The prior art devices of FIG. 4 are indicative of the problems often associated with, especially, graphic elements rendered on different client devices with different display ratios. For example, a graphic may be approximately 300 pixels in width and renders without complication on device 410 with a 320 pixel-width. That same graphic, in the context of device 420 with a 176 pixel-width, however, may be distorted 440 in that it is 'cut off' or 'wrapped around' due to the limited screen width. This distortion is often the result of different devices and/or platform rendering the same graphic. This distortion can be especially problematic in the context of user interfaces offered by third-party software service providers either for functionality and/or branding purposes.

The device-neutral user interface described herein aids in preventing inevitable pixel variances and other differences between devices and/or platforms from resulting in the distorted graphic and informational images. The device-neutral user interface will specify a particular layout but also provides for adjustment of the interface depending on the particular nuances of any particular platform or device, for example, screen width. These adjustments may be relative (e.g., as a result of screen width) or 'as needed' or 'dynamic' per the particular demands of a user of any particular device. Exemplary methods for screen adjustment are disclosed in...
co-pending U.S. patent application Ser. No. 11/227,013, which has been incorporated herein by reference.

[0065] FIG. 5 illustrates an exemplary embodiment of a virtual platform 500 comprising a shell program 510 and an abstraction layer 520. In some embodiments of the present invention, the abstraction layer 520 and the shell program 510 may be a single module of software.

[0066] Abstraction layer 520 is similar to the abstraction layer 310 described in FIG. 3. Abstraction layer 520 interacts with the shell program 510 to effectively translate or otherwise offer portability of commands or instructions issued by a device-neutral interface or other platform environment as if the commands were actually issued in the native platform associated with the client. For example, if an event 530 (e.g., a graphic rendering instruction) occurs in a particular platform environment (e.g., the Nokia Series 40 Developer Platform) that event 540 might—and likely will—substantially differ in structure and content (i.e., syntax) relative to a different platform (e.g., the Palm OS®).

[0067] The virtual platform 500 is capable of normalizing syntax (e.g., code) of the two different platforms environments into a common format (e.g., a common syntax format with reliance semantic structure). That is, the virtual platform 400, in conjunction with abstraction layer 520, provides the necessary translation so that the syntax of the two platforms (e.g., code related to a graphic rendering instruction) may be reconciled to achieve the related semantic purpose (e.g., invoking the rendering of a graphic in accordance with the instruction event 530) in, for example, a device-neutral interface.

[0068] The event 530 or certain information generated by the event 530 (e.g., a notification of the event) is, in certain instances, intercepted by the shell program 510. In some instances, the event 530 may be "passed" upon by the shell program 510. This "pass" may be the result of the event 530 not requiring translation or platform 500 and shell program 510 not being concerned with the particular event 530. This "pass" determination may be the result of certain manual programming of the platform 500 before or after it leaves an original equipment manufacturer or as the result of training, updating by the user or installation of software patches and the like.

[0069] The shell program 510, should it intercept the event 530, prevents the event 530 or the information generated by the event 530 (e.g., a notification of the event) from being immediately handled by any relevant logic on the actual device or platform. The abstraction layer 520 then processes the event 530 intercepted by the intermediary shell program 510 and determines the proper response, reaction and/or instruction 540 to the event 530 for the particular device and/or platform hosting the virtual platform 500.

[0070] The proper response, reaction and/or instruction 540, in some instances, will be to translate the event 530. The proper response 540, in other instances, will be to pass the event 530 on to some other aspect of the device for management. The proper response 540, in yet another instance, may be to "null" the event 530 and not allow it to be processed or translated by the platform 500 and/or any other element of the device.

[0071] An event 530 generally falls into one of three categories. The first category may generally be described as a one-to-one translation. That is, the event 530 occurs and results in a particular reaction. For example, a button is pressed and a character (eventually) appears on the screen. This reaction is the result of the event 530 (or a notification of the event 530) notifying the appropriate device elements of the occurrence (the button press) and/or invoking the necessary code and/or routines to generate, for example, the aforementioned character.

[0072] It should be understood that the event 530 and the eventual response 540 are not necessarily a direct relationship (e.g., the button press does not directly cause the appearance of a character on the screen). The button press, instead, may be recognized by the device, a notification of the recognition of the occurrence thereby causing the execution of certain instruction sets that, in turn, cause a display or graphics module to render the letter ‘A’ on the display screen.

[0073] The second category of event 530 may generally be described as a synthetic event. In this instance, an action is recognized but the related function is not immediately present. The function, in this instance, must be synthesized to correspond to the event 530. For example, a particular command in an interface environment may be recognized but not present on a particular device. In this case, the issuance of the particular command causing the device to undertake the desired action would be synthesized and executed.

[0074] The third category of event 530 may be described as an ad hoc synthetic event wherein a series of actions occur internally. That is, one event 530 (the button press) results in the generation of a second event 530 (the execution of command code), which in turn results in the occurrence of some action by another element (e.g., hardware or a software module) of the device.

[0075] It should be noted that in some instances, the proper response/reacton 540 may be inaction. That is, the platform 500 does nothing in response to the event 530. Similarly, the platform 500 may take a ‘wait-and-see’ approach and wait for the occurrence (or non-occurrence) of a subsequent event 530. This ‘wait-and-see’ approach would be apropos in the instance of a timer-related situation such as triple-tap text entry. Ultimately, the appropriate response/reaction 540 will be dependent upon the context of the event 530 as may be governed by, for example, a particular software application.

[0076] For example, the aforementioned button-press in a Nokia Series 40 Developer Platform operating environment may be equated to activating a backlight for a display screen. In another operating environment, however, the button press may be associated with sending a device into a ‘sleep’ state or may lack an associated function altogether. Absent the virtual platform 500, a user-interface would be unable to communicate the semantic content of the button press (e.g., undertake a particular action or cause a particular result) to both the Nokia platform and an alternate platform, such as the Palm OS®, as the syntax between the two platforms would differ.

[0077] Utilizing the virtual platform 500, however, the shell program 510 (in a Nokia platform environment, for example) would interpret and recognize the button press event 530 as indicative of the user’s desire to enter sleep mode and communicate with the abstraction layer 520 in order to translate the event 530 into the proper response 540 for a Nokia-related device, which may normally be associated with a double press of another button. Similarly, the same virtual platform 500, when installed on a Palm OS® device could aid in translating the event 530 into a response 540 as recognized by a Palm OS® related device. A command issued by or in the context of a non-native device-neutral interface is recognized and translated, if necessary, for processing as if initially issued in the native device/platform environment. For
example, a user could issue a sleep command as associated with a particular button as proffered by the device-neutral user interface and that button press, in part because of virtual platform 500, will be translated and recognized on a multitude of devices and/or platforms.

[0078] Similar functionality is applicable as it pertains to the rendering of graphic and/or textual information. For example, event 530 may comprise information related to the rendering of a certain graphic image (e.g., a bitmap file, vector graphic instructions or coordinate mapping instructions) with particular colors, size, shading and so forth. The virtual platform 500 will provide for rendering the graphic information in a substantially similar manner notwithstanding the particularities of a particular device and/or operating environment. The event 530 is intercepted by the shell program 510 and determination are made as to whether the event 530 may be ‘passed’, processed ‘as is’ or requires certain modifications such as relative adjustment to allow for proper rendering. This operability is further discussed in the context of the layout engine 600 in FIG. 6A and FIG. 6B.

[0079] FIG. 6A illustrates an exemplary embodiment of a layout engine 600 for controlling a device- and platform-neutral interface as may be found in the present invention. Layout engine 600 comprises a rules engine 620 and a logic engine 630. An embodiment of the layout engine 600 provides intelligent flexibility for adjusting interface layout (e.g., spatial interrelationships between elements and/or information and/or structural aspects therein) to fit multiple screen sizes, densities and aspect ratios.

[0080] Rules engine 620 comprises a variety of defined constraints with regard to the display of user information on the display of a device. For example, rules engine 620 may be programmed to understand that the particular device on which the rules engine 620 resides has a limited screen size in terms of pixels or limitations with regard to the number of colors the display can render. Other rules may include this display of certain language or file formats (e.g., HTML, *pdf, or *.ppt). Additional rules may be related to limitations on dedicated processing power for the rendering of any particular graphic as it pertains to the general operation of the device or during a particular operation (e.g., while downloading content from a website).

[0081] The constraints delineated in the rules engine 620 can be installed by an original equipment manufacturer and may be subject to user adjustment (e.g., deactivating default settings). Constraints in the rules engine 620 may also be updated automatically during the operation of the device or configured as the result of intelligent determinations by the device.

[0082] For example, if a rules engine 620 determines that it is resident on a device for which it does not know the pixel limitations of the display, it can make certain intelligent assumptions as to the display size. The rules engine 620 might recognize that the layout engine 600 is resident on a Nokia 6600 Series phone but not that it is on a Nokia 6680 phone, in particular. From the rules engine’s 620 knowledge of the Nokia 6600 Series, it can make an assumption that the pixel limitations are “at least” or “at most” certain numbers. As a result, the layout engine 600 may not produce an optimized graphic image on the device but at least one that is uniform and consistent amongst various devices and displays the best possible quality in light of the limitations of the particular device.

[0083] The rules engine 620 can also receive new updates with regard to device information during a synchronization operation with a desktop PC or server that hosts other programs related to the device (e.g., a mail forwarding program that forwards mail from the desktop to the mobile device). These updates might be downloaded at the desktop PC or server automatically or as a result of the user affirmatively downloading an upgrade or patch from the appropriate provider of that information (e.g., the device manufacturer or the user interface designer).

[0084] The rules engine 620 can also request the user manually provide this information if an assumption or synchronization operation fails to provide the necessary information.

[0085] An input request 610 from the user of the device or a program running on the device comprises a request to display certain information on the device. The input request 610 is similar to the event 530 disclosed in the context of virtual platform 500 in FIG. 5 and, in that regard, certain functionalities of the virtual platform 500 may be integrated into the layout engine 600. For example, input request 610 may consist of a request or instruction to render a text box of x*y pixel size and/or of a particular color. This request may be generated by the user during the course of using a drawing application. Similarly, this request may be generated by a particular program as a result of the occurrence of a particular event, for example, an alarm indication that generates a text box indicating a certain event is about to begin.

[0086] The input request 610 need not be of any particular format or language so long as it may be processed by the layout engine 600 with regard to determining whether the particular text and/or graphic event may be displayed on the device in accordance with requested size, color, configuration, etc.

[0087] The layout engine 600 also comprises the aforementioned logic engine 630. The logic engine 630, based on an input request 610, will query the rules engine 620 to determine if the particular input request 610 may be processed as requested on the particular device or if some adjustments will be required with regard to the limitations of the device as set forth in the rules engine 620. For example, an input request 610 might request the display of a text box of x*y size and of a particular shade of aqua. The layout engine 600’s logic engine 630 will identify the requested parameters (e.g., size and color) and make a query of the rules engine 620 to determine if the particular device hosting the layout engine 600 can accommodate the request 610. If the rules engine 620 reflects that the request 610 can be processed and subsequently rendered without violating a particular rule, the logic engine 630 will approve the request 610 thereby resulting in an output instruction 640 to execute or effectuate the execution of the rendering of a text box of x*y size and the aforementioned shade of aqua.

[0088] Output instruction 640, like the input request 610, is not of any particular format or language so long as it may be generated by the layout engine 600 with regard to indicating that a particular text and/or graphic event may be displayed on the device in accordance with requested size, color, configuration, etc. Instruction 640 only needs to be capable of being processed by the appropriate component of the device providing for the display of the text and/or graphic event (e.g., a graphics or rendering engine (not shown)).

[0089] Should the logic engine’s 630 query of the rules engine 620 determine that the requested text and/or graphic
event cannot be displayed on the particular device, the logic engine 630 may further query the rules engine 620 to determine what the particular constraints of the device are with regard to the rejected event (e.g., the device cannot display aqua but can display light blue). This information may also reside directly in the logic engine 630 or at a locale on the device accessible by the engine 630. For example, information pertaining to commonly requested display events might be cached in the logic engine 630 or in memory (not shown) accessible by the logic engine 630.

Similarly, the logic engine 630, in certain embodiments, may be trained whereby the logic engine 630 begins to recognize a repeated display event and without query to the rules engine 620, understands that such a display event is impossible or otherwise violates the rules of the device as set forth in the rules engine 620. Through the training of the logic engine 630 and the now absent need for continued queries to the rules engine 620, the processing speed of a display event may be increased.

The logic engine 630, in some embodiments, may also be expressly instructed by the user (e.g., through programming or a query during processing) to respond to a particular violation of a constraint set forth in the rules engine 620 in a particular manner. For example, if the request 610 pertains to the display of aqua but the device can only display light blue, the user might pre-program the logic engine 630 to display sea-foam green instead of resorting to light blue.

Once the logic engine 630 determines the constraints of the particular device in conjunction with the requested event as reflected by the input request 610, the layout engine will generate the output instruction 640 that best reflects the scope of the initial request 610 but while remaining within the particular constraints as set forth by the rules engine 620 or, in some embodiments, as directly instructed by the user. For example, the logic engine 630 may resort to the aforementioned example of light blue versus aqua.

By further example, if a request 610 pertains to the display of a graphic or text information that exceeds the size of the actual device, the logic engine may determine what information is necessary to be displayed to carry out the scope of the initial request 610.

For example, request 610 might pertain to displaying a user's contacts directory. On one device, the display of the directory might normally result in the concurrent display of the date and time as well as a telephone icon whereby a user can highlight a particular name in the contact directory and then 'tap' the telephone icon resulting in the phone dialing the number of the person in the contact directory (i.e., a speed-dial feature).

If the physical limitations of a particular device are such that the time and date, directory and speed-dial icon cannot all be displayed, the logic engine 630 will determine what information is critical to the scope of the request 610 and, operating within the confines of the rules engine 620, generate an output instruction 640 that will result in, for example, the relocation of the speed-dial icon on the display to a more efficient space, the reduction in size of the contacts directory (or the display of only a limited number of names in the directory) and the total removal of the date and time from the display during this particular operation.

An embodiment of the layout engine 600 may also provide for cross-representation of resources such as bitmaps, templates or screen layouts, animations and sounds.

As illustrated in FIG. 6B, the layout engine 620 of the layout engine 600 may be integrated with the abstraction layer 520 of the virtual platform 500 that allows for the interoperability of a particular user interface on any variety of devices and/or platforms. While the layout engine 600 and virtual platform 500 need not necessarily be physically integrated, the device-neutral user interface of the present invention requires that the two components at least be capable of communicating with one another as to allow for the translation of what might be a foreign instruction by the virtual platform 500 into an instruction otherwise comprehensible by the layout engine 600. Other integration schemes are envisioned by the present invention and are not meant to be limited to the embodiment depicted in FIG. 6B.

In some embodiments, the layout engine 600 may be further integrated with a cross-platform events engine as is described in co-pending U.S. patent application Ser. No. 11/227,323, which has been incorporated herein by reference.

In some embodiments of the present invention, the rendering of a device-neutral user interface will be effectuated utilizing vector graphics although the rendering of the user interface information may also occur through the use of other graphic rendering techniques. Vector graphics represent those graphic images generated from mathematical descriptions that determine the position, length, and direction in which mathematically-describable objects—such as lines, ellipses, rectangles, rounded rectangles, abstract polygons, filled and non-filled regions, gradients, fountain fills, Bezier curves and so forth—are drawn. Unlike raster graphics, objects are not created as patterns of individual dots or pixels. Through utilizing vector graphics, the 'look and feel' of a particular interface is maintained across platforms and devices thereby resulting in increased scalability as each element is stored as an independent object.

Vector graphics also aid with regard to 'skinning' whereby the look of a particular platform or software program is changed but its underlying functionality remains unaltered. Through the use of skinning, opportunities for branding, advertising, and user customization are also increased. Skinning also allows for platform independence whereby one customized user interface can be ported to various devices or operating platforms and because of the utilization of vector graphics versus rasterization or bitmapping, that one interface can be scaled and adjusted as necessary by, for example, a layout engine 600 and/or virtual platform 500. The end result of using vector graphics is that 'real space' remains consistent and relative.

Graphic renderings may also be expressed as a relationship between a particular point and its location on a Cartesian grid (e.g., a grid system). For example, FIG. 7 illustrates such a Cartesian grid 700. In such a rendering system, a base coordinate 710 is first identified that will serve as the starting point (either directly or indirectly) for all other graphic information rendered on a display. In the presently illustrated embodiment, all points on the Cartesian grid are expressed in the form of pixels. Other embodiments may utilize any type of scaling unit so long as it provides a consistent basis for determining distance between points.

Rendering a graphic from the base coordinate 710, a second coordinate 720 may be identified. The second coordinate 720, in the present example, may be reflected formally as a base coordinate plus a modifier in the context of an overarching constant (base*% modifier*fx()). In this
example, the constant has been reflected as pixels, more specifically one pixel; scaling units other than a pixel can be utilized as can constants other than one. Second coordinate 720, in this instance, is rendered as a result of being located on the Y-axis at a 4-times percentage increase over the Y-axis location of base coordinate 710 in the context of a 1 pixel scaling unit. That is, coordinate 720 is located 4 pixels higher on the Y-axis than base coordinate 710.

[0103] Third coordinate 730 is depicted in a similar fashion wherein it is located at 4-times the pixel percentage on the X-axis as from second coordinate 720 and 4-times the pixel percentage of the X-axis and Y-axis relative to base coordinate 710. Base coordinate 710 in conjunction with second 720 and third coordinates 730 result in the rendering of a triangle 740 on the display.

[0104] A coordinate layout system is not meant to be limited to only a Cartesian grid but also encompasses, for example, polar coordinates and a three dimensional grid (i.e., x*y*z).

[0105] The final rendering of one object can be used as a base coordinate for a second object in a semantic coordinate layout system. For example, third coordinate 730 can be utilized as a first base coordinate 750 for a new object. That is, the upper right hand corner of a first object (triangle 740) serves as the bottom left corner coordinate 750 of a second object (square 760).

[0106] By further example, utilizing an exemplary semantic coordinate layout system, a first base coordinate might be identified as the upper right hand corner of an object (e.g., coordinate 730 of triangle 740). In some instances, however, the location of the upper right hand corner (730) of the object (740) will not be known as a layout engine 600, for example, may still be determining the locale of certain information to be rendered as determined by certain rules and limitations of the particular OS or device.

[0107] Once the layout engine 600 evaluates the layout of a particular device, the actual location of the upper right hand corner of the object may be ascertained. Once that location is determined (e.g., coordinate 730), semantic coordinates allow for the rendering of additional coordinates and/or the entire remainder of an object. This late binding of locations through the use of a semantic specification (e.g., the upper left of an objects Y is ten pixels from the lower right of object X) further allows for synchronous adjustment of layout.

[0108] Formulaic expression may also be used in a semantic coordinate layout system. For example, (Lower Y=Upper Right X+20% of Width of X+10 Display Units). In this example, none of the values are immediately calculable until the layout engine 600 renders object Y and a relationship between display unit and pixel (or some other base measurement) is determined by the layout engine 600.

[0109] The exemplary formulas provided herein are not meant to be limiting. Various other formulaic entries may be utilized in the rendering and layout of objects and information.

[0110] In some embodiments of the present invention, the rendering of objects or information in a scalable user interface that operates with neutrality towards a device or platform will often utilize a combination of vector graphics, a grid system and/or a semantic coordinate layout system. For example, a line is specified as being drawn from point \( x_1, y_1 \) to point \( x_2, y_2 \) with a specified line width and perhaps a specified arc. Points \( x_1, y_1 \) and \( x_2, y_2 \) may be determined as the result of utilizing a grid layout system. Individual objects may then be rendered in light of these coordinates using vector graphics. Additional objects may then be expressed as semantic coordinates considering certain coordinates of previously rendered objects.

[0111] Alternatively, some embodiments of the present invention may utilize bitmapping/rasterization in the context of a particular layout system (e.g., an object is rendered utilizing a combination of techniques individually and in conjunction with one another). For example, utilizing a semantic coordinate system, a base coordinate for a display button may be indicated as ten display units right from a previously rendered object. The actual button, however, may be a bitmap in a library and is rendered on the screen with its lower-left corner being at the base coordinate as determined by a semantic coordinate layout system.

[0112] The layout engine 600 of FIG. 6A may operate in conjunction with various rendering tools to result in scalable or intelligently placed graphic events. For example, the layout engine 600 may determine that an input request 610 to render a particular button or icon cannot be displayed as requested following a query to the rules engine 620. The logic engine 630, however, may instead determine what aspects of the particular icon need to be adjusted or scaled (e.g., adjusting the display unit or constant) whereby the icon is still rendered but on a smaller scale in accordance with various vector graphic or coordinate layout techniques.

[0113] It should further be noted, as is illustrated in FIG. 8A, that the rendering of graphic events or information can occur hierarchically. For example, display 800 may exhibit certain limitations as are recognized by a rules engine 620 (FIG. 6A). Limitations on display or other events can also be hierarchical and also stored in the rules engine 620.

[0114] An example of hierarchical limitations is shown whereby a sidebar 810 is comprised of various smaller icons 820-840. Sidebar 810 may impose its own independent limitations as they pertain to smaller icons 820-840, that is, smaller icons 820-840 cannot exceed the width and height of the sidebar 810 just as sidebar 810 may not exceed the limitations of display 800.

[0115] A similar situation exists with text boxes 850 and 870. Both text box 850 and text box 870 comprise smaller display elements 860 and 880-890, respectively. Display elements 860 and 880-890 must not exceed the limitations imposed by text boxes 850 and 870 just as those text boxes must not exceed the limitations of display 800.

[0116] Furthermore, limitations can exist between the smaller sub-elements of the display. For example, icons 820-840 may have a limitation wherein they cannot come within X pixels of one another due to color schemes that might begin to “blend” together and result in a deteriorated viewing experience.

[0117] Similarly, display elements 880 and 890 may be fixed as to a certain size that cannot be scaled any larger or smaller due to the amount of textual information contained therein where, if reduced further than its default font size, would render the amount of text illegible.

[0118] FIG. 8B illustrates information rendered in a device- and platform-neutral interface display wherein the information is rendered at different resolutions but maintains the appearance and quality as a result of the rendering methodologies disclosed in the present invention. The image is not rasterized (e.g., bitmapped) whereby distortion would occur.
Instead the image scales smoothly and efficiently and may be rendered on various devices having the aforementioned different resolutions. As a result, the three different devices with three different resolutions result in a near identical rendering of information allowing for a common and enjoyable user experience.

[0119] The above-described embodiments are exemplary. For example, the present interface also allows for building various applications (e.g., gaming applications) across various platforms and devices. One skilled in the art will recognize and appreciate various applications of the disclosed invention beyond those presently described here. This disclosure is not meant to be limiting beyond those limitations as expressly provided in the claims.

What is claimed is:

1. A method for semantically rendering information in a display environment, rendering a first object; identifying a point on the first object as a base coordinate for a second object; and rendering a second object relative the base coordinate.

2. The method of claim 1, wherein the second object is a polygon.

3. The method of claim 1, wherein the second object is a raster image.

4. The method of claim 1, wherein the second object comprises a base coordinate for the rendering of a third object.

* * * * *