

41

- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user at FIG. 5.
- 3: Whether the application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

The header records are written at step 153.

During the build process, as new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, at 154, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the external database as described at 156, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record at 155 based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist. The font and color objects are serialized as is discussed in greater detail below (See 159 below).

The body of the external database is then written at 156. All Boolean values are written inside a four-dimensional loop at 157. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment.).

42

All integer values are written inside a four-dimensional loop at 158. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

5 Multimedia objects are written inside a two-dimensional loop at 159. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains 15 web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other 20 objects on a web page.

String records are written inside a four-dimensional loop at 160. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The 25 third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized. 30

Single and double floating-point, and long integer records are written inside a two-dimensional loop at 161. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

FIG. 25 describes the techniques used to create a customized and optimized run time engine by the build engine (31 of FIG. 4). A versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted at 162. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation at 163.

All external image, video and audio files are resolved at 164. The external references can be copied to designated directories at 164, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries from Sun, Microsoft and Netscape are either installed on the local system (See FIG. 5) or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code at 165. Finally, the run time engine for the web site is created at 166. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file (See FIG. 27).

FIG. 26 shows the techniques used to create the HTML Shell File (HSF) (32 of FIG. 4).

The first step of the process at 167 is to determine whether the dynamic web page and object resizing is desired by testing the application setting, set by the user at FIG. 21a, or possibly

reset at FIG. 24. If the application was a web page, and thus requiring dynamic web page and object resizing, virtual screen resolution settings, calculated at FIG. 24 at 153, are placed in an appropriate HTML compliant string at 168. If the application is a banner or other customized application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values at 169.

An analysis is made for the background definition for the first internal web page at 170. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the browser. More specifically, if the application required dynamic web page and object resizing (See 167) then JavaScript and HTML compliant strings are generated at 171 so that, when interpreted by the browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated at 172 in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current browser type.
- 2: Load the SRS from either a JAR or CAB File, based on browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated at 173 that perform the following steps at the time the HSF is initialized by the browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the External Database created at FIG. 24.
- 3: Generate an HTML complaint string, dependent upon the type of browser, which causes the current browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

At 174, writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Websitename".html file is created.

FIG. 27 describes the processes for creating the CAB and JAR Files (33a of FIG. 4). The image objects, if any, which were defined on the first internal web page are analyzed at

175. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed at 176 to determine which JAVA classes have been compiled (See FIG. 25). These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created at 177 that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Websitename".class, customized run time engine file, and the "Websitename".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Websitename".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Websitename".bat and "Websitename".lib files are written at 178. The "Websitename".bat and "Websitename".lib files are then executed under DOS, creating compressed "Websitename".cab and "Websitename".lib files and compressed "Websitename" jar and "Websitename.lib" jar files at 179. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the run generation processes.

Run Time Process

The run time process is shown in FIG. 28 through FIG. 36.

FIG. 28 shows the web page size generation technology utilized by the run time engine. A web surfer points a browser at the HTML shell file (HSF) at 180. The browser begins to interpret the HTML and JavaScript code in the HSF that was created (See FIG. 26). The browser draws either the background color or background image pattern, as defined by the HTML complaint code (See FIG. 26) at 170. The browser then executes the HSF's JavaScript initialization code, which "sniffs" the current browser at 181 to determine its type, and then generates the appropriate HTML code for that particular browser to interpret. This code defines whether the executable files and database will be extracted from inside a compressed CAB file or a compressed JAR file and its location.

Based on the user application (defined at FIG. 21a, or possibly reset at FIG. 24), the HSF at 182 will then execute an appropriate JavaScript function (as created in FIG. 26 at 167). If the application required dynamic resizing of the web page's dimensions, JavaScript code is called which generates HTML code using the JavaScript "document.write" function, which causes the SRS applet to be immediately executed by the browser at 183. The JavaScript code then goes into a timer loop, checking on when the SRS applet is alive before initiating any communication.

After detecting that the SRS has been initialized, a JavaScript function calls the appropriate SRS applet methods at 185, which return the width and height, in pixels, of the current browser window. JavaScript Code is then called which converts the screen resolution independent window width and height values into absolute pixel values. A JavaScript function is then called which use the JavaScript "document.write" function to generate HTML code that define the run time engine specifications, etc. (see FIG. 26) and cause the browser to immediately execute the run time engine. If the application had not required dynamic resizing of the web

45

page's dimensions, then a JavaScript function is called which generates HTML code using the JavaScript "document.write" function that defines the fixed dimensions for the web page size and cause the browser to immediately execute the run time engine at 184.

FIG. 29 shows the techniques employed by the run time engine to read the external database and to generate the necessary web page objects (35 of FIG. 4). The run time engine reads a "PARAM" value at 186, from HTML Code that was generated above (see FIG. 26), which points to the "Website-
name".dta external database that is compressed into the JAR or CAB File (that was loaded and accessed in FIG. 28). The run time engine then initiates the read operation. In one implementation, the read technique is always non-privileged. If permitted by the current browser as a non-privileged operation, the "Website-
name".dta file will be extracted and read from the CAB/JAR file residing in temporary local storage. If not, the run time engine A will read the "Website-
name".dta file directly from the server. The header record is read at 187. Any objects, such as fonts and colors, are cast into their original form. The high watermark values, as they are encountered in the header and in the body of the database, are immediately used for setting the limits for the subsequent multi-level read loops for reading in the style record and the web page(s) and object(s) definitions. The virtual screen resolution values are read for the subsequent dynamic resizing of the web page objects.

The style record is read based on its high watermarks, and processed at 188. The definitions for all paragraph, text button, image or other styles are read and stored for subsequent initialization and processing of all paragraph, text button, image or other objects. The data representing the values for the first web page and all its objects is read at 189. The Boolean, integer, string and floating point fields for the first web page are initialized. The serialized multimedia objects for the first web page are read and cast into their final form. (See FIG. 24)

If external files, such as image, audio and video files, must be read as part of the first web page's generation, exception handling routines are executed at 190, as necessary, in the event of any processing errors. In one implementation, error recovery at this stage places the highest priority on a graceful operation cancellation, rather than a web page crash. In the worst case, a particular image, sound or video file may not be available to the web surfer. All other aspects of the web page will likely be available even in this error scenario.

Process step 191 is executed simultaneously with the generation of all the other web pages at 192 by means of multi-programming utilizing thread technology. Thus the first web page will be drawn and active for user viewing and user interaction long before the data for all the other web pages have been read, processed, and initialized. The data representing the values for the subsequent web pages and all their objects are read at 192. The Boolean, integer, string and floating point fields for these web pages are initialized. The serialized multimedia objects for these web pages are read and cast into their final form. (See FIG. 24)

FIG. 30 shows the scaling techniques employed by the run time engine for web page generation (36 of FIG. 4). The first step in the scaling process is to calculate the coordinates that define the origin for the placement of each object for a given web page. (This is usually the upper left corner of the object, defined in actual screen pixels.) A test is made at 193 to determine if the centering attribute is set for the object. If not, the left and top coordinates are converted from the virtual screen values to the local screen values, based on the local screen window resolution at 194. In one implementation,

46

multiplying the virtual coordinate by the local screen window resolution and dividing by the virtual screen resolution determine the conversion.

If the centering attribute is on, then a calculation for the object's width is performed. See processes 197, 198, and 199 below for a description of this calculation. Based on this calculated width, and based on the local screen window resolution, the left coordinate is calculated at 195. One algorithm that can be used is to subtract the screen width, as calculated in 197-199 below, from the local screen window resolution, and divide that result by 2. The top coordinate is calculated the same as in process 194 above.

Based on the object type, determined at 196, a different scaling technology is employed.

If the object is a text button object at 197, the text button object itself, including its background, is not scaled. The virtual width and the local screen width remain the same. However, if a 3D Frame effect is defined, it is scaled based on the following algorithm: if the text string's orientation is Left to Right, the inner width of the 3D Frame, and its placement relative to the text string, is calculated as the length of the text string, plus $\frac{1}{2}$ of an "n" space on each side, plus an additional offset appended to the right of the inner width to compensate for the italic font style, if defined for the font of that text string. The italic offset can be defined as the font size for the text string, divided by 10, plus 1. The inner height of the 3D Frame can be defined as the font height plus 2 pixels. The font height equals the font's leading plus its ascent plus its descent specifications. The inner height origin can equal the text string origin. The style of the 3D effect (i.e., either a 3D raised look or a 3D depressed look), plus the inner width and height, is sent to a 3D frame build method for the construction of the 3D frame. The width of the 3D frame in pixels can be calculated as the inner width divided by 10 plus 3.

If the text string's orientation is vertical, the inner width of the 3D Frame is an "m" space. The inner height of the 3D Frame can be calculated as the font height times the number of characters in the text string. Both the left and top placement of the 3D frame can be set to the left and top origin of the text string. The width of the 3D Frame can then be calculated as the inner height divided by 10, plus 3.

If an animation is assigned to the text string, the font size used for the initial calculation of the 3D frame is the same as that used to define the animation's initialization value. If the object is a paragraph at 198, and the scaling attribute is on, the maximum width for the paragraph can be defined by the attached paragraph style (or paragraph override) as a percentage of the screen width. This screen width percentage can be converted into an actual width in pixels, based on the local screen's window resolution. If the current screen resolution is the same as that used by the web designer, then the paragraph line end values (just read from the external database) are used without adjustment, bypassing the entire paragraph reformat process. If the current screen resolution is different than that of the virtual screen resolution, then a very compact method of reformat is called (relative to the build engine reformat methods at FIG. 6 and at FIG. 18), and the text for the paragraph is reformatted based on this width.

The run time engine's reformat technology begins by creating one paragraph line for the entire text string assigned to the paragraph text area. All the style overrides are renumbered sequentially with the style records or the non-marked text strings ignored. A simplified "Overflow" reformat method can be called, which chops up the single paragraph line first into paragraph line segments, where each word is defined as a line segment. Because of the word oriented style override architecture, the style overrides have a one-for-one corre-

47

spondence with the line segments. Each paragraph line break can be calculated by relying on the simplified word oriented style override technology described above. The paragraph line can be built inside a tight word-by-word loop, with a simple logic check for a style override or hard line break. The paragraph width is then derived as the width of the longest line of the reformatted paragraph, while the paragraph height is defined as the font height times the number of lines. If a 3D frame was defined for the paragraph, it can be scaled based on the following algorithm:

The inner width is defined as the same as that of a text string, but the width of the text string for the longest line is used. The same "n" space and italic offset calculations are used. The inner height is calculated as the font height times the number of lines plus 2 pixels.

If the object is a paragraph, and the scaling attribute is off, then the paragraph is treated the same as a text button object, with the only exceptions that there is no vertical orientation, and the height and width of the 3D frame, if defined, is calculated using the same algorithm as was used for the scaled paragraph above.

If the object is an image at 199, and the scaling attribute is on, the image width can be calculated as the virtual width times the local screen window width divided by the virtual screen width. The image height can be calculated as the virtual height times the local screen window height divided by the virtual screen height. If the image had been resized or rotated, then the virtual width and height of the image would differ from that of that of the original image. If a 3D frame is defined for the image, it can be scaled based on the following algorithm:

The inner width and the inner height of the 3D frame will coincide exactly with the outer edges of the image, after the image had been scaled. Adding the scaled image height to the scaled image width and dividing the result by 40 and adding 3 can calculate the width in pixels of the 3D frame.

If an animation is assigned to the image, then the animation's initialization values for the image's width and height can be used to calculate and draw the initial 3D frame. The coordinates and sizes for the backgrounds for text button, image and paragraph objects can be calculated using the same algorithms as was employed for the calculation and placement of the inner width and inner height for the 3d frame for each object.

FIG. 31 through FIG. 35 shows the multilevel web page and object thread technology employed by the run time engine. The description includes all the animation technologies, transformation technologies, time line technologies and drawing technologies that support this multilevel architecture.

FIG. 31 describes the initial processes for the invention's multilevel web page and object thread technology employed by the run time engine (37 of FIG. 4). Upon the completion of the processing of all the data definitions for the first internal web page (FIG. 30), the main web page thread is created and executed. This causes the run method for the main run time engine class to be executed simultaneously with the reading, processing, and scaling of the data for the subsequent web pages (See FIG. 29). In addition, the reading of any image files defined for the first web page is also performed simultaneously, under the control of an image observer (See FIG. 12). The main run method enters a web page counter loop at 200, the loop being defined from the first internal web page to the high watermark that was set to the number of existing internal web pages for the web site.

A check is made at 201 to see if the current web page exists. If the web page does not exist, and the current web page

48

number is less than that of the high watermark, then the web page counter is incremented by one and the web page counter loop is reentered. If the current web page number equals the high watermark at 202, then the web page counter is reinitialized to the first web page, so that the web page loop may repeat itself, from the first internal web page, depending upon the delay setting for the last web page.

A test is then made on all objects defined for this web page at 203, utilizing a loop whose range is defined by the number of objects per web page high watermarks. More specifically, within this universe of possible objects, if the object exists, and it is defined by a time line in which there is a delayed entrance, then a boolean flag is set for those objects that causes the draw system to suppress drawing these objects during the web page transition as defined below.

A test is then made to determine if the web page has a transition animation defined at 204. If not, the draw system is called for the first time. The draw system for a given web page utilizes a loop whose range is defined by the number of objects per web page high watermarks. The draw system can also employ technology so that the draw process generates a screen image in one or more off-screen buffers, only drawing to the screen when the screen image, or the clipping area for the screen, has been fully generated. This greatly reduces, if not totally eliminates, any screen flicker, and creates visually smooth animation effects.

The first draw function is to draw the web page background into the primary off-screen buffer. The web page background color is drawn, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. A test is then made to determine if the web page has a background image pattern, as defined initially at FIG. 21a, or modified for that particular web page at FIG. 20. If it does, and the image observer reports that the image is ready to be drawn, a background image draw loop is executed, defined by the height and width of the background image, and the screen resolution of the current browser window. In the unlikely event that the background image pattern is not yet available, there is a delay until the image observer reports the completion of the image processing operation. The tiled background image pattern is also drawn into the primary off-screen buffer, completely over-drawing the background color. The backgrounds for all non-suppressed (See 203) parent web page text button and paragraph objects are then drawn into the primary off-screen buffer, unless a background transparency flag has been set (See FIG. 7).

The text strings for non-suppressed parent web page text button and paragraph objects are then drawn into the primary off-screen buffer. These text strings are drawn based on their font name, style, size, effect(s), and color. If a paragraph line string, the string may have multiple string segments, each with their own font name, style, etc. If the text button object has its vertical attribute set to true, then the draw system executes a loop defined by the number of characters defined in the text button object. The top and left origin coordinates were set in the usual way (See FIG. 30), but the top coordinate is adjusted by the font height for each iteration of this draw loop. The intelligent 3D Frame, if defined, is then drawn into the primary off-screen buffer for the paragraph and text button objects (See FIG. 30). The primary image objects for the web page are then processed by the draw system. If the image observer reports that the image is ready to be drawn, it is drawn into the primary off-screen buffer, based on the coordinates and size as defined in FIG. 30. If not ready, there is a delay until the image observer reports the completion of the image processing operation. The Intelligent 3D frame, if

49

defined, is then drawn into the primary off-screen buffer for the image objects (See FIG. 30).

The draw system is responsive to two other technologies at this stage. The first is user interaction based on the location of the mouse cursor and any user initiated mouse event. This subject will be described in greater detail below in association with FIG. 36. The second is object animation for non-delayed web page objects. This subject will be described in greater detail below in association with FIG. 33.

If the web page transition test at 204 was true, then the run time engine's main run method executes the web page transition animation technology at 205.

FIG. 32 describes the web page transition animation technology. First a lock is placed on this method at 212, as a safety precaution to prevent any interference from other threads during the animation. A test is then made on whether the transition animation setting (See FIG. 20) for the web page is random at 213. If so, a random transition number is generated at 214. The web page thread then begins a particular animation loop at 215, depending upon the random number that was generated at 214 or by the transition animation that was set previously (at FIG. 20). In one implementation, 13 different transition animations plus random are supported including. They are: Fade In, Zoom In, Zoom Out, Zoom to Upper Left, Zoom to Lower Right, Rotate to the Left, Rotate to the Right, Rotate Bottom to Top, Rotate Top to Bottom, Slide to the Left, Slide to the Right, Slide Bottom to Top, and Slide Top to Bottom.

For all web page transition animations, the X and Y animation increment values are calculated by dividing the current browser's screen width and height by the user defined animation resolution at 215. In all animation and draw loops, the number of loops can equal the number of animation frames as set at FIG. 20. The timer delay for all animations, in milliseconds, can be calculated by dividing the number of frames per second (See FIG. 20) into 1,000.

For a description of "Fade In" Technology see FIG. 33. A "Zoom In" algorithm sets the initial scaled width and height for the current web page image to zero and the prior web page image to its full size. In each animation and draw loop the previous web page's final image state is drawn into a secondary off-screen buffer at 216. (If this is the first occurrence of the first web page, then the secondary off-screen buffer is set to the background of the first web page.) The upper left hand corner (origin) of the current web page can be calculated based on the following formula: browser screen width minus the scaled width divided by two.

The scaled image of the current web page is then drawn into the secondary off-screen buffer at the calculated origin, using the current scaled width and height for the web page image. This merged image of the prior and scaled version of the current web page is then drawn to the screen.

A timer delay then occurs as defined at 215, after which the X and Y animation increment values are added to the scaled width and height for the current web page image. The animation loop is then repeated to its conclusion at 218.

The other eleven web page transition animations follow a similar methodology, but have quite different calculations, which are based on the following variables:

- 1: Order of drawing of the prior and current web pages.
- 2: Initialization values for the X and Y origin coordinates for the current and prior web pages.
- 3: The initial values for the scaled width and height for the current and prior web pages.
- 4: Whether X and Y origin coordinates for the current and prior web pages increment, decrement, or remain the same.

50

- 5: Whether the values for the scaled width and height for the current and prior web pages increment, decrement, or remain the same.

For the "Zoom Out" animation, the current page is drawn first and always drawn at 100%. The prior web page is initialized also at 100%, but its X and Y origin coordinates are incremented and its scaled width and height values are decremented, by the appropriate values, for each animation iteration.

For the "Zoom to Upper Left", "Zoom to Lower Right", "Rotate to the Left", "Rotate to the Right", "Rotate Bottom to Top" and "Rotate Top to Bottom" animations, a common data initialization and data increment strategy is implemented.

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: The scaled width and height variables for page image one is set to 100% of the browser window's resolution.
- 4: The scaled width and height variables for page image two is set to zero.
- 5: For each loop iteration, the scaled width and height variables for page image one are decremented by the X and Y animation increment values defined at 215.
- 6: For each loop iteration, the scaled width and height variables for page image two are incremented by the X and Y Animation increment values defined at 215.

For the "Zoom to Upper Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. (upper left corner of the browser window) Its scaled width and height values are always set to the current values for scaled width and height variables for page image one. The X and Y origin coordinates for the current web page can be calculated by subtracting the current values of image two's scaled width and height variables from the initial values of the X and Y variables for page image two. The scaled width and height values for the current web page can be set to the current values for the scaled width and height variables for page image two.

For the "Zoom to Lower Right" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width and height values are always set to the current values for scaled width and height variables for page image two. The X and Y origin coordinates for the prior web page are set to current values of image two's scaled width and height variables. The scaled width and height values for the prior web page are set to the current values for the scaled width and height variables for page image one.

For the "Rotate to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to current value of image one's scaled width variable. Its scaled height value is always set to the bottom of the browser's window. The X origin coordinate for the current web page can be calculated by subtracting the current value for image two's scaled width variable from the initial value for image two's X origin coordinate. The Y origin coordinate for the current web page is always set to zero. Its scaled width value is set to current value of image two's scaled width variable. Its scaled height value is always set to the bottom of the browser's window.

For the "Rotate Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image one's scaled height variable.

The current web page's X origin coordinate is always set to zero. The Y origin coordinate is calculated by subtracting the current value of image two's scaled height variable from the

51

initial value for image two's Y origin coordinate. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image two's scaled height variable.

For the "Rotate Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. Its scaled width value is set to the width of the browser window. Its scaled height value is set to current value of image two's scaled height variable.

The prior web page's X origin coordinate is always set to zero. The Y origin coordinate is set to the current value of image two's scaled height. Its scaled width value is always set to the right edge of the browser's window. Its scaled height value is set to current value of image one's scaled height variable.

For the "Slide to the Left", "Slide to the Right", "Slide Bottom to Top" and "Slide Top to Bottom" transition animations, a common data initialization and data increment strategy is implemented. The strategy includes:

- 1: The X and Y variables for page image one is set to zero.
- 2: The X and Y variables for page image two is set to the right and bottom edges of the browser window.
- 3: For each loop iteration, the X and Y variables for page image one are incremented by the X and Y animation increment values defined at 215.
- 4: For each loop iteration, the X and Y variables for page image two are decremented by the X and Y animation increment values defined at 215.
- 5: The scaled width and height values always remain at 100% of the browser windows width and height.

For the "Slide to the Left" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's X origin coordinate is set to the current value of page image two's X variable. Its Y origin coordinate is always set to zero.

For the "Slide to the Right" Animation, the current web page is drawn first, with its X and Y origin coordinates always set to zero. The prior web page's X origin coordinate is set to the current value of page image one's X variable. Its Y origin coordinate is always set to zero.

For the "Slide Bottom to Top" animation, the prior web page is drawn first; with its X and Y origin coordinates always set to zero. The current web page's Y origin coordinate is set to the current value of page image two's Y variable. Its X origin coordinate is always set to zero.

For the "Slide Top to Bottom" animation, the current web page is drawn first; with its X and Y origin coordinates always set to zero. The prior web page's Y origin coordinate is set to the current value of page image one's Y variable. Its X origin coordinate is always set to zero.

After the last animation cycle is completed for any of the transition animations at 218, the animation process is unlocked, and process step 206 shown in FIG. 31 is then executed.

Returning to FIG. 31, the main web page thread's run method then executes a text button and image object time line, transformation and animation loop at 206. This range loop is defined from the first object on the given web page to the high watermark for the number of those objects on a web page for this web site. A test is made on each object on whether an animation, transformation and/or time line has been assigned at 208.

If so, an "instance" of the time line class for that particular object type is created at 209. An "instance" of a class is a fundamental aspect of object oriented programming (OOP). Each time, the line class is implemented with a "runnable" interface, so that they can be executed as independent threads.

52

Communication of data, between the "instance" of a class and the main run engine class can be accomplished in OOP using several different techniques. In one implementation, this construction, passed as an argument, is used to permit different objects to address each other's variables and databases. A thread is then created, utilizing a two-dimensional object internal database architecture (web page number by internal object number). This methodology is convenient for permitting all object time lines for a given web page to be managed and synchronized. The object's thread is then "started".

The result of this process at 209 is that an independent thread has been created for each appropriate object on a given web page, all executing simultaneously with each other and with the main run time engine web page thread, subject to the definitions of their independent time lines at 210. See FIG. 33 for a description of the time line technology. When the main web page thread has finished the text and image loop at 207, the draw system is activated; the run time engine can now respond to user interactions, and the main web page thread transitions into a "Join" loop at 211. See FIG. 35 for a description of this process.

FIG. 33 shows the time line technology used by the run time engine. The techniques and algorithms employed to create this technology permit each web page object to have an independent yet synchronized existence with each other, with the main web page thread, and with child objects that each main or parent object may spawn. Furthermore, each object and each of their child objects are capable of performing multiple animations and transformations, either serially or simultaneously. Database initialization is first accomplished for each object thread. This assures that the object thread's database is set to the correct initial values as required for that particular object, and that the references to the main web page thread's database are established.

A test is then made to determine if the object has a time line definition assigned to it at 219.

If not, a test is made at 220 on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and animation and transformations occurring in a serial manner.

If the test shows that the object has an animation defined, but no transformation, then certain two-dimensional status variables are set, and an "instance" of the "animation class" for that particular object type is created at 229. Each "animation class" is also implemented with a "runnable" interface. An object animation thread is then created, utilizing the two-dimensional object internal database architecture (See FIG. 8). This object animation thread is then "started". Communication between the object animation thread, the parent time line thread, and its parent, and the main web page thread, are accomplished as discussed in process 209. The object time line thread then executes a "Join" method. This puts the object time line thread in a "wait state". When the thread it is waiting for is completed, this child thread "joins" the parent object time line thread, and the object time line thread then continues its process. Other forms of synchronization between two independent threads could have been implemented as is known in the art.

The techniques employed at 229 to implement object animation vary by object type. In one implementation, for text button object animations, 26 different animations are supported including: Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S,

53

Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW. In one implementation, for image object animations, 29 different animations are supported including: Fade In, Fade out, Rotate, Zoom In, Zoom Out, Grow NW, Grow NE, Grow SE, Grow SW, Shrink SE, Shrink SW, Shrink NW, Shrink NE, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W, Enter NW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

As discussed above with regard to FIG. 17, each animation type has a defined speed, resolution, and number of animation cycles. These settings are stored in the main web page class, and are passed to the particular animation thread through a two-dimensional object internal database architecture as discussed in process step 209 above during the animation thread's initialization process. The animation thread then executes, in its run method, a main animation loop that has the number of iterations set to the end number of animation cycles, as assigned to that particular text button object.

Text button animations are currently implemented in three logical groups. Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW", "Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter S", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE", "Exit E", "Exit SE", "Exit S", "Exit SW", "Exit W" and "Exit NW".

For Group One text button animations, the animation font size is initialized at a very small value, and in one implementation is set at 4 Points. The animation point size increment can be derived by dividing the resolution (number of animation frames) into the font size for that text button object. The run method then executes a secondary animation loop, which will terminate when the animation font size equals the text button object point size. For each secondary animation loop, the length of the current animated text string is calculated, a new font object is created for the current animation point size, and the font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the current animated font. The animated text button object's height is calculated to be current animated font height times the number of characters in the text string. If the text had a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the current animated font. The animated text button object's height can be calculated to be the font height of the current animated font. The calculations for X and Y coordinates for the animated text button object depend upon which animation was defined within the Group One-text button animations. The X and Y animation increments can be calculated utilizing the height and width, in pixels, of the text button object scaled to the current browser's window, utilizing the text button animation resolution, and considering whether the animating text button object is being centered during the animation ("Zoom Out") or not. These calculations are similar to those for the web page transition animations discussed with regard to FIG. 32.

The draw system is then called. Based on the values of the two-dimensional status variables that had been set initially, the draw system executes the appropriate animation draw routine utilizing, through the data communication techniques already discussed, the current animation font point size, and the current animation X and Y coordinates. If a text background is to be drawn, the same algorithm as defined in FIG. 31 is used. If a 3D Frame is assigned, the current animated string width and height are passed to the appropriate 3D frame

54

generation method, and the frame is drawn with the same algorithm as defined in FIG. 31, but utilizing the current animation X and Y coordinates. The text button object's orientation is also handled by the draw system with the same algorithms as defined in FIG. 31.

The text button animation thread then executes a timer delay, whose value had been defined in FIG. 17. When the timer reactivates the text button animation thread after the appropriate delay, an animation cycle completion test is made to see if the text button object's point size minus the animation point size is less than the animation point size increment. This type of testing methodology permits the invention to utilize integer values, as opposed to floating point values, for the text button animation. This improves the execution of the animation considerably.

If the above test is true, the animation point size is set equal to the object point size and a final call is made to the draw system for that animation cycle. A test is then made to see if the current animation cycle equals the total number of animation cycles as defined in FIG. 17. If not, a new animation cycle is initiated, with the animation values reinitialized. If this was the last animation cycle the text button animation thread calls its "stop" method, which sets the required status variables as appropriate, then terminates itself. This causes the parent text button time line thread to be reactivated through the "join" mechanism.

If the results of animation cycle completion test are false, the current animation point size is increased by the animation point size increment. A new font object is created for the now current animation point size, and new font metrics for that new font are created. If the text button object has a vertical orientation, the animated text button object's width is calculated to be the width of an "m" space, in the now current animated font. The animated text button object's height is calculated to be the now current animated font height times the number of characters in the text string. If the text has a horizontal orientation, the animated text button object's width is calculated to be the width of the text string in the now current animated font. The animated text button object's height is calculated to be the font height of the now current animated font. The calculations for the new X and Y coordinates for the animated text button object are then completed, as appropriate, and the draw system is called again.

The algorithms for Group Two text button animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation point size increments and the animation X and Y coordinate increments are added or subtracted from the then current animation point size and the then current X and Y coordinates for the animating text button object.

For Group Three text button animations, the distance that the text button animation will move is calculated, in pixels, from its initial location to its final location in the current browser window. The X and Y animation increments are calculated by dividing that distance by the resolution of the text button animation. All the other algorithms for Group Three text button animations are generally a subset of those for Group One, and similar to the web page slide transition animations defined with reference to FIG. 31.

Referring again to FIG. 33, image animations at process step 229 can currently be grouped into five logical classes. As with text button animations, Group One includes "Zoom In", "Grow NW", "Grow NE", "Grow SE", and "Grow SW". Group Two includes "Zoom Out", "Shrink SE", "Shrink SW", "Shrink NW", and "Shrink NE". Group Three includes "Enter N", "Enter NE", "Enter E", "Enter SE", "Enter S", "Enter SW", "Enter W", "Enter NW", "Exit N", "Exit NE",

55

“Exit E”, “Exit SE”, “Exit S”, “Exit SW”, “Exit W” and “Exit NW”. In addition, image animations have a Group Four, which includes “Fade In” and “Fade Out”. Group 5 image animations include the “Rotate” Animation.

For Group One image animations, the animation width and height increments are calculated by dividing the image object’s width and height by the resolution (number of animation frames) as set in FIG. 17. The initial animation width and animation height values are set to a very small number, currently equal to the animation width and height increment values just calculated. The calculations for X and Y coordinates for the animated image object depends upon which animation was defined within the Group One text button animations. The X and Y animation increments are calculated utilizing the height and width, in pixels, of the image object scaled to the current browser’s window, utilizing the image animation resolution, and considering whether the animating image object is being centered during the animation (“Zoom Out”) or not. These calculations are similar to those for the web page Transition Animations discussed above with regard to FIG. 32.

The run method then executes a secondary animation loop, which will terminate when the animation width equals the image object’s width. The algorithms employed by the invention to change the animating object’s height, width, X coordinate, and Y coordinate are very similar to those employed for Group One text button animations, and will not be repeated here. The techniques to utilize the draw system for drawing the image animation, the time delay technique, and the post draw logic tests and actions are also very similar.

The algorithms for Group Two image animations are very similar to those of Group One. The differences are just in what are the initial animation values, and whether the animation width and height increments and the animation X and Y coordinate increments are added or subtracted from the then current animation width and height and the then current X and Y coordinates for the animating image object.

For Group Three image animations, the algorithms are identical to those of Group Three text button animations. For Group Four image animations, the “alpha” value of a given image object is utilized in order to implement “Fade In” and “fade Out” animations. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha animation increment variable can be calculated by dividing the resolution of the animation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. For a “Fade In” animation the value of an alpha animation variable is set to zero. The run method then executes a secondary animation loop, which will not terminate until 255 minus the then current value of the alpha animation variable is less than the value alpha animation increment variable. A “Fade In” image filter can be created for each iteration of the animation loop, using the current setting of the alpha animation variable. An image producer can also be created with pointers to the last image bitmap produced for the image object in the last animation loop and to the image filter that has just been created. The image producer, under the control of a media tracker then creates a new image bitmap. The animation thread then “waits” for the completion of this image-processing event using the media tracker. Upon completion, the draw system is called which draws the then current state of the image object. The image animation thread goes into a timer delay of some preset value (in one implementation 500 milliseconds), to permit a smooth visual animation effect. The value for the alpha animation increment is added to alpha animation variable and the loop is then

56

repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the “Join” mechanism.

The “Fade Out” animation employs very similar technology, except that:

- 1: the alpha animation variable is set to zero,
- 2: the value for the alpha animation increment is subtracted, and
- 3: the loop termination test is when the value for the alpha animation variable is less than the value for the alpha animation increment.

For the Group Five image rotate animation, a different bitmap for the image object is created for each animation frame through the use of a progression of standard geometrical transformations on the original image bitmap. A secondary animation loop is then executed as defined by the number of animation frames. In each loop iteration, an image object is created from an appropriate image bitmap selected from among the set just created, the necessary two-dimensional variables are set to communicate with the draw system, and the draw system is then called. The image animation thread then executes a timer delay method based on the delay setting as defined above with reference to FIG. 17. When the timer reactivates the image animation thread after the appropriate delay, the next iteration of the secondary animation loop is repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image animation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 shown in FIG. 33, if the object had a transformation, but not an animation, then certain two-dimensional status variables are set, and an “instance” of the “transformation class” for that particular object type is created at 228. Each “transformation class” is also implemented with a “runnable” interface. An object transformation thread is then created, utilizing the invention’s two-dimensional object internal database architecture. This object transformation thread is then “started”. The inter-thread communication technology and the “join” technology employed for object transformations is the same as for object animations.

If the transformation is being applied to a text button object at 228, then a timer delay method is executed based on the delay setting as described in association with FIG. 18. When the timer reactivates the text button transformation thread after the appropriate delay, the appropriate two-dimensional status variables are set to inform the draw system which state of the current text button object to draw. The draw system is called and, based on the settings for the above mentioned two-dimensional status variables, either the “normal”, mouse over”, mouse down” or “pop-up” states of the text button object’s background, if any, the text button object’s string, and the 3D frame, if any, are drawn. If additional transformations are defined (FIG. 18), the above process is repeated, based on the timer delay and object states defined for the subsequent transformations. When the last transformation is completed, the “stop” method is called, which sets the required status variables as appropriate. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If the transformation is being applied to an image object at 228, then a timer delay method is executed based on the delay setting (as defined in FIG. 18). When the timer reactivates the image transformation thread after the appropriate delay, image transformation technology is executed. In one imple-

57

mentation, the image transformation technology utilizes the “alpha” value of a given image object state in order to fade in and fade out images. The alpha value can range from 0 to 255, depending upon the image strength desired. The value for an alpha transformation increment variable is calculated by dividing the resolution of the transformation into 255, after making the necessary adjustments to keep the data in integer form, without losing resolution due to integer rounding errors. The value of an alpha transformation variable is set to zero. Depending upon the settings as defined in FIG. 18, the bitmap for one image object state is initialized to an alpha value of zero, while another is initialized to an alpha value of 255. The appropriate two-dimensional status variables are set for communication with the draw system.

A transformation loop is then executed, until 255 minus the then current value of the alpha transformation variable is less than the value alpha transformation increment variable. This methodology again keeps all calculations in the form of integers, as opposed to floating point, thus speeding up the transformation process.

Two “Fade In” image filters are created for each iteration of the transformation loop. The first uses an alpha value calculated at the current setting of the alpha transformation variable. The second uses an alpha value calculated at 255 minus the current setting of the alpha transformation variable. Two image producers are also created with pointers to the last image bitmap produced for each image object state in the last transformation loop and to the two image filters that had just been created. The two image producers under the control of two media trackers then create two new image bitmaps. The transformation thread then “waits” for the completion of these two image processing events using the media trackers. Upon completion, the draw system is called which draws the then current state of the two image object states, in the correct order, and in the correct location. The image transformation thread goes into a timer delay of some preset value (500 milliseconds in one implementation), to permit a smooth visual transformation effect. The loop is then repeated until the loop condition is met. Then the “stop” method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with an animation and transformation that would execute in a serial manner, then certain two-dimensional status variables are set, and an “instance” of the “transformation” class for that particular object type is created at 230. An object transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object transformation thread is then “started” and the parent object time line thread “waits” to be “joined”.

If a text button object, then a primary loop is executed, with the number of iterations set to the number of transformations. After the execution and return from a timer delay event, if any, an “instance” of the text button animation class is created, and then a text button animation thread is created and “started”. The parent text button transformation thread then waits to be “joined”. This causes the text button animation thread to be executed, in the manner described at 229. When the text button animation thread completes its execution, it calls its “stop” method, which sets the necessary status variables and then terminates itself. This causes the text button animation thread to “join” the parent text button animation thread, causing that thread to resume processing. The first text button transformation is then executed, in the manner described at 228. After the execution and return from another timer delay event, if any, another “instance” of the text button animation

58

class is created, and then another text button animation thread is created and “started”. The parent text button transformation thread again waits to be “joined”. This causes the text button animation thread to be executed again with the animation being executed, based on the definition set at FIG. 18, on a different text button object state. The loop is then repeated until the last text button transformation is completed. Then the text button transformation thread calls its “stop” method, certain status variables are set, and the text button transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If an image object, the mechanism of the image transformation thread spawns image animation threads, before each transformation, and is the same as that of a text button object. The actual image transformation process is identical to that described at 228. When completed the “stop” method is called, certain status variables are set, and the image transformation thread terminates itself. This causes the parent image time line thread to be reactivated through the “join” mechanism.

Returning to process step 220 in FIG. 33, if the object was defined with a simultaneous animation and transformation, then certain two-dimensional status variables are set, and an “instance” of the “super transformation class” for that particular object type is created at 231. In one implementation, the animation, transformation, and super transformation classes are integrated into one structure in order to reduce code size and increase execution speed. Each “super transformation class” is also implemented with a “runnable” interface. An object super transformation thread is then created, utilizing the two-dimensional object internal database architecture. This object super transformation thread is then “started”. The inter-thread communication technology and the “join” technology employed for object super transformations is the same as for object transformations.

If a text button object, a calculation is made in order to prorate the text button animation process across the defined text button transformation process. The calculation is driven by the number of text button animation frames, and prorates from that total the number of frames that should be assigned to each transformation state. This can be done by dividing the sum of all the transformation times by each individual transformation time, and multiplying that result by the number of frames, making necessary adjustments to prevent integer rounding error. After these calculations are completed, the text button animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, certain two-dimensional status variables are set prior to calling the draw system for the next animation frame, so that the correct text button object state is drawn, in the correct size and with the correct coordinates, by the draw system. When the super transformation process is completed the “stop” method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

If an image object, a calculation is made in order to prorate the image animation process across the defined image transformation process. The calculation is driven by the number of image transformation events that would occur (where each one can be set at approximately 500 milliseconds) over the entire animation event. A calculation is performed in order to calculate how many image transformation events should be assigned to each transformation state. This is done by dividing the sum of all the transformation times by each individual

59

transformation time, and multiplying that result by the total number of transformation events, making necessary adjustments to prevent integer rounding error. A calculation is then made to allocate the number of animation frames to each image transformation event. After these calculations are completed, the image animation is executed in a similar manner as was defined at 229. However, when the appropriate number of animation frames had been drawn, the image transformation technology is called to perform the next transformation event. The alpha transformation increment can be defined by dividing 255 by the number of transformation events assigned to that transformation. The draw system is then called. When the number of image transformation events assigned to a given image transformation is reached, then certain two-dimensional status variables are set prior to calling the draw system for the next animation frame, so that the correct image states, in the correct size and with the correct coordinates, are utilized by the draw system. This entire animation/transformation process will be repeated by the number of image animation cycles. When the super transformation process is completed the “stop” method is called, certain status variables are set, and the text button super transformation thread terminates itself. This causes the parent text button time line thread to be reactivated through the “join” mechanism.

Returning to process step 219 in FIG. 33, if the object had a time line, then a test is made at 221 on whether an appearance delay had been defined in FIG. 19. If so, a timer event is set at 222.

When the timer reactivates the object time line thread after the appropriate delay, a test is made on whether an entry animation/transformation has been defined for this object time line at 224, as described FIG. 19. If so, based which animation/transformation process was defined, it is created and executed at 228, 229, 230, or 231. In one implementation, 13 entry animations are supported for both text button and image objects, and an additional “Fade In” entry animation is supported for image objects. The 13 common entry animations supported include Zoom In, Grow NW, Grow NE, Grow SE, Grow SW, Enter N, Enter NE, Enter E, Enter SE, Enter S, Enter SW, Enter W and Enter NW.

If no entry animation/transformation is defined, or when the entry animation/transformation has “joined” the object time line thread, a test is made to determine if any child time lines have been defined at 225, as described in FIG. 19, for this parent object time line. If so, an “instance” of the “child time line class” for that particular object type is created at 226. Each “child time line class” is also implemented with a “runnable” interface. An object child time line thread is then created, utilizing the two-dimensional object internal database architecture. This object child time line thread is then “started”. The inter-thread communication technology and the “join” technology employed for object child time lines is the same as for object time lines. Either a text button child time line thread or an image child time line thread, or both, can be spawned at this time.

Simultaneous with the execution of any spawned text button child time line threads, the parent object thread then executes the defined main animation and or transformation. As with non-time line object threads, a test is made on certain two-dimensional object definition variables in order to determine which of the following four states have been defined for the object at 227: animation without a transformation; transformation without animation; animation, with the transformation occurring simultaneously with the animation; and, animation and transformations occurring in a serial manner.

Based on the results of this test, an appropriate “instance” of an appropriate animation, transformation, or super trans-

60

formation class is created, and an appropriate animation, transformation, or super transformation thread is created and “started”. This results in the execution of process steps 228, 229, 230, or 231, as defined above.

The parent object time line thread then executes a “join” method. This again puts the object time line thread in a “wait state”. When the thread it is waiting for is completed, the child thread “joins” the parent object time line thread, and the object time line thread then continues its process.

The object time line thread then checks to see if there is a departure delay defined at 232. If so, it sets a timer event at 233. When the timer reactivates the object time line thread after the appropriate delay, a test is made at 234 on whether an exit animation/transformation has been defined for this object time line, as described in FIG. 19. If so, it is created at 235, and performed as discussed with reference to processes 228, 229, 230, or 231. In one implementation, 13 exit animations supported for both text button and image objects, and an additional “Fade Out” exit animation is supported for image objects. The 13 common exit animations include: Zoom Out, Shrink NW, Shrink NE, Shrink SE, Shrink SW, Exit N, Exit NE, Exit E, Exit SE, Exit S, Exit SW, Exit W and Exit NW.

If no exit animation/transformation is defined, or when the exit animation/transformation has “joined” the object time line thread, the parent object time line thread then executes a “join” method if it had spawned any child time lines. This again puts the object time line thread in a “wait state”. Finally, when then the child time line threads, if any, “join” the parent object time line, the “stop” method for the parent time line is called. Certain status variables are set, and the parent object time line thread terminates itself. This causes the main web page time line that had been in a “join” loop at 211 of FIG. 31, since the invocation of the object time lines, to be “joined” by this particular object time line thread.

FIG. 34 shows the technology employed by the run time engine for implementing child time lines for text button and image objects. Child text button object time lines and child image object time lines are subsets of their parent object time lines. First a test is made at 237 on whether an appearance delay had been defined (See FIG. 19). If so, a timer event is set at 238. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an entry animation has been defined for this child object time line at 239 (as described FIG. 19). If so, it is created and executed at 240 in a manner identical to that described at process step 229 in FIG. 33. The same 13 entry animations supported for parent object time lines are also supported for both child text button and image objects, and the additional “Fade In” entry animation is supported for child image objects. The “join” mechanism described in FIG. 33 is employed in an identical manner at 240 to synchronize the child time line thread with its entry animation thread.

After being “joined” and reactivated, the child object time line performs a test at 241 on whether an exit delay had been defined (See FIG. 19). If so, a timer event is set at 242. When the timer reactivates the child object time line thread after the appropriate delay, a test is made on whether an exit animation has been defined for this child object time line at 243, as described in association with FIG. 19. If so, it is created and executed at 244 in a manner identical to that described above at process step 229 in FIG. 33. The same 13 exit animations supported for parent object time lines are also supported for both child text button and image objects, and the additional “Fade Out” exit animation is supported for child image objects. The “join” mechanism described above in association with FIG. 33 is employed in an identical manner at 245 to synchronize the child time line thread with its parent object

61

time line thread. As discussed at process step 236 in FIG. 35, the parent object time lines “wait” until all their child time lines have terminated, before they in turn terminate and “join” the main web page time line at FIG. 35.

FIG. 35 describes technology employed by the run time engine for the web page and object thread loop. As noted in FIG. 31 at process step 211, after all the text button and image time line threads for the current web page had been launched, the main web page thread executed a “join” loop, waiting for the completion of all the parent object time line threads. Because each parent object time line thread waited for their child object time line threads to be “joined”, as well as any other spawned animation threads, transformation threads, and/or super transformation threads, the effect of this “join” loop at 246 is that the web page thread will not resume processing until all parent time line threads have completed and that of all of their spawned threads.

Upon resuming its processing after the “join” process at 246 has been completed, the main web page thread checks at 247 to see whether the current web page has an automatic termination, based on a timer delay, or whether the web page will wait for a user interaction before terminating.

If the web page has a time delay based termination setting, then a timer method is called at 249, and the web page goes to “sleep” awaiting the completion of the timer event.

When the timer event occurs, the web page thread resumes processing by incrementing the web page counter by one, and the entire web page process, which began at process step 200 in FIG. 31, is repeated. If the current web page termination setting was to set to wait until user interaction, then Web page thread is placed in a “pause” state, and the run time engine waits to respond to any mouse, keyboard or other user initiated event.

FIG. 36 describes the technology employed by the run time engine for responding to user interactions. As mentioned in association with process step 204 of FIG. 31, as soon as the draw system has been activated, the run time engine will respond to any user interactions that have been defined (See FIG. 16). This is also true during any object time line events, as with respect to process step 207 of FIG. 31. The run time engine currently responds to “mouse over” and “mouse down” events for text button, image, and paragraph objects. For form objects, the run time engine will also respond to keyboard events. As the full-featured programming languages supported by browsers evolve, the run time engine may be configured to respond to other user interactions, including but not limited to single and double clicks from both the left and right mouse button, voice commands, eye focusing technologies, touch screen technologies, and push technologies originating from a server.

The run time engine invokes a “dynamic mouse to object recognition” technology at 251 in order to be responsive to the following elements:

- 1: The location of objects will vary based on the viewer’s screen resolution and browser window size as discussed above with regard to FIG. 27.
- 2: Objects may move or resize themselves based on time lines and animations.
- 3: Objects may have different sizes based on the state they are being displayed in based on time lines and transformations.
- 4: More than one object can occupy the same screen location, and which objects occupy that location may change over time based on time lines, animations, and transformations.

The run time engine maintains, in its internal database, the object’s current X and Y origin coordinates, and the object’s

62

current width and height, in pixels, based on the current viewer’s screen and browser window size. This can be accomplished by first converting all coordinates and sizes to the current viewer environment with the scaling technology as discussed above with regard to FIG. 27. Every time line, animation, and transformation thread updates, in real time, the run time engine’s internal database positional and size variables of the objects they define, utilizing the data communication techniques described above with reference to FIG. 33.

The run time engine employs mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods to constantly monitor the state of the mouse at all times. The onClick method (to detect a single click) and a specialized method to detect a double click event are also employed. The onKeyDown method, with processing the returned scan code, permits the run time engine to process all keyboard events. The mouseEnter, mouseMove, mouseDown, mouseDrag, mouseUp, and mouseExit methods return to the run time engine the exact X and Y coordinates of the mouse cursor at the instant that particular mouse event occurred. Thus for each supported mouse based user interaction technique supported by the run time engine, a two-dimensional loop exists (web page number by object number) in which the current bounding rectangle for every object on a given web page is being compared to the current mouse cursor location at all times. The bounding rectangle is simply the current X and Y origin coordinates of an object, extended by its current width and height. In this way, the run time engine is informed if the current mouse cursor location falls within one or more bounding rectangles. Parenthetically, the run time engine also knows when the current mouse cursor falls outside the bounding rectangle of a given object.

Based on the type of mouse user interaction at 252, the run time engine employs different techniques and executes different methods. If the viewer moves the mouse at 253, the mouseMove method informs the run time engine immediately of this event and the current mouse cursor coordinates.

If this user mouse move action caused the mouse to move into one or more bounding rectangles of any text button or image object(s) at 254, or out of one or more bounding rectangles of any text button or image object(s) at 255, then appropriate two-dimensional status variables are set and the draw system is called. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop as described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the “mouse-Move” computational two-dimensional loop. If the mouse has entered into or out of the bounding rectangles of any image and/or text button object that has a defined text button, image and/or video pop-up object (See FIG. 16), then the draw system paints or effectively erases the appropriate background, text string, images and/or 3D frame for these pop-up objects. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur. If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks

63

can be constructed, in which certain designated sounds are played, or stopped, based purely on user mouse movement.

If the viewer moves the mouse into or out of the bounding rectangle for a paragraph hot link at 256, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number by paragraph line segment number) are set or reset and the draw system is called. The draw system paints the background color behind the hot link text string, and the color for the hot link text string in the hot link "mouse over" or the hot link normal colors. The text string may be underlined or in a bold font, depending on the settings.

If the viewer presses a mouse button at 257, the mouse-down method informs the run time engine immediately of this event and the current mouse cursor coordinates. If the viewer releases the mouse at 259, the mouse-up method informs the run time engine immediately of this event. Either way, if the original "mouse down" event had occurred inside one or more bounding rectangles of any text button or image object(s) at 258, then appropriate two-dimensional status variables are set and the draw system is called. The "mouse up" event will cause the run time engine to reset those appropriate two-dimensional status variables, and then call the draw system. If the mouse was inside of the bounding rectangles of any image and/or text button object that had a defined text button, image and/or video pop-up object (See FIG. 16) with the freeze attribute, the appropriate two-dimensional status variables are set so that the draw system will not erase these pop-up objects when the mouse moves outside the bounding rectangle(s) of the parent text button or image objects. The draw system interprets the relevant two-dimensional variables for the existing text button and image object(s) on the current web page in its draw loop described above with reference to FIG. 16, and draws the correct backgrounds, text strings, images, and 3D frames based on the state of each object, as just set by the "mouseDown" computational two-dimensional loop.

If the 3D frame had been previously defined (See FIG. 16) to have the "live" setting, then the 3D effect is changed from a "raised" effect to a "depressed" effect. If the location of any of these text button and image objects or child pop-up objects is changing over time because of time line, animation, or transformation threads, the draw system, as described in association with FIG. 33 and FIG. 34, is aware of these dynamics, and redraws the screen as these real time events occur, but does not recognize any new "mouse down" or "mouse up" Events. In one implementation, only "mouse over" events are recognized dynamically by the draw system.

If any sound or video events were defined (See FIG. 16) for any text button or image objects, then the run time engine plays those sound and/or video files or channels as defined. As multiple objects can be defined that each have associated sound (and even video) files, and these objects can be overlaid on each other, either completely or partially, very interesting synchronized multiple sound tracks can be constructed, in which certain designated sounds are played, or stopped, based purely on the user pressing a mouse button.

If a mouse down event as described previously in association with FIG. 16 was defined for one or more effected text button or image objects, only the object that was drawn last will have its event processed. If the event was to go to a different internal web page, then the run time engine sets the web page counter described in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the "mouse down" event was to go to an external web page in a different window, then the run time engine creates a new

64

browser window. In JAVA this can be accomplished with the "getAppletContext().showDocument(theURL, " _blank")" method, where "theURL" is the URL address for the external web page. The run time engine, however, continues executing.

If the mouse down event was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. In JAVA this can be accomplished with the "getAppletContext().showDocument(theURL)" method, where "theURL" is the URL address for the external web page.

If the viewer presses a mouse button while inside the bounding rectangle for a paragraph hot link at 260, then appropriate four-dimensional status variables (web page by paragraph number, by paragraph line number, by paragraph line segment number) are set. If the hot link setting at FIG. 16 was to go to a different internal web page, then the run time engine sets the web page counter described previously in association with FIG. 31 for the "current" page to be one less than that of the desired web page. The current web page's object time lines, if any, complete normally, and the desired web page is then executed. If the hot link setting was to go to an external web page in a different window, then the run time engine creates a new browser window and loads the external web page. The run time engine, however, continues executing. If the hot link setting was to go to an external web page in the same browser window, then the run time engine terminates by turning control over to the designated external web page. This completes the detailed description of the run time process at 261.

As is obvious from the descriptions of the various features and processes described above, it will be apparent to one skilled in the art that variations in form and detail may be made in the preferred implementation and methods without varying from the spirit and scope of the invention as defined in the claims or as interpreted under the doctrine of equivalents. It should also be clear that terms such as "browser", "mouse", "server", "web", etc., while adequate to describe the current state of interactive systems such as the World Wide Web, may evolve into new and more powerful entities. This evolution of terminology and technology is independent of the preferred implementation and methods of the invention as defined in the claims or as interpreted under the doctrine of equivalents. The preferred implementation and methods are thus provided for purposes of explanation and illustration, but not limitation.

I claim:

1. A system for assembling a web site comprising:

a server comprising a build engine configured to:

accept user input to create a web site, the web site comprising a plurality of web pages, each web page comprising a plurality of objects,

accept user input to associate a style with objects of the plurality of web pages, wherein each web page comprises at least one button object or at least one image object, and wherein the at least one button object or at least one image object is associated with a style that includes values defining transformations and time lines for the at least one button object or at least one image object; and wherein each web page is defined entirely by each of the plurality of objects comprising that web page and the style associated with the object, produce a database with a multidimensional array comprising the objects that comprise the web site including data defining, for each object, the object style, an object number, and an indication of the web page that each object is part of, and

65

provide the database to a server accessible to web browser;
wherein the database is produced such that a web browser with access to a runtime engine is configured to generate the web-site from the objects and style data extracted from the provided database.

2. The system of claim 1,
wherein one of said plurality of objects is a child, and
wherein the build engine is configured to accept user input to associate a style with child button and child image objects.

66

3. The system of claim 2, wherein at least one of said styles includes values defining time lines for child button and child image objects.

4. The system of claim 1, wherein at least one of said styles includes settings for multiple object states.

5. The system of claim 1, further including file size reduction means for reducing the total size of files generated by said build engine to a size of between 12K and 50K.

6. The system of claim 1, where said data is stored as one or more of a Boolean an integer, a string, a floating point variables, or a URL.

* * * * *

EXHIBIT C



US009063755B2

(12) **United States Patent**
Rempell et al.

(10) **Patent No.:** **US 9,063,755 B2**
(45) **Date of Patent:** **Jun. 23, 2015**

(54) **SYSTEMS AND METHODS FOR
PRESENTING INFORMATION ON MOBILE
DEVICES**

(75) Inventors: **Steven H. Rempell**, Novato, CA (US);
David Chrobak, Clayton, CA (US); **Ken
Brown**, San Martin, CA (US)

(73) Assignee: **Express Mobile, inc.**, Novato, CA (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 154 days.

(21) Appl. No.: **12/936,395**

(22) PCT Filed: **Apr. 6, 2009**

(86) PCT No.: **PCT/US2009/039695**

§ 371 (c)(1),

(2), (4) Date: **Nov. 3, 2010**

(87) PCT Pub. No.: **WO2009/126591**

PCT Pub. Date: **Oct. 15, 2009**

(65) **Prior Publication Data**

US 2011/0107227 A1 May 5, 2011

Related U.S. Application Data

(60) Provisional application No. 61/166,651, filed on Apr.
3, 2009, provisional application No. 61/113,471, filed
on Nov. 11, 2008, provisional application No.
61/123,438, filed on Apr. 7, 2008.

(51) **Int. Cl.**

G06F 3/048 (2013.01)

G06F 9/44 (2006.01)

G06F 9/45 (2006.01)

(52) **U.S. Cl.**

CPC **G06F 9/4443** (2013.01)

(58) **Field of Classification Search**

CPC **G06F 3/048**

USPC **715/738**

See application file for complete search history.

(56)

References Cited

U.S. PATENT DOCUMENTS

2004/0055017	A1 *	3/2004	Delpuch et al.	725/110
2004/0163020	A1 *	8/2004	Sidman	714/100
2005/0149935	A1 *	7/2005	Benedetti	718/102
2005/0273705	A1 *	12/2005	McCain	715/513
2006/0063518	A1 *	3/2006	Paddon et al.	455/418

OTHER PUBLICATIONS

Stina Nylander et al. "The Ubiquitous Interactor-Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Jan. 2004, pp. 271-282).*

* cited by examiner

Primary Examiner — Jennifer To

Assistant Examiner — Xuyang Xia

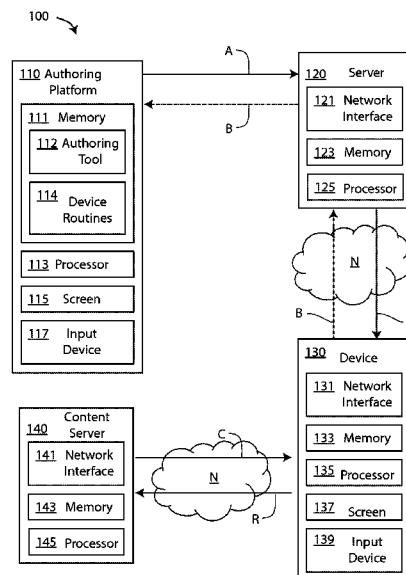
(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57)

ABSTRACT

Embodiments of a system and method are described for generating and distributing programming to mobile devices over a network. Devices are provided with Players specific to each device and Applications that are device independent. Embodiments include a full-featured WYSIWYG authoring environment, including the ability to bind web components to objects.

28 Claims, 18 Drawing Sheets



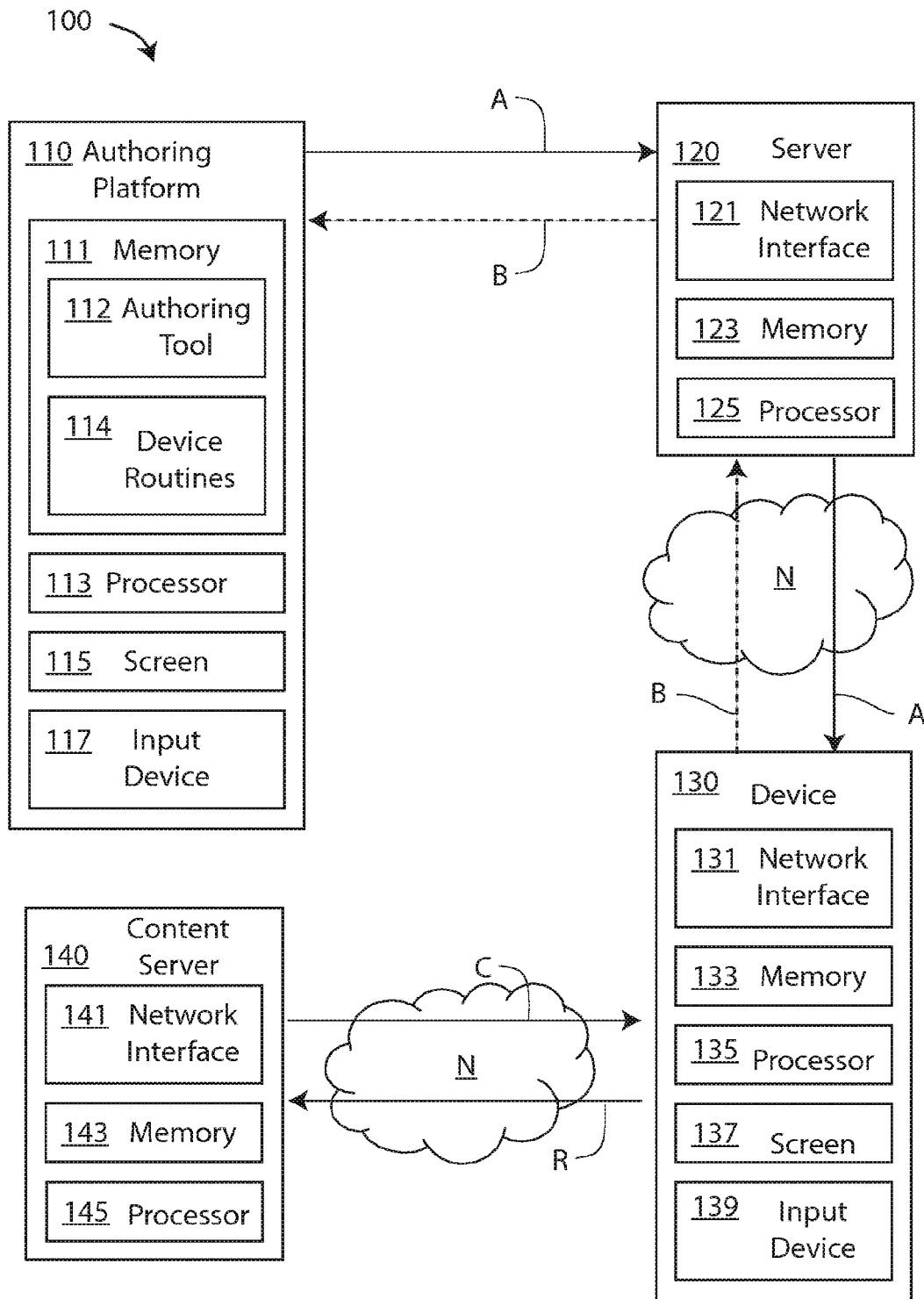


FIG. 1A

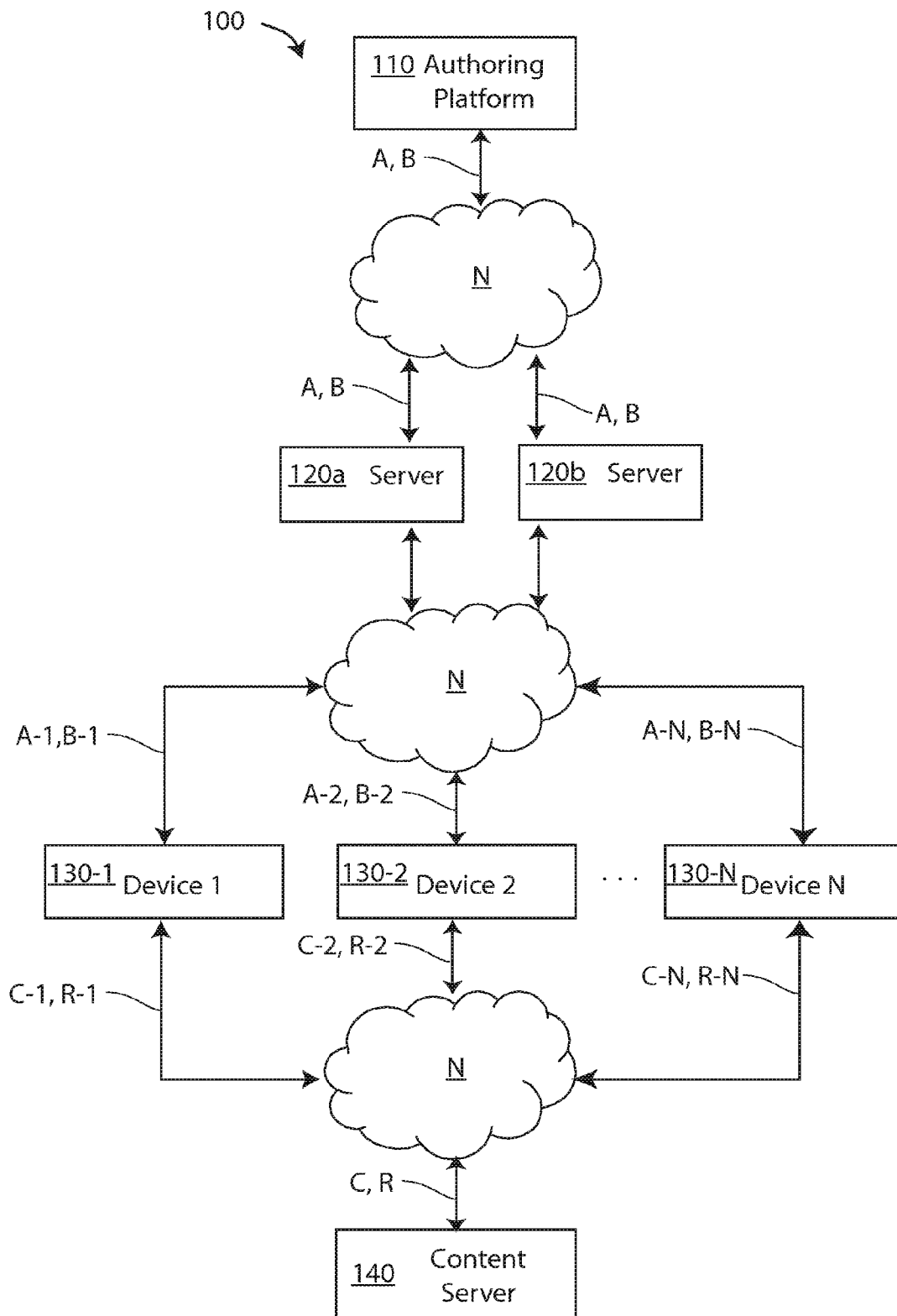


FIG. 1B

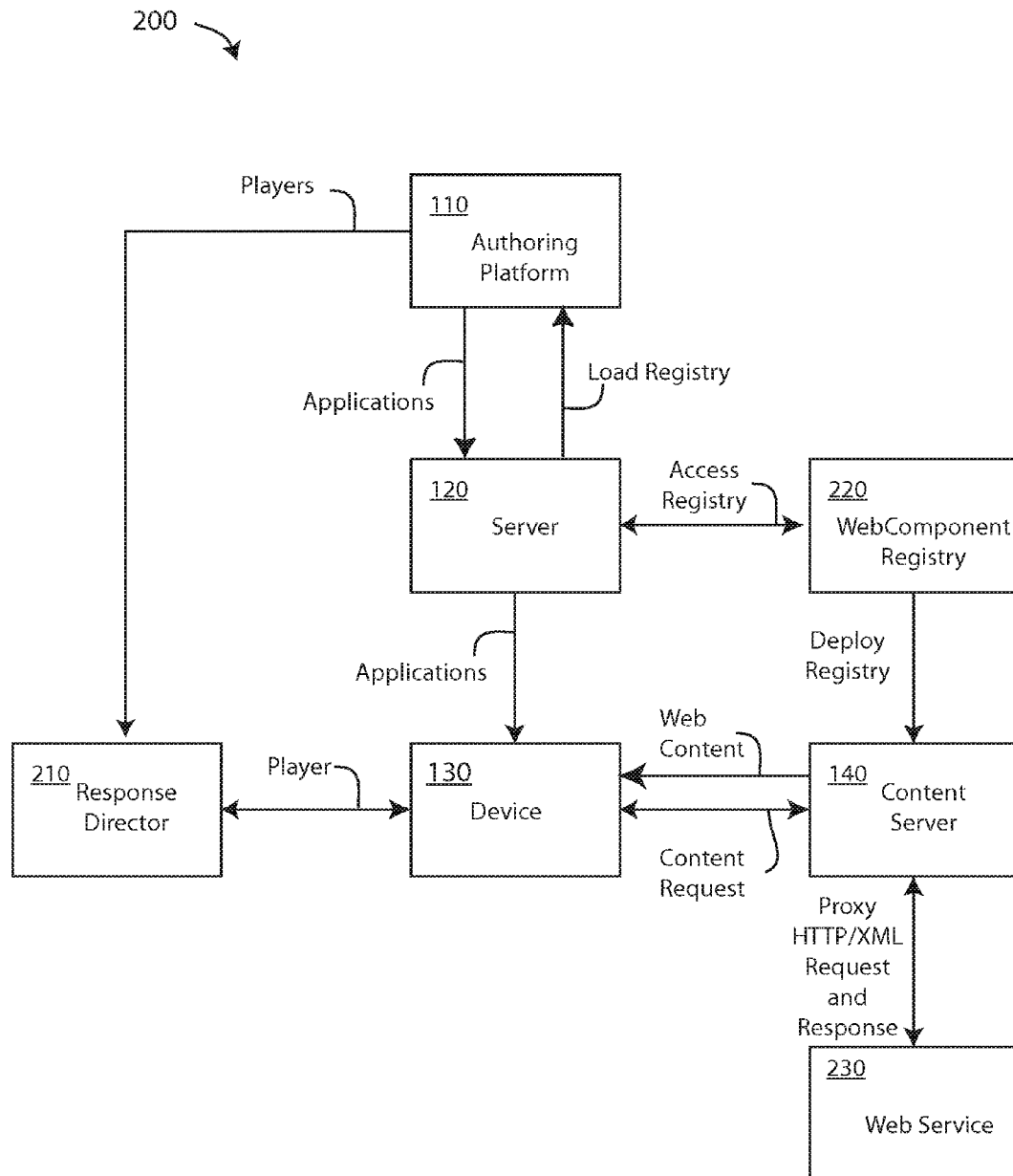
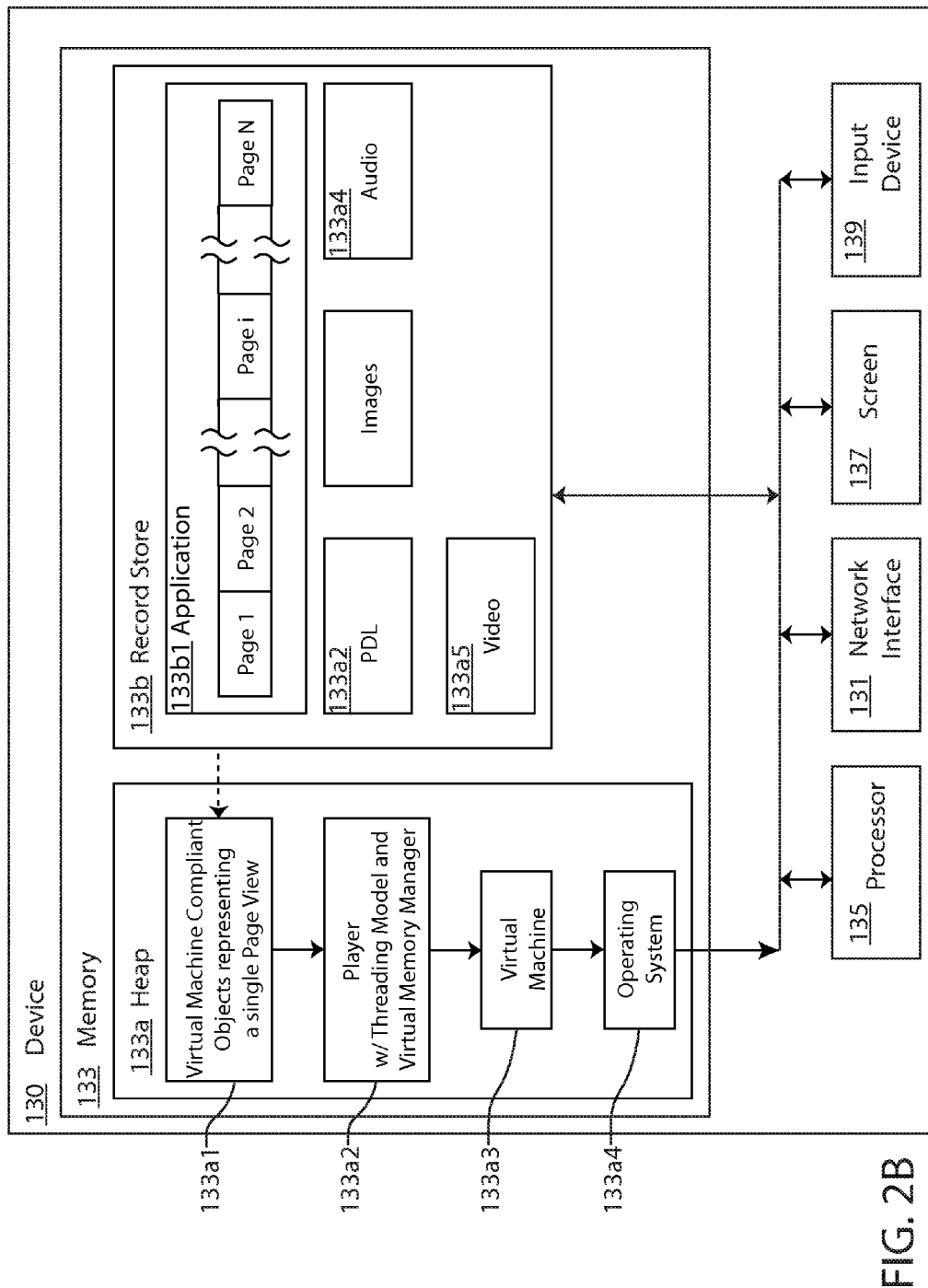


FIG. 2A



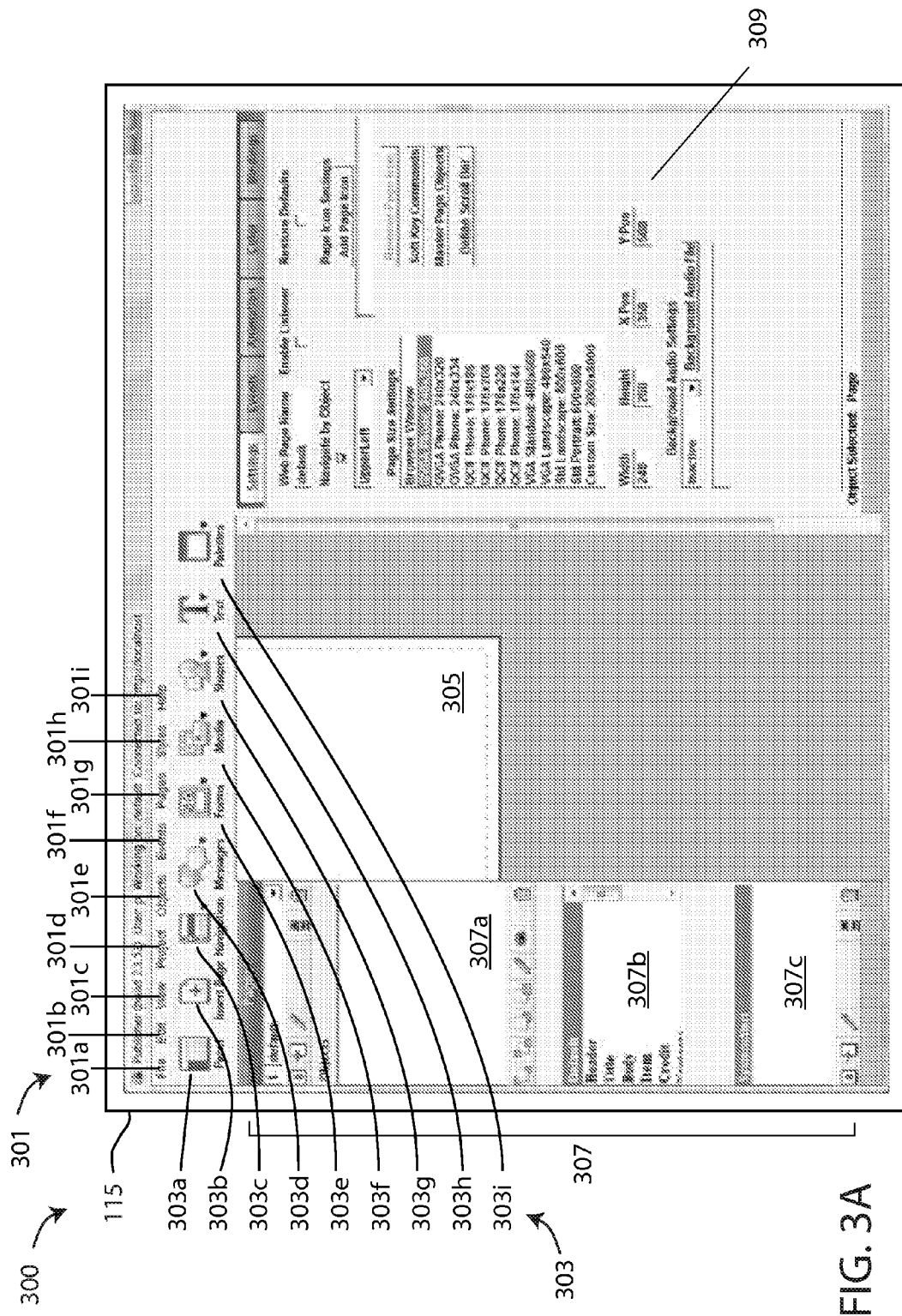


FIG. 3A

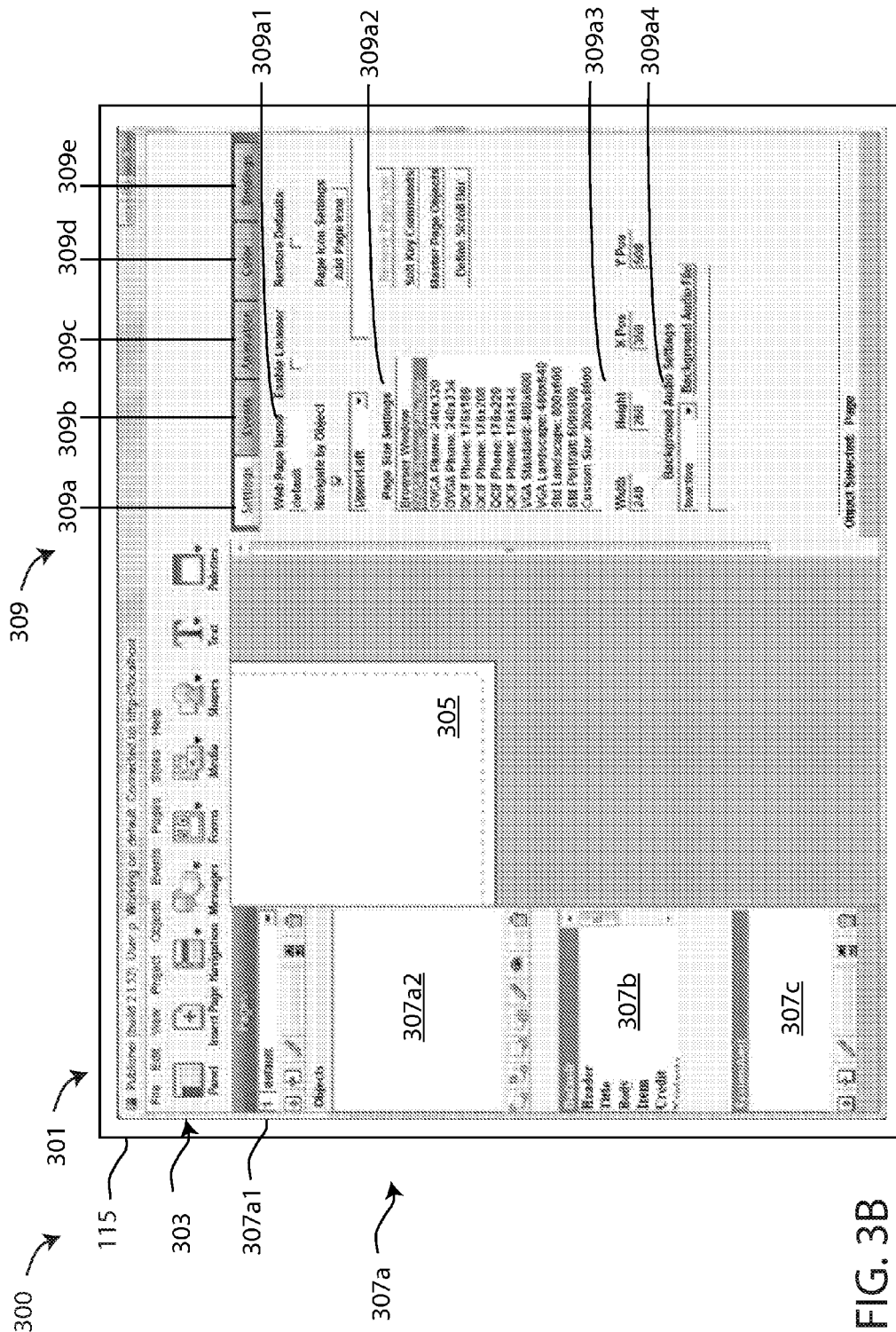


FIG. 3B

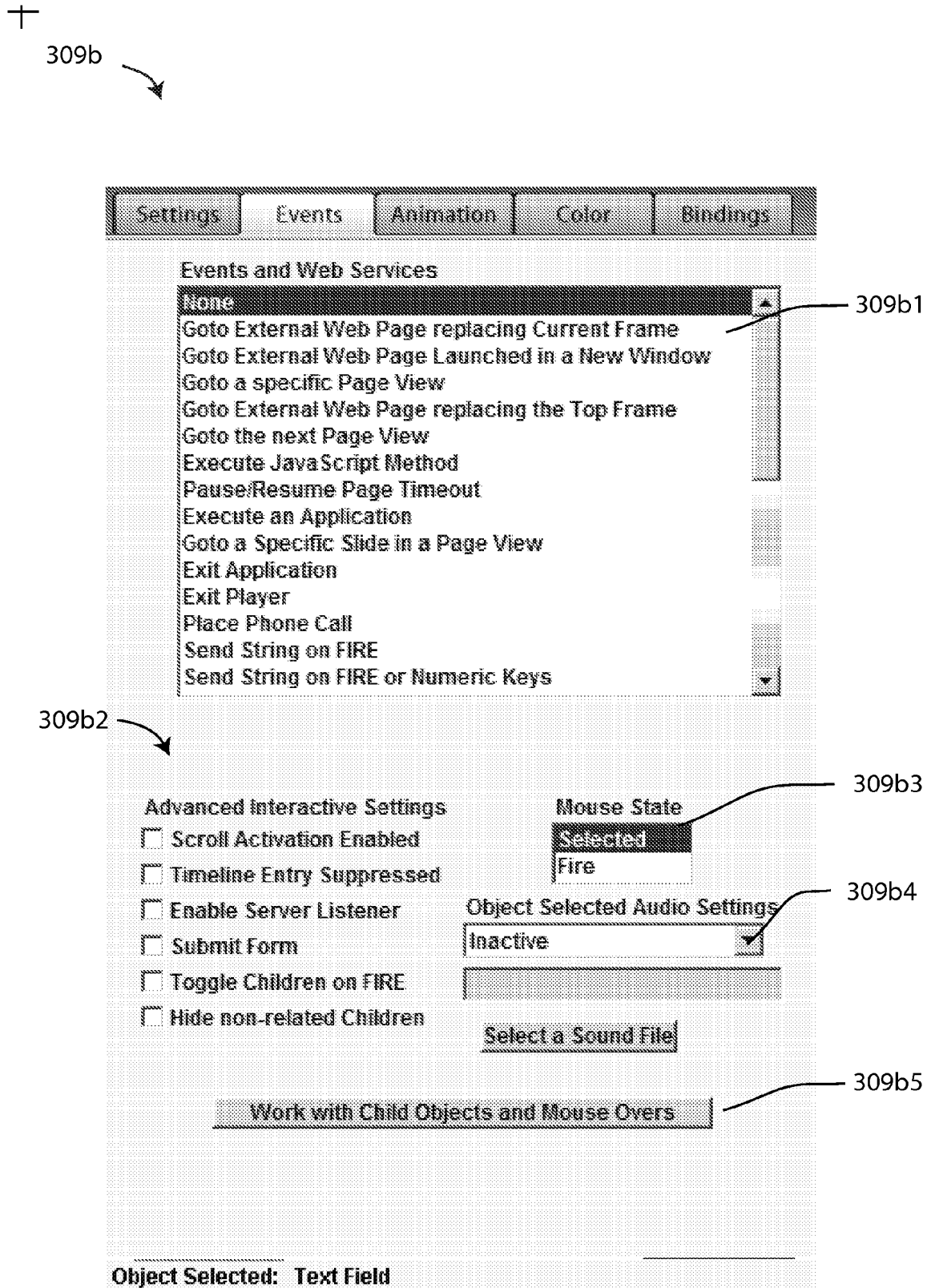


FIG. 3C

309c

Settings Events Animation Color Bindings

Object Entry Timeline Specifications

Activate Timeline Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

Object Animation Specifications

Delay (Sec)	Direction	Movement	Duration (Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	.1	2
2	Scroll Right	Fade In	.2	3
3	Custom	Fade Out	.3	4
4	Multi-Point		.4	5
5	Seek Cursor		.5	6
6	Attach		.6	7
7	Deposit		.7	8
8	Send Home		.8	9
9	Carom N		.9	10
10	Carom NE		1.0	11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles Custom Zoom % Avoid Cursor Dampen Anim.

1 0 ☒ ☐

Object Exit Timeline Specifications

Activate Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

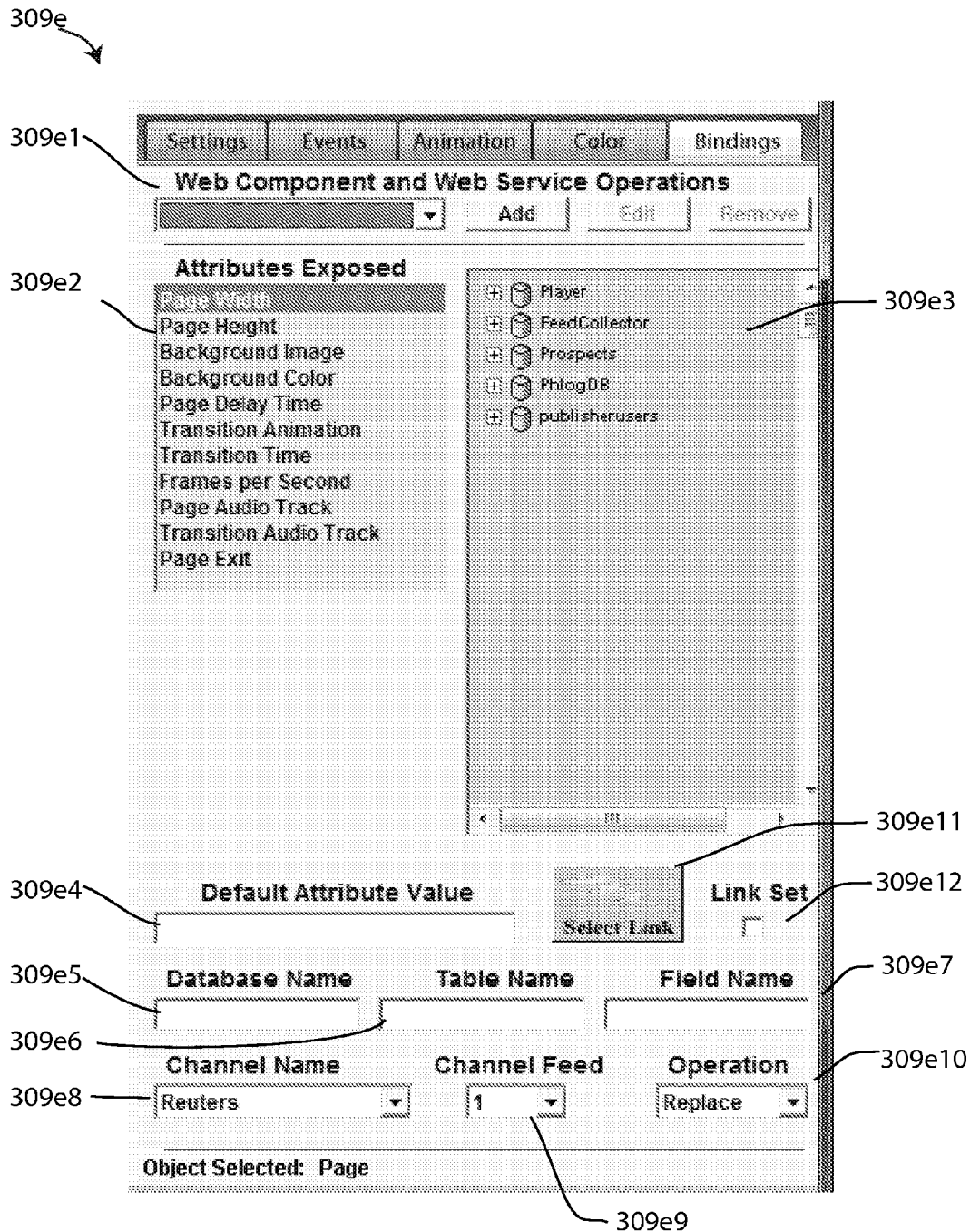


FIG. 3E

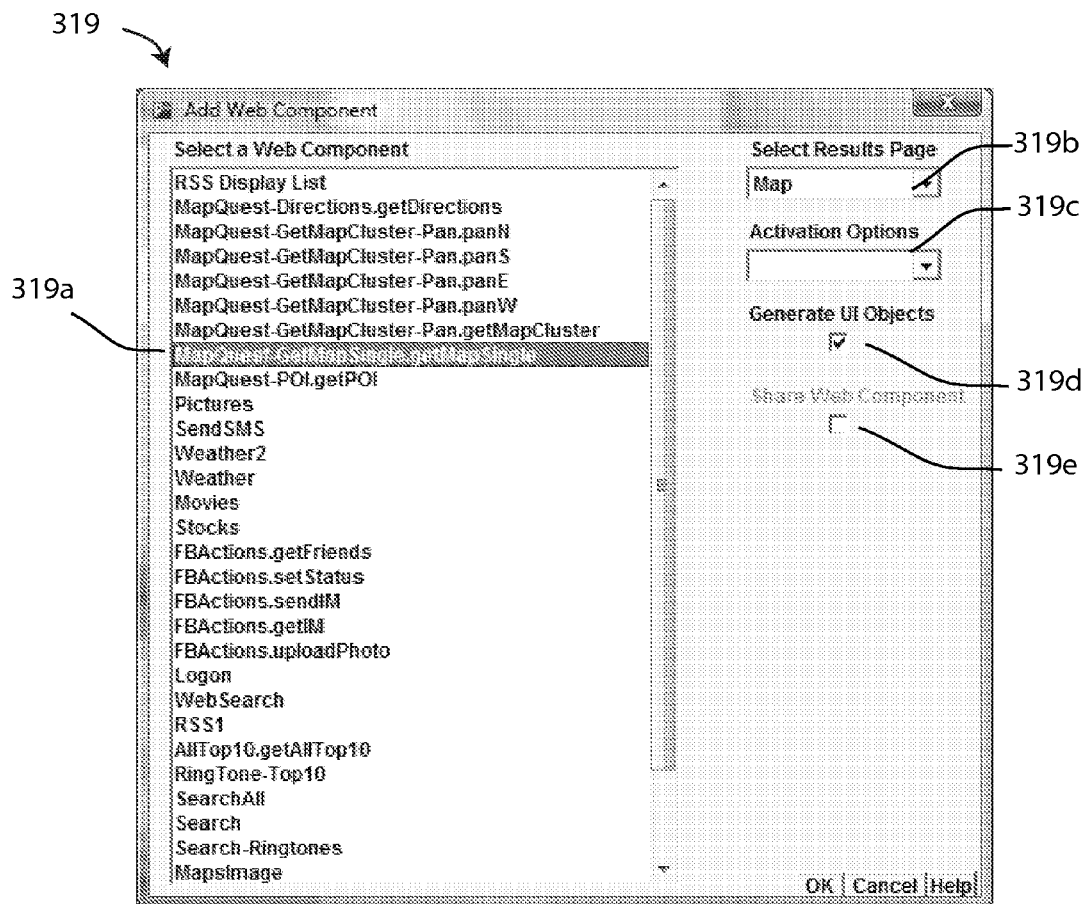


FIG. 3F

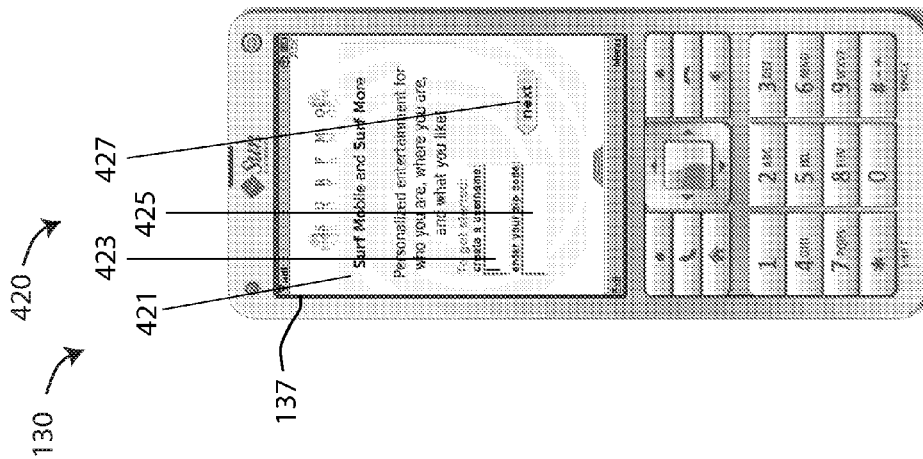


FIG. 4B

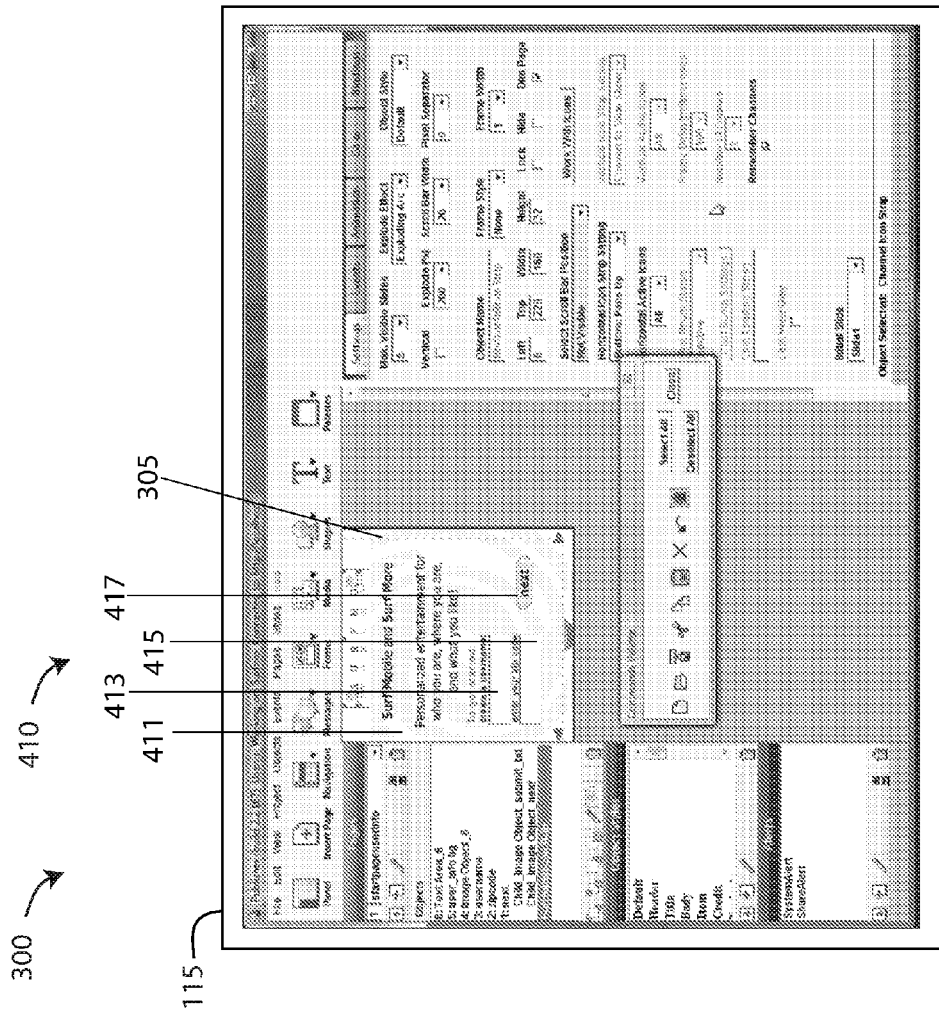
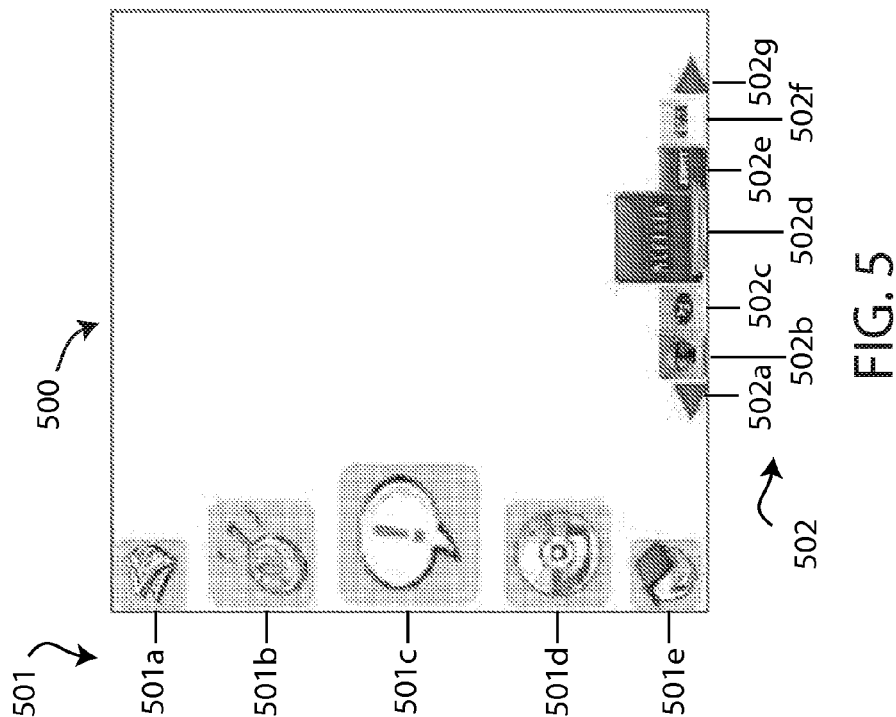
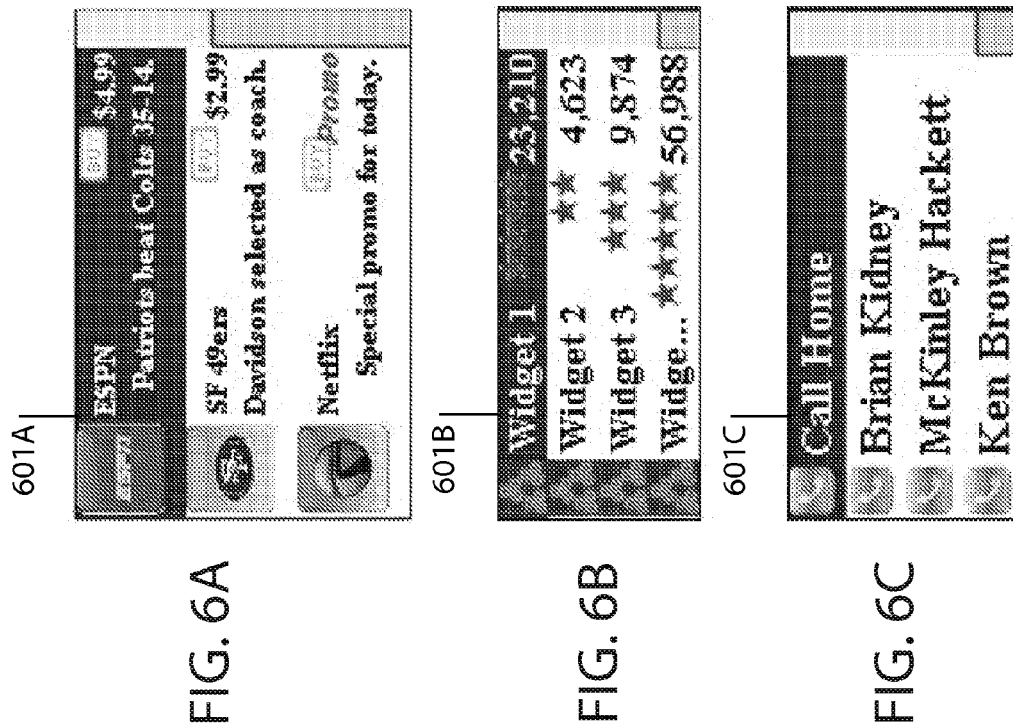


FIG. 4A



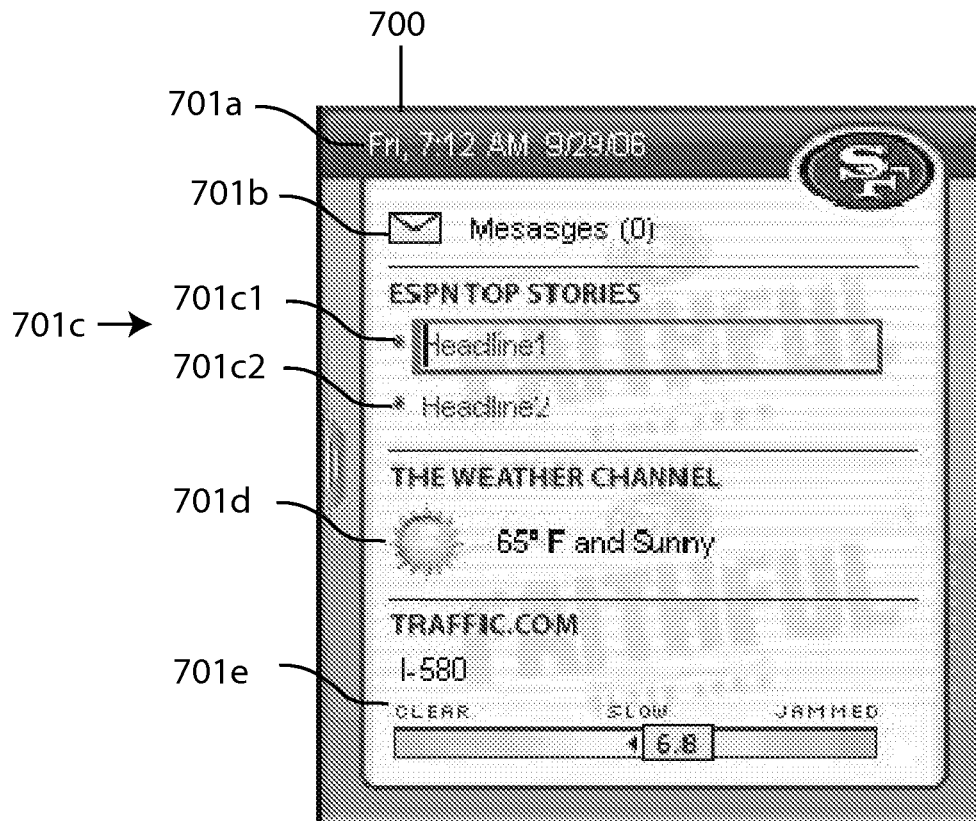


FIG. 7

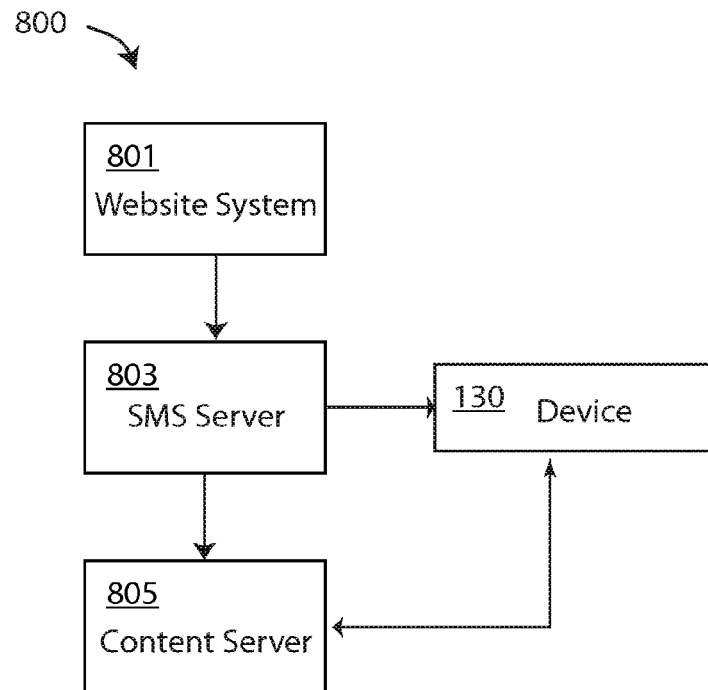


FIG. 8

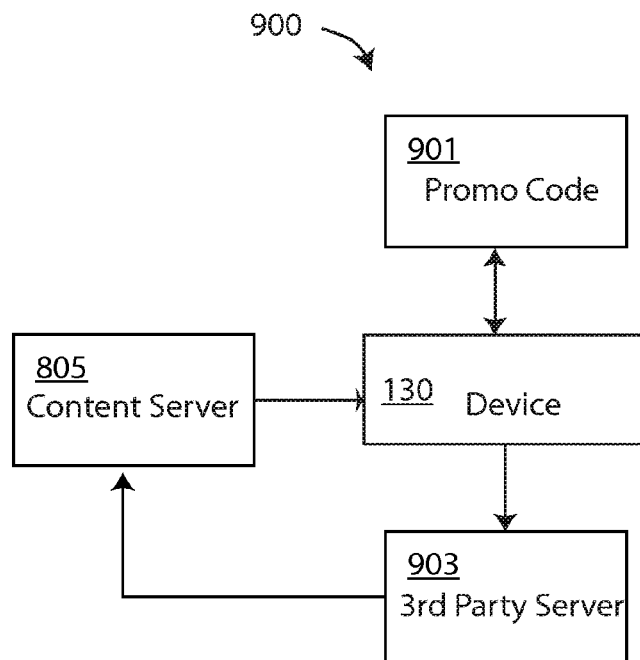


FIG. 9

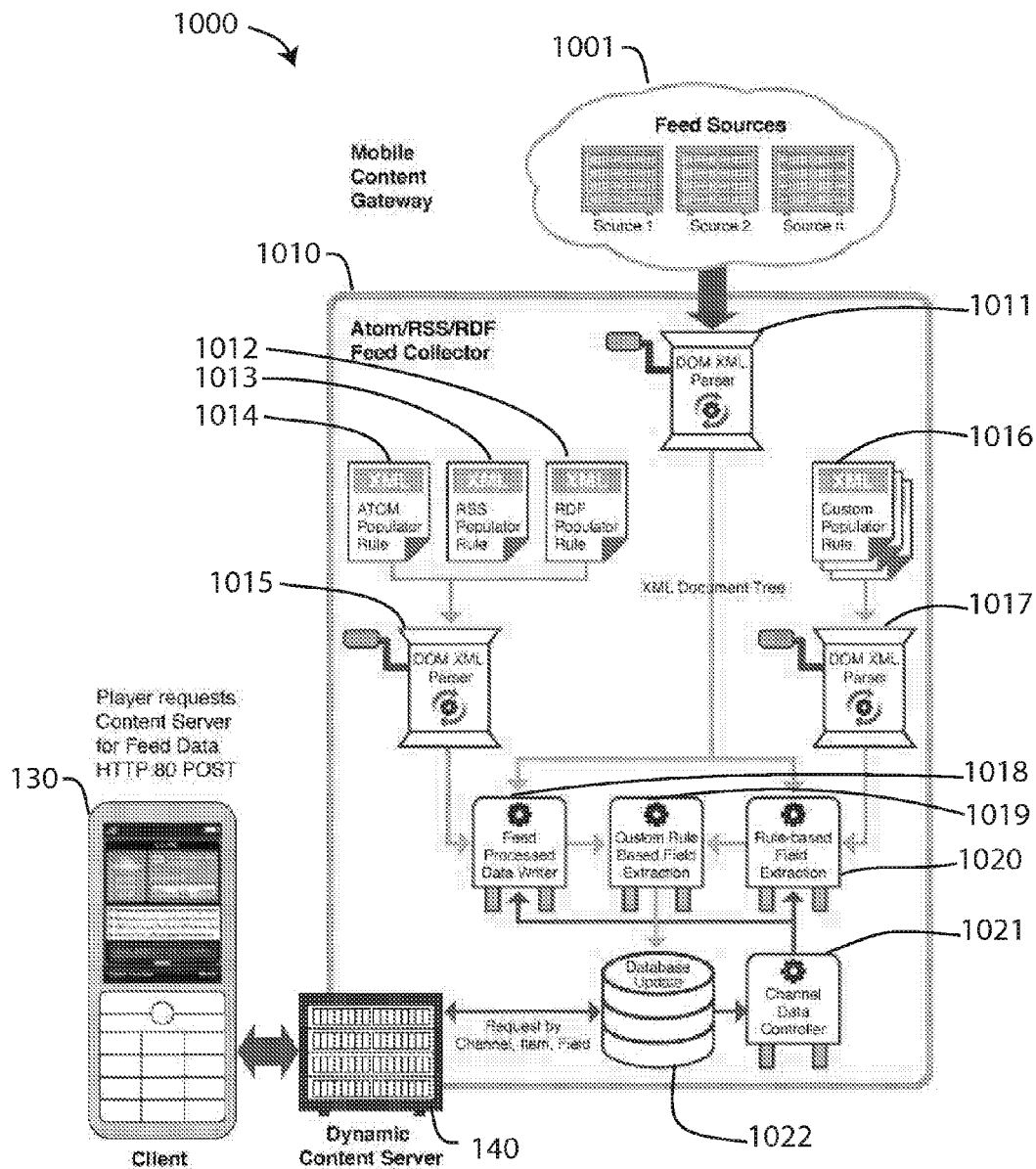


FIG. 10

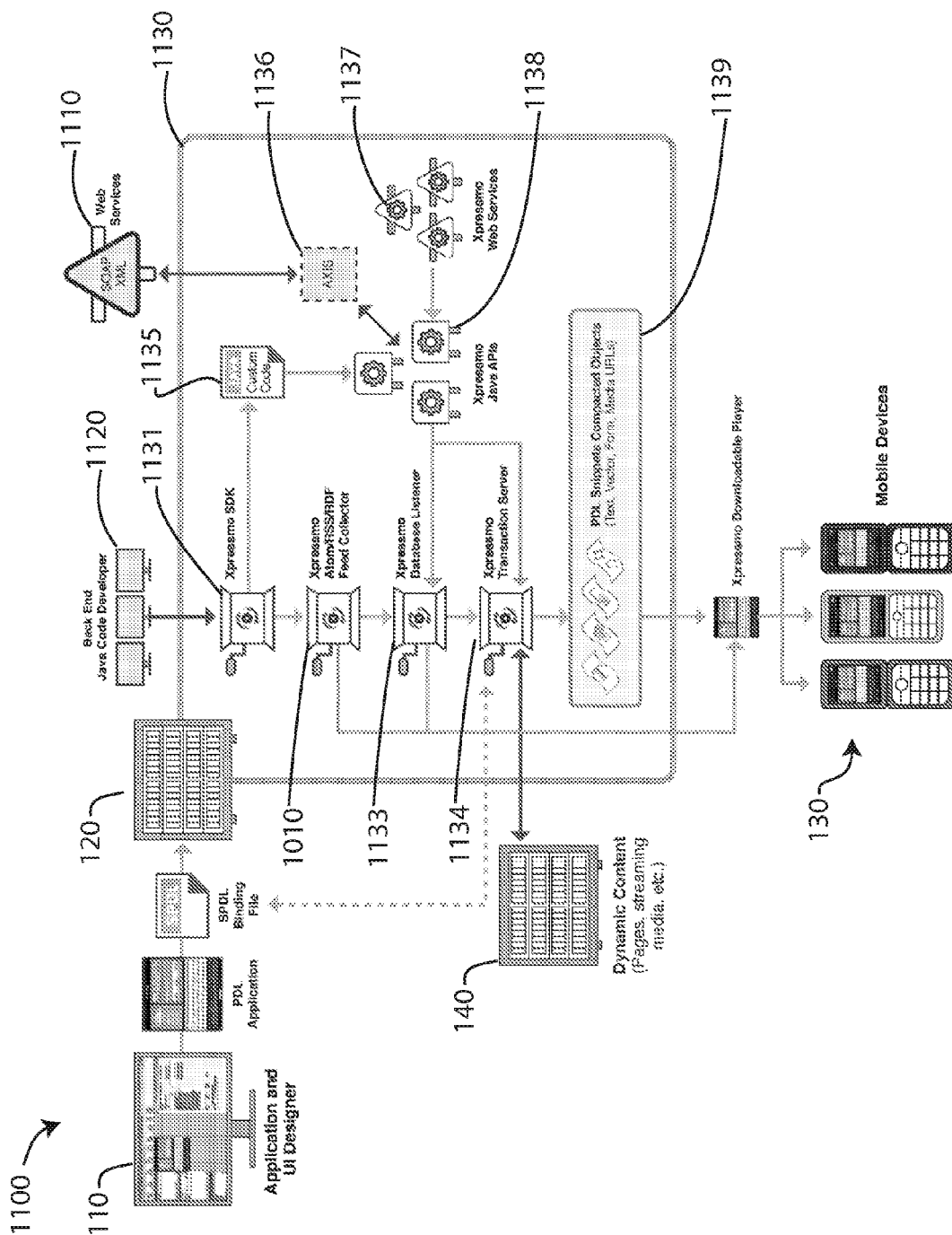


FIG. 11

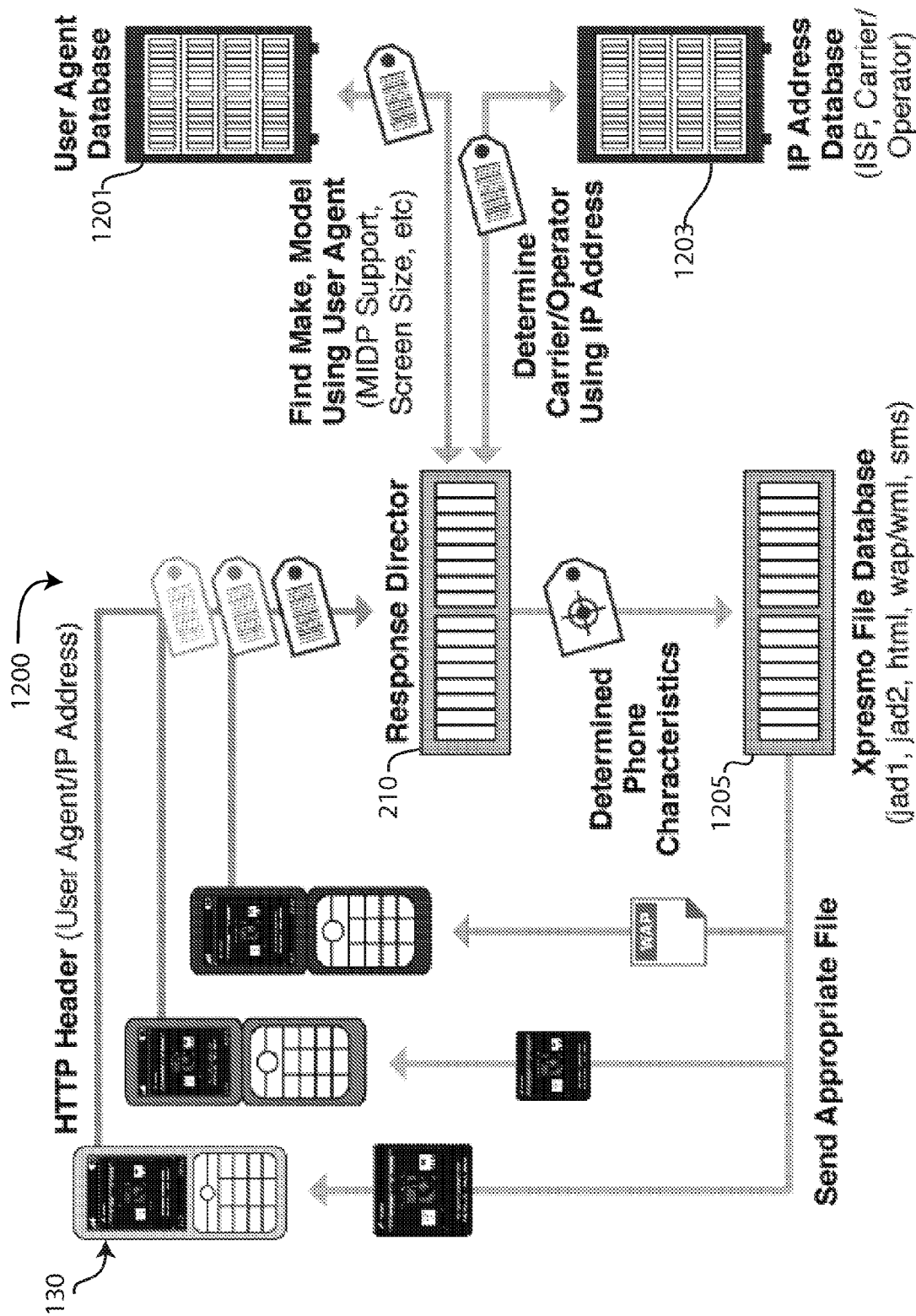


FIG. 12

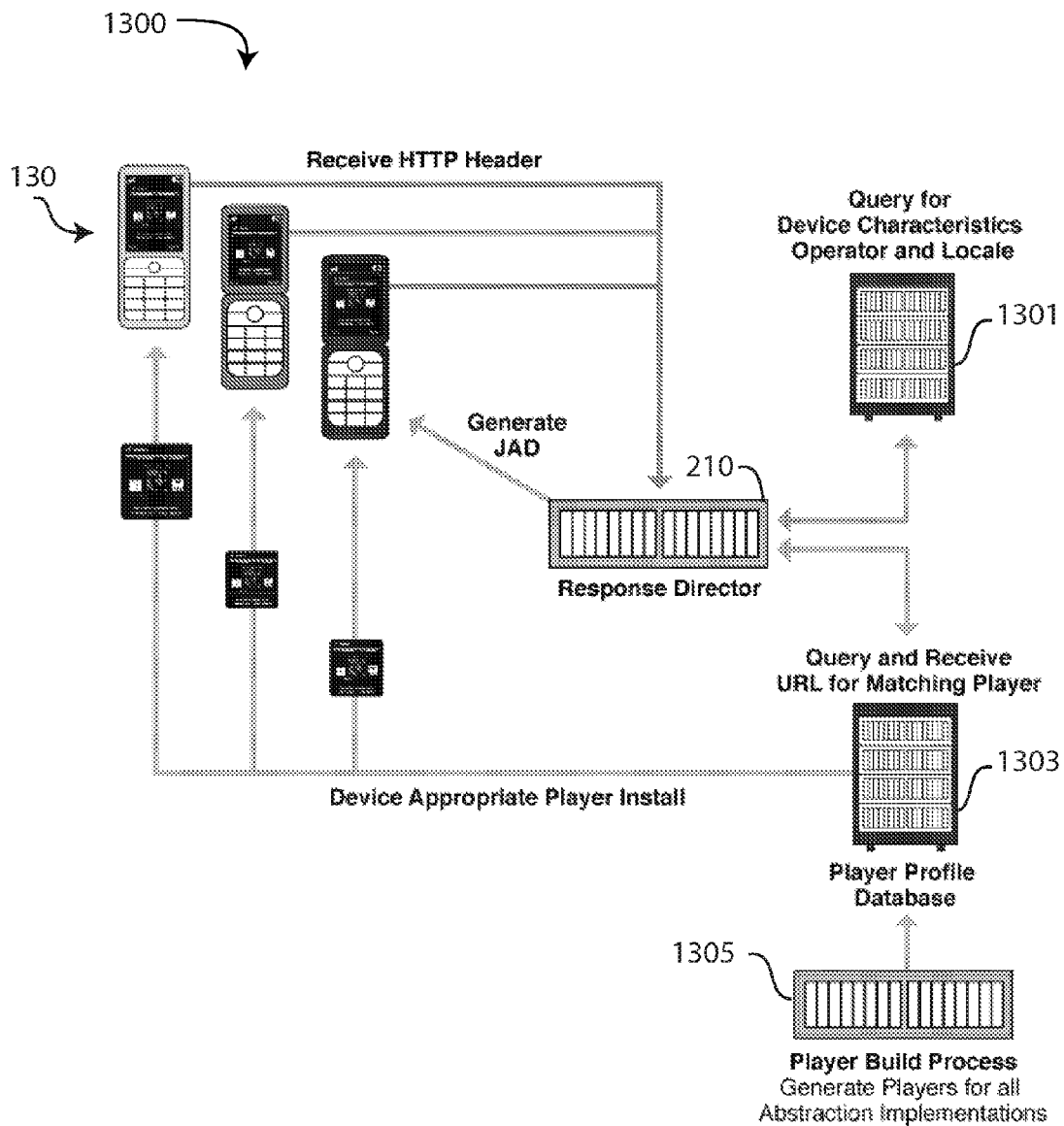


FIG. 13

1

SYSTEMS AND METHODS FOR PRESENTING INFORMATION ON MOBILE DEVICES

TECHNICAL FIELD

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

BACKGROUND ART

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

DISCLOSURE OF INVENTION

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab'

FIG. 3D illustrates one embodiment of an Animation Tab;

FIG. 3E illustrates one embodiment of Bindings Tab;

FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips;

FIG. 6A is a display of a Channel Selection List;

FIG. 6B is a display of a Widget Selection List;

FIG. 6C is a display of a Phone List;

FIG. 7 shows a display of a mash-up;

FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

MODE(S) FOR CARRYING OUT THE INVENTION

FIG. 1A is an illustrative schematic of one embodiment of a system 100 including an authoring platform 110 and a server 120 for providing programming instructions to a

device 130 over a network N. In one embodiment, device 130 is a wireless device, and network N includes wireless communication to the device. Alternatively, system 100 may provide access over network N to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform 110 may produce programming instructions or files that may be transmitted over network N to operate device 130, including instructions or files that are sent to device 130 and/or server 120. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices 130 and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network N, and that, in particular, the location is not limited to any particular server, for example.

Device 130 may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface 131, a memory 133, a processor 135, a screen 137, and an input device 139. Network interface 131 is used by device 130 to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory 133 includes programming required to operate device 130 (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface 131, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen 137 is a touch screen, providing the functions of the screen and input device 139.

Authoring platform 110 includes a computer or computer system having a memory 111, a processor 113, a screen 115, and an input device 117. It is to be understood that memory 111, processor 113, screen 115, and input device 117 are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory 111 may also instruct authoring platform 110 to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory 111 is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool 112. In one embodiment, authoring tool 112 is a graphical system for designing the layout of features as a display that is to appear on screen 137. One example of authoring tool 112 is the CDER™ publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device 130 may include an operating system having a platform that can interpret certain routines. Memory 111 may optionally include programming referred to herein, and without limitation, as routines 114 that are executable on device 130.

Routines 114 may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices 130, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for devices using routines provided on a platform. Thus as an example, routines

114 may include Java API's and an authoring tool System Development Kit (SDK) for specific devices 130.

Server 120 is a computer or computer system that includes a network interface 121, a memory 123, and a processor 125. Is to be understood that network interface 121, memory 123, and processor 125 are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform 110; accept input and/or provide output through network interface 121 over network N to network interface 131; or store information from authoring platform 110 or from device 130 for transmission to another device or system at a later time.

In one embodiment, authoring platform 110 permits a user to design desired displays for screen 137 and actions of device 130. In other words, authoring platform 110 is used to program the operation of device 130. In another embodiment, authoring platform 110 allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory 123 and may then be sent, when requested by device 130 or when the device is otherwise accessible, over network N, through network interface 130 for storage in memory 133.

In an alternative embodiment, analytics information from devices 130 may be returned from device 130, through network N and server 120, back to authoring platform 110, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server 140 is a computer or computer system that includes a network interface 141, a memory 143, and a processor 145. It is to be understood that network interface 141, memory 143, and processor 145 are configured such that a stored program in the memory may be executed by the processor to accept requests R from device 130 and provide content C over a network, such as web server content the Internet, to device 130.

FIG. 1B is schematic of an alternative embodiment system 100 for providing programming instructions to device 130 over a network N that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system 100 may include multiple servers 120 and/or multiple devices 130.

In the embodiment of FIG. 1B, system 100 is shown as including two or more servers 120, shown illustratively and without limitation as servers 120a and 120b. Thus some of the programming or information between authoring platform 110 and one or more devices 130 may be stored, routed, updated, or controlled by more than one server 120. In particular, the systems and methods described herein may be executed on one or more server 120.

Also shown in FIG. 1B are a plurality of devices 130, shown illustratively and without limitation as device 130-1, 130-1, . . . 130-N. System 100 may thus direct communication between individual server(s) 120 and specific device(s) 130.

As described subsequently, individual devices 130 may be provided with program instructions which may be stored in each device's memory 133 and where the instructions are executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different

5

“device platforms.” Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices 130 are provided with some programming from authoring system 100 that is particular to the device.

In one embodiment, system 100 provides permits a user of authoring platform 110 to provide instructions to each of the plurality of devices 130 in the form of a device- or device-platform specific instructions for processor 135 of the device, referred to herein and without limitation as a “Player,” and a device-independent program, referred to herein and without limitation as an “Application.” Thus, for example, authoring platform 110 may be used to generate programming for a plurality of devices 130 having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device 130 utilizes a Player and an Application to execute programming from authoring platform 110. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device 130 (“activated”) through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device 130. In yet another alternative embodiment, Player is activated through a signal to device 130 by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory 133, the functioning of device 130 may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory 133 and which, when executed by processor 135, generate the designed displays on screen 137. The Application and Player may also include programming instructions which may be stored in memory 133 and which provide instructions to processor 135 to accept input from input device 139.

Authoring tool 112 may, for example, produce and store within memory 111a plurality of Players (for different devices 130) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers 120 and then provided to individual devices 130. In general, Applications are provided to device 130 for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device 130.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device 130. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen 137, and the Player may interpret the Java and instruct processor 135 to produce the display according to the Application for execution on a specific device 130 according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen 137, instructions for accepting input from input device 139, instructions

6

for interacting with a user of device 130, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device’s Player interprets or executes the Application to generate one or more “pages” (“Applications Pages”) on a display as defined by the PDL. The Player may include code that is device-specific—that it, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool 112 may thus be used to design one or more device-independent Applications and may also include information on one or more different devices 130 that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system 100 provides Players and Applications to one server 120, as in FIG. 1A. In another embodiment, system 100 provides Players to a first server 120a and Applications to a second server 120b, as in FIG. 1B.

In one embodiment, authoring tool 112 may be used to program a plurality of different devices 130, and routines 114 may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a “thin client”—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform 110 allows user to arrange objects for display on screen. A graphical user interface (“GUI,” or “UI”) is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform 110 also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform 110 also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor 135 to store or modify information in memory 133, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool 112, the Application, and Player. Thus while either designing the Application with the authoring tool 112, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in U.S. Pat. No. 6,546,397 to Rempell (“Rempell”), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a data-

base. The terminology used here is a PDL, that is the “internal database” of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server

120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring platform 110 which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service 230 to a GUI component of the Application as will be displayed on screen 137. In addition, multiple components of one or more web service 230 can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service 230, are stored in the PDL. The content server 140

handles all communication between device **130** and the web service **230** and can be automatically deployed as a web application archive to any content server.

Device **130**, upon detecting an event in which a component of a web service **230** has been defined, assembles and sends all related inputs to content server **240**, which proxies the request to web service **230** and returns the requested information to device **130**. The Player on device **130** then takes the outputs of web service **230** and binds the data to the UI components in the Application, as displayed on screen **137**.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform **110** wants to present an ATOM feed on device **130**, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server **120** and processed by device **130** at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services **230** are available either to the user of authoring platform **110** or otherwise accessible through the SDK and Java APIs of routines **114**. System **200** permits an expanding set of components of web services **230** including, but not limited to: server pages from content server **120**; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services **230** defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System **200** also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen **137**. Thus, for example, a user of authoring platform **110** may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device **130**, using system **200**. Authoring platform **110** may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices **130**. In addition, Authoring platform **110** may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text

element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services **230** that may be provided using system **200**, a user of authoring platform **110** can place, for example, Yahoo maps into device **130** by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform **110**. System **200** also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device **130** includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform **110** having access to platform dependent routines **114**, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform **110** will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device **130**, much like how a web browser downloads an HTML page through a external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services **230** include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform **110**, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined Icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device **130** illustrating an embodiment of the programming generated by authoring platform **110**. Memory **133** may include several different logical portions, such as a heap **133a**, a record store **133b** and a filesystem (not shown).

As shown in FIG. 2B, heap **133a** and record store **133b** include programming and/or content. In general, heap **133a** is readily accessible by processor **135** and includes, but is not limited to portions that include the following programming: a portion **133a1** for virtual machine compliant objects representing a single Page View for screen **137**; a portion **133a2** for

11

a Player; a portion **133a3** for a virtual machine; and a portion **133a4** for an operating system.

Record store **133b** (or alternatively the filesystem) includes, but is not limited to, portions **133b1** for Applications and non-streaming content, which may include portions **133a2** for images, portions **133a4** for audio, and/or portions **133a5** for video, and portions **133b2** for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlaid on the current page view. Other content, such as streaming content may be provided from network interface **131** directly to the Media Codec of device **130** with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at a time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion **133a1** as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieve from the Record store, converted into virtual machine compliant objects (by processor **135** and according to operation by the Player, Virtual Machine, etc). and stored in heap **133a**. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions **133a1**, **133a2**, **133a3**, **133a4**, record store **133b** and processor **135** are illustrative of the logical connection between the different types of programming stored in Heap **133a** and record store **133b**, that is, how data is processed by processor **135**.

The Player determines which of the plurality of Applications Pages in portion **133b1** is required next. This may be determined by input actions from the Input Device **139**, or from instructions from the current Application Page. The Player instructs processor **135** to extract the PDF from that Applications Page and store it in portion **133a1**. The Player then interprets the Application Page extracted from PDL which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions **133a3**, **133a4**, **133a5**, respectively.

The Virtual Machine in portion **133a3** processes the Player output, the Operating System in portion **133a3** processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion **133a4**.

In another embodiment, the Player is a native program that interacts directly with the operating system.
Embodiments of a Publishing Environment

In one embodiment, authoring platform **110** includes a full-featured authoring tool **112** that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool **112** permits a user to design an Application by placing objects on canvas **305** and optionally assigning actions to the objects and save the Application. System **100** then provides the Application and Player to a device **130**. The Application as it runs on device **130** has the same look and operation as designed on authoring platform

12

110. In certain embodiments, authoring platform **110** is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform **110** produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device **130** may, according to its contents, modify what is displayed on screen **137** or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool **112** allows a user to place one or more objects on canvas **305** and associate the objects with an Applications Pages. Authoring platform **110** maintains a database of object data in memory **111**, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool **112**, which are also maintained in memory **111**. Authoring tool **112** also allows a user to define more than one Applications Page.

In another embodiment, authoring tool **112**, provides Java programming functions of the Java API for specific devices **130** as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform **110** to position objects that, after being provided as an Application to device **130**, activate such Java functions on the device.

In certain embodiments, authoring platform **110**, as part of system **100**, permits designers to include features of advanced web and web services Applications for access by users of device **130**. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform **110** through interactive object libraries.

In certain embodiments, authoring platform **110**, as part of system **100**, allows the inclusion of child objects which may eventually be activated on device **130** by the user of the device or by time. The uses of the child objects on device **130** include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

In certain other embodiments, authoring platform **110**, as part of system **100**, provides advanced interactive event models on device **130**, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform **110** may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device **130**.

In certain embodiments, authoring platform **110**, as part of system **100**, provides full style inheritance on device **130**. Thus, for example and without limitation, both master page

13

inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool 112 may create various text objects on canvas 305 using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool 112 changes the font color of a specific button to green. If later, the user of authoring tool 112 changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform 110 provides page view, style, object, widget and Application template libraries. Authoring platform 110 may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform 110 to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours a seen on device 130.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface 300 as it appears, for example and without limitation, on screen 115 while executing authoring tool 112. In one embodiment, publisher interface 300 includes a Menu bar 301, a Tool bar 303, a Canvas 305, a Layer Inspector 307 having subcomponents of a page/object panel 307a, an object style panel 307b, and a page alert panel 307c, and a Resource Inspector 309.

In general, publisher interface 300 permits a user of authoring platform 110 to place objects on canvas 305 and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface 300 permits a user to program a graphical interface for the screen 137 of device 130 on screen 115 of authoring platform 110, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device 130 when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool 112 maintains, in memory 111, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool 112 further maintains, in memory 111, a listing of the objects. As the user selects objects, publisher interface 300

14

provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory 111.

In one embodiment, publisher interface 300 is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas 305 placed and manipulated by the various commands within publisher interface 300 or inputs such as an input device 117 which may be a keyboard or mouse. As described herein, the contents of canvas 305 may be saved as an Application that, through system 100, provide the same or a similar placement of objects on screen 137 and have actions defined within publisher interface 300. Objects placed on canvas 305 are intended for interaction with user of device 130 and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface 300 may assign or associate actions or web bindings to UI objects placed on canvas 305 with result in the programming device 130 that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool 112, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer)	Text Field (integer)	Text Field (integer)
		Text Field (Phone #)		Text Field (Phone #)
		Text Field (SMS #)		Text Field (SMS #)
		Choice		Choice
		List (single select)		List (single select)
				Text Button

TABLE I-continued

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
String multilineString	Text Field (Alpha) Text Area	Any Text Area	Text Field (Alpha) Text Area	Any Text Area Paragraph
ImageURL	N/A	N/A	Image	Image Slide Show
VideoURL	N/A	N/A	Video	Video Slide Show
List	Single Item List	Single Item List Multi-Select List Complex List Choice Slide Show	Single Item List	Any List Type Any Choice Type (see Complex List Specification)
ComplexList	Complex List	Single Item List Multi-Select List Complex List	Single Item List	Any List Type (see Complex List Specification)
Slideshow	Slide Show	Slide Show	Slide Show	Slide Show
SearchResponseList	N/A	N/A	Search Response List	Search Response List Control List Complex List Choice
RSSList	N/A	N/A	RSS Display List	RSS Display List Control List Complex List Choice
SingleSelectionList	Choice	Choice Complex List	Choice	Choice Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image Video Slide Show
ChannelDescription	N/A	N/A	Text Area	Text Area Paragraph Text Field Text Button List Choice
ChannelTitle	N/A	N/A	Text Field	Text Field Text Button Paragraph Text Area List Choice
URL			Text Field (URL request)	Text Field (URL request)
Audio URL			Text Field (Audio URL request)	Text Field (Audio URL request)
Purchase URL			Text Field (Purchase URL request)	Text Field (Purchase URL request)
Image Data			Image	Image Slide Show
Image List Data			Slide Show	Slide Show Image
Persistent Variable	N/A	N/A	N/A	N/A
Pipeline Multiple Select	Multi-select List	Multi-select List Complex List Slide Show	N/A	N/A
Phone Number	Text Field (numeric type)	Text Field	Text Field (numeric type)	Text Field Text Button
Hidden Attribute	Complex List	Text Button Complex List Slide Show	Complex List	Complex List Slide Show
Collection List	N/A	N/A	Slide Show	Complex List Slide Show

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and

17

operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated parent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current

18

Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **301i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c**

19

displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303j** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within and Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

20

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshown) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define a unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not limited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and

Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services	
Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert.
Goto a specific Internal Web Page	"Backend Synchronization"
Goto the next Internal Web Page	Goto Widget Object
	Generate Alert.
	"With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children. Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen, Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for

example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319d**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3rd party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. 10.

Referring again to FIG. 3E, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define

23

the record. If the Feed Collector data base is selected, for example, then the RSS “Channel Name” and the “Channel Feed” drop down menus will be available for symbolically selected the record. For other data bases the RSS “Channel Name” and the “Channel Feed” drop down menus are replaced by a “Record ID” text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected. Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by “Record ID” if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the “Select Link” will cause the current data base selections to begin the blink in some manner, and the “Select Link” will change to “Create Link”. The user could still change the data base and attribute choices. Touching the “Create Link” will set the “Link Set” checkbox and the “Create Link” will be replaced by “Delete Link” if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. 4A and 4B according to the system **100**, where FIG. 4A shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. 4B shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform **110** is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. 4B. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. 4A, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes.

Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following

24

description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways.

Launch Strips and Graphical List Templates Launch Strips

Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. 5 shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip **502** also includes UI objects **502a** and **503g**, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device **130**. Launch strip **502** may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List **601a**, in FIG. 6B as a display of a Widget Selection List **601b**, and in FIG. 6C as a display of a Phone List **601c**. Lists **601a**, **601b**, and **601c** may be displayed on canvas **305** or on screen **137** of device **130** having the proper Player and Application. As illustrated, graphical lists **601a**, **601b**, and **601c** may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service **250** or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list **601a**, **601b**, and **601c** has several individual entries that are

25

each linked to specific actions. Thus Channel Selection List **601a** shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List **601b** includes several objects presenting different widgets for selecting. Phone List **601c** includes a list phone number objects of names that, when selected by a user of device **130** cause the number to be dialed. Entries in Phone List **601c** may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List **601c** may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform **110** allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

Personalization and Temporal Adoption

System **100** and **200** permit for the personalization of device **130** by a variety of means. Specifically, what is displayed on screen **137** may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display **700** of a mash-up which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **700** includes several object **701** that have been dynamically bound, including an indication of time **701a**, an indication of unread text messages **701b**, an RSS news feed **701c** (including 2 "ESPN Top Stories" **701c1** and **701c2**), components **701d** from two Web Services—a weather report ("The Weather Channel"), and a traffic report **701e** ("TRAFFIC.COM").

In assembling the information of display **700**, device **130** is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device **130** has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be

26

maintained as persistent storage on device **130** when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base.

Push Capable Systems

In another embodiment system **100** or **200** is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device **130** can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device **130**.

FIG. 8 is a schematic of an embodiment of a push enabled system **800**. System **800** is generally similar to system **100** or **200**. Device **130** is shown as part of a schematic of a push capable system **800** in FIG. 8. System **800** includes a website system **801** hosting a website **801**, a server **803** and a content server **805**. System **801** is connected to servers **803** and/or **805** through the Internet. Server **803** is generally similar to server **120**, servers **805** is generally similar to server **140**.

In one embodiment, a user sets up a weekly SMS update from website system **801**. System **801** provides user information to server **803**, which is an SMS server, when an update is ready for delivery. Server **803** provides device **130** with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website **801** also provides content server **805** with the content of the update. When a user of device **130** responds to the SMS query, the response is provided to content server **805**, which provides device **130** with updates including the subscribed content.

In an alternative embodiment of system **800**, server **803** broadcasts alerts to one or more devices **130**, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system **900**. System **900** is generally similar to system **100**, **200**, or **800**. In system **900** a user requests information using an SMS code, which is delivered to device **130**. System **900** includes a promotional code **901**, a third-party server **903**, and content server **805**. Server **803** is connected to servers **803** and/or **805** through the Internet, and is generally similar to server **120**.

A promotional code **901** is provided to a user of device **130**, for example and without limitation, on print media, such as on a cereal box. The use of device **130** sends the code server **903**. Server **903** then notifies server **805** to provide certain information to device **130**. Server **805** then provides device **130** with the requested information.

Device Routines

Device routines **114** may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server **140**; an expanding set of web services **250** available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector **1010** and content gateway **1130**.

Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform **110** SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless

extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some filed value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response. RSS/ATOM and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010,

database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can to presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, a as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image,

audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made at on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.
- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.
- 3: Whether the Application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values

31

required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requiring dynamic web page and object resizing, virtual screen resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.

32

5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.
- 3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "WebsiteName".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "WebsiteName".class, customized run time engine file, and the "WebsiteName".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "WebsiteName".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "WebsiteName".bat and "WebsiteName.lib".bat files are written. The "WebsiteName".bat and "WebsiteName".bat files are then executed under DOS, creating compressed "WebsiteName".cab and "WebsiteName.lib".cab files and compressed "WebsiteName" jar and "WebsiteName.lib" jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

Displaying Content on a Device
Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reas-

33

sembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform **110**.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device **130**. Specifically, device **130** operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device **130** for display on screen **137**.

Response Director

In one embodiment, system **100** includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. **12** illustrates one embodiment of a system **1200** that includes a response director **210**, a user agent database **1201**, an IP address database **1203**, and a file database **1205**. System **1200** is generally similar to system **100**, **200**, **800**, **900**, **1000**, or **1100**.

Databases **1201**, **1203**, and **1205** may reside on server **120**, **210**, or any computer system in communication with response director **210**. System **1200**, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database **1201** includes user agent information regarding individual devices **130** that are used to identify the operating system on the device. IP address database **1203** identifies the carrier/operator of each device **130**. File database **1205** includes data files that may operate on each device **130**.

The following is an illustrative example of the operation of response director **210**. First, a device **1300** generates an SMS message, which automatically sends an http://stream that includes handset information and its phone number to response director **210**. Response director **210** then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database **1201** which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address **1203** identifies the carrier/operator.

File database **1205** contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device **130**. A list of available Applications are returned to a decision tree, which then returns, to device **130**, the Application that is appropriate for the respective device.

34

For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director **210** generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of.

If there is an Application that has a data type that device **130** cannot support, for example, video, response director **210** sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director **210** is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algorithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform **110** automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128×128, 176×208, 240×260, 220×220, and many other customized sizes in between.

FIG. **13** is a schematic of an embodiment of a system **1300**. System **1300** is generally similar to system **1200**. System **1300** is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System **1300** includes response director **210**, a device characteristics operator and local database **1301**, a player profile database **1303** and a player build process **1305**, which may be authoring platform **110**.

As an example of system **1300**, when response director **210** receives an SMS message from device **130**, the response director identifies the device characteristics operator and locale from database **1301** and a Player URL from database **1303** and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director **210** by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting

35

hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform 110 may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a mag-

36

netic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases "in one embodiment," "in an embodiment," or "in certain embodiments" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term "comprising" shall be synonymous with "including," "containing," or "characterized by," is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. "Comprising" is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. "Comprising" leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be

37

interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:
computer memory storing a registry of:

- a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and
- b) the address of the web service;

an authoring tool configured to:

define a user interface (UI) object for presentation on the display, where said UI object corresponds to the web component included in said registry selected from the group consisting of an input of the web service and an output of the web service,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object,

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code, and

produce a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and

38

generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

define a SMS field or list; and

generate code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

10. The system of claim 1,

where said code includes three or more codes, where one of said three or more codes is device specific, and where two of said three or more codes is device independent.

11. The system of claim 1, where said code is provided over said network.

12. A method of displaying content on a display of a device utilizing a registry of one or more web components related to inputs and outputs of a web service obtainable over a network, where each web component includes a plurality of symbolic names of inputs and outputs associated with each web service, and where the registry includes: a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and

b) the address of the web service, said method comprising: defining a user interface (UI) object for presentation on the display, where said UI object corresponds to a web component included in said registry selected from the group consisting of an input of the web service and an output of the web service;

selecting a symbolic name from said web component corresponding to the defined UI object;

associating the selected symbolic name with the defined UI object;

producing an Application including the selected symbolic name of the defined UI object, where said Application is a device-dependent code; and

producing a Player, where said Player is a device-dependent code;

such that, when the Application and Player are provided to the device and executed on the device, and when a user of the device provides one or more input values associated with an input symbolic name to an input of defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for a display of the device to present an output value in the defined UI object.

13. The method of claim 12, where said registry includes definitions of input and output related to said web service.

14. The method of claim 12, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

15. The method of claim 12, where said UI object is an input field for a chat.

16. The method of claim 12, where said UI object is an input field for a web service.

17. The method of claim 12, where said UI object is an input field usable to obtain said web component, where said

39

input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

18. The method of claim 12, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

19. The method of claim 12, further comprising:

defining a phone field or list; and

generating code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

20. The method of claim 12, further comprising:

defining a SMS field or list; and

generating code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

21. Previously Presented The method of claim 12, and such that said Player interprets dynamically received, device-independent values of the web component defined in the Application.

22. The method of claim 12, further comprising:

providing said Application and Player over said network.

23. A method of providing information to a device having a display from a web component of a web service to a device on a network, said method comprising:

accepting, on the device, a first code over the network, where said first code is device-dependent;

accepting, on the device, a second code over the network, where said second code is device-independent and includes a plurality of symbolic names of inputs and outputs associated with the web service; and

executing said first code on the device,

where the symbolic names are provided from a registry of one or more web components related to inputs and outputs of a web service obtainable over a network,

40

where the web service requires both an input symbolic name and one or more associated input values and returns one or more output values having an associated output symbolic name, and

where the registry includes

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, and

b) the address of the web service;

where said executing includes:

processing said symbolic names of the second code on the device,

transmitting processed instructions from the device to the web service, and

accepting a third code on the device over the network, where said third code is a device-independent third code including the output of the web component provided by the web service over the network and in response to the second code.

24. The method of claim 23, where said third code is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

25. The method of claim 23, where said third code is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

26. The method of claim 23, where said first code and said second code are generated using an authoring tool.

27. The method of claim 23, where said first code is a Player.

28. The method of claim 23, where said second code is an Application which includes one or more web components.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 9,063,755 B2
APPLICATION NO. : 12/936395
DATED : June 23, 2015
INVENTOR(S) : Rempell et al.

Page 1 of 1

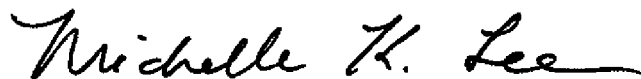
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page, item 73, Assignee: Express Mobile, inc., Novato, CA (US) - the "inc." should be -- Inc. --.

In the claims

Column 39, line 18, Claim 21, delete the text reading "Previously Presented".

Signed and Sealed this
Twentieth Day of October, 2015



Michelle K. Lee
Director of the United States Patent and Trademark Office

EXHIBIT D



US009471287B2

(12) **United States Patent**
Rempell et al.

(10) **Patent No.:** **US 9,471,287 B2**
(45) **Date of Patent:** ***Oct. 18, 2016**

(54) **SYSTEMS AND METHODS FOR
INTEGRATING WIDGETS ON MOBILE
DEVICES**

(71) Applicant: **Express Mobile, Inc.**, Novato, CA (US)

(72) Inventors: **Steven H. Rempell**, Novato, CA (US);
David Chrobak, Clayton, CA (US);
Ken Brown, San Martin, CA (US)

(73) Assignee: **Express Mobile, Inc.**, Novato, CA (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

This patent is subject to a terminal dis-
claimer.

(21) Appl. No.: **14/708,074**

(22) Filed: **May 8, 2015**

(65) **Prior Publication Data**

US 2015/0317130 A1 Nov. 5, 2015

Related U.S. Application Data

(63) Continuation of application No. 12/936,395, filed as
application No. PCT/US2009/039695 on Apr. 6,
2009, now Pat. No. 9,063,755.

(60) Provisional application No. 61/123,438, filed on Apr.
7, 2008, provisional application No. 61/113,471, filed
on Nov. 11, 2008, provisional application No.
61/166,651, filed on Apr. 3, 2009.

(51) **Int. Cl.**

G06F 3/048 (2013.01)
G06F 9/44 (2006.01)
H04L 29/08 (2006.01)
G06F 3/0484 (2013.01)
G06F 3/0482 (2013.01)
H04L 29/06 (2006.01)
H04L 12/58 (2006.01)

(52) **U.S. Cl.**

CPC **G06F 8/34** (2013.01); **G06F 3/0482**
(2013.01); **G06F 3/04842** (2013.01); **G06F**
9/4443 (2013.01); **H04L 51/046** (2013.01);
H04L 65/60 (2013.01); **H04L 67/02** (2013.01)

(58) **Field of Classification Search**

CPC **G06F 3/048**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0055017 A1 3/2004 Delpuch et al.
2004/0163020 A1 8/2004 Sidman
2004/0199614 A1* 10/2004 Shenfield H04L 29/06
709/220

(Continued)

OTHER PUBLICATIONS

Stina Nylander et al. "The Ubiquitous Interactor—Device Indepen-
dent Access to Mobile Services" (Computer-Aided Design for User
Interfaces IV, Proceedings of the Fifth International Conference on
Computer-Aided Design of User Interfaces CADUI'2004, Jan.
2004, pp. 271-282).*

(Continued)

Primary Examiner — Jennifer To

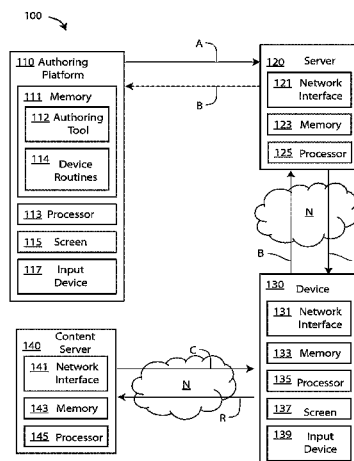
Assistant Examiner — Xuyang Xia

(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57) **ABSTRACT**

Embodiments of a system and method are described for
generating and distributing programming to mobile devices
over a network. Devices are provided with Players specific
to each device and Applications that are device independent.
Embodiments include a full-featured WYSIWYG authoring
environment, including the ability to bind web components
to objects.

28 Claims, 18 Drawing Sheets



(56)

References Cited

U.S. PATENT DOCUMENTS

2005/0149935 A1 7/2005 Benedetti
 2005/0273705 A1* 12/2005 McCain G06F 17/24
 2006/0063518 A1 3/2006 Paddon et al. 715/234

OTHER PUBLICATIONS

Stina Nylander et al. "The Ubiquitous Interactor—Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth international Conference on Computer-Aided Design of User Interfaces CADUT2004, Jan. 2004, pp. 271-282).

International Search Report and Written Opinion —PCT/US2009/039695—Aug. 21, 2009.

International Preliminary Report on Patentability and Written Opinion—PCT/US2009/039695—Oct. 21, 2010.

Rempell et al, co-pending U.S. Appl. No. 14/708,087, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,094, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,097, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,100, filed May 8, 2015.

Rempell et al, co-pending U.S. Appl. No. 14/708,108, filed May 8, 2015.

* cited by examiner

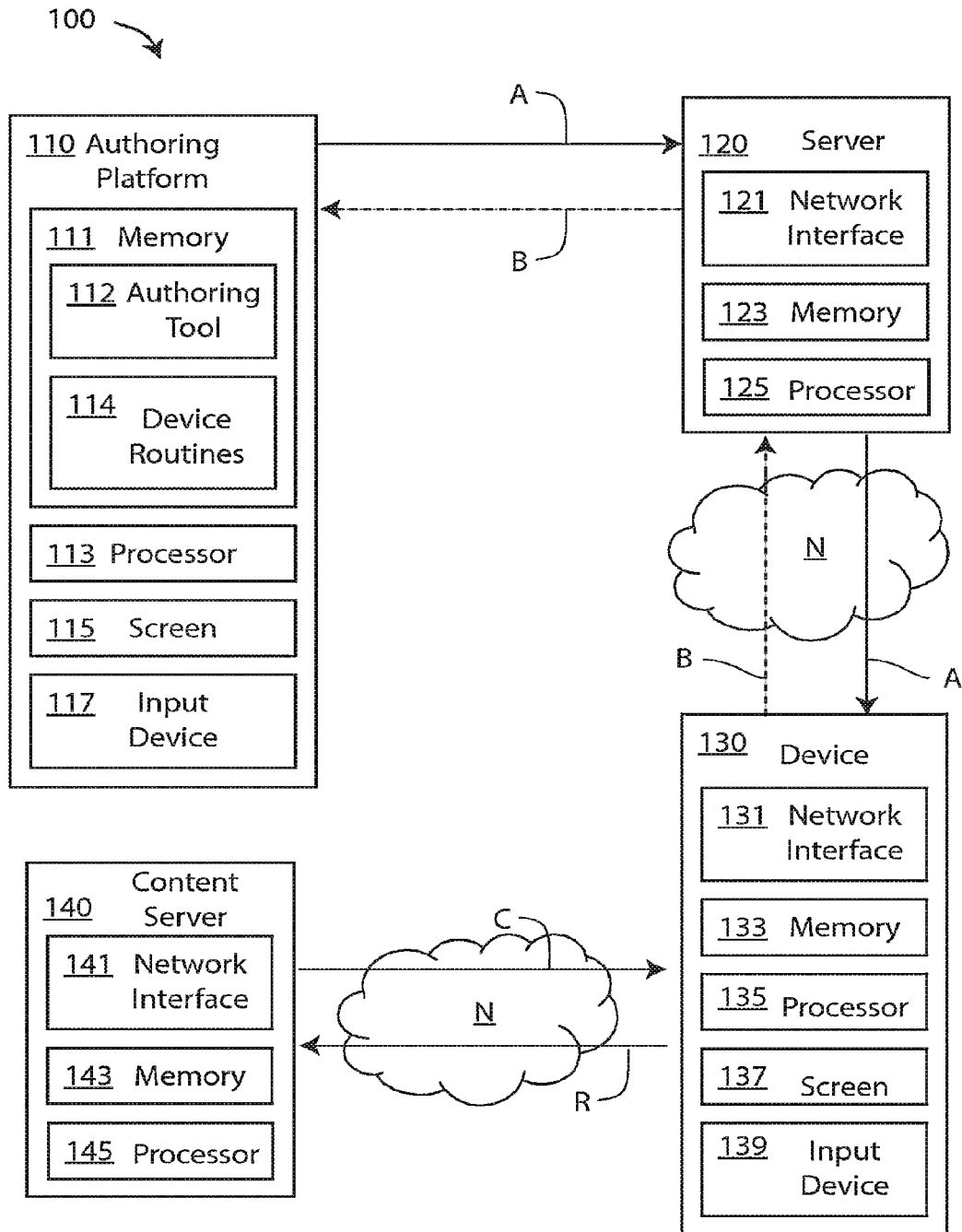


FIG. 1A

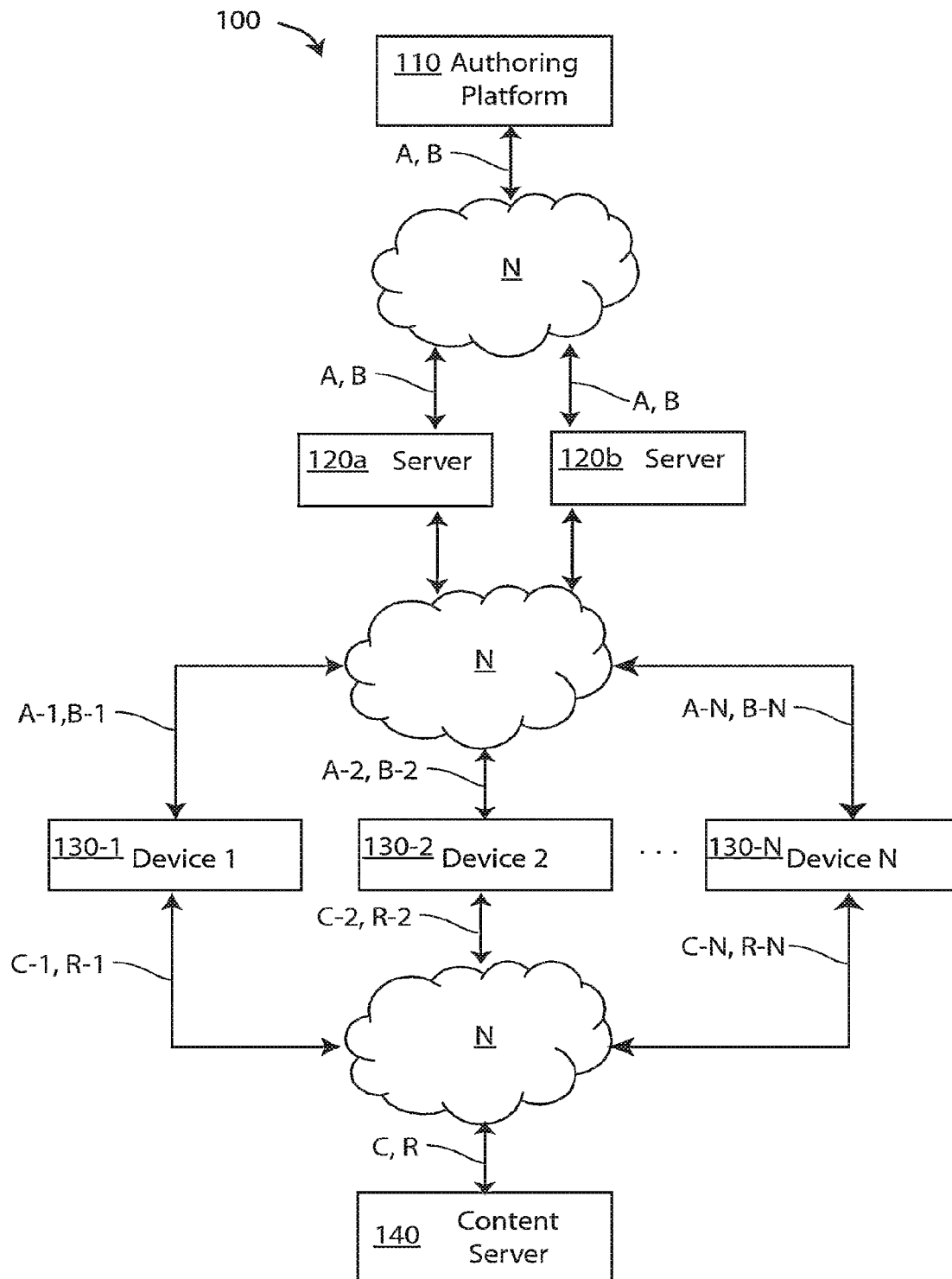


FIG. 1B

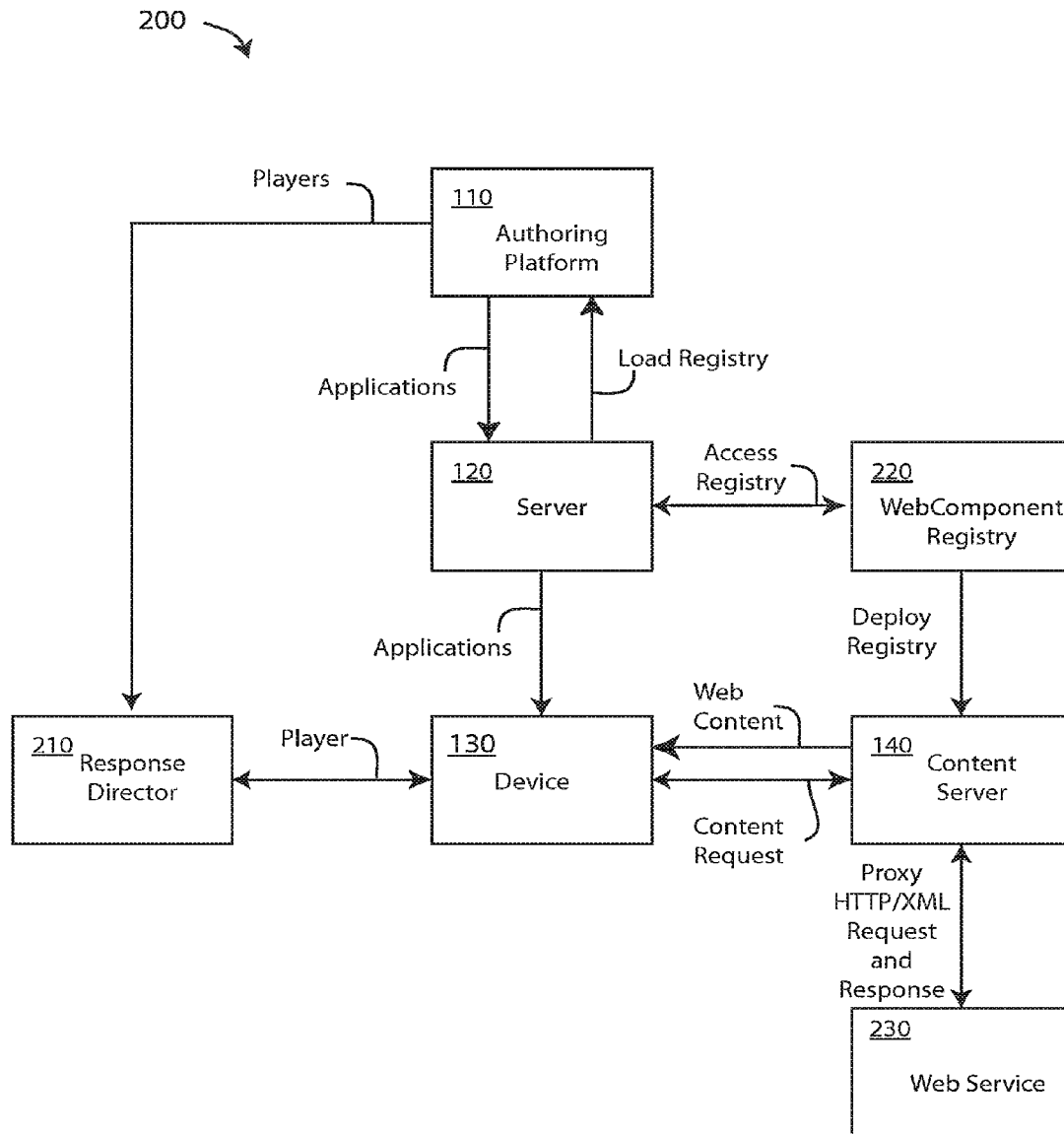


FIG. 2A

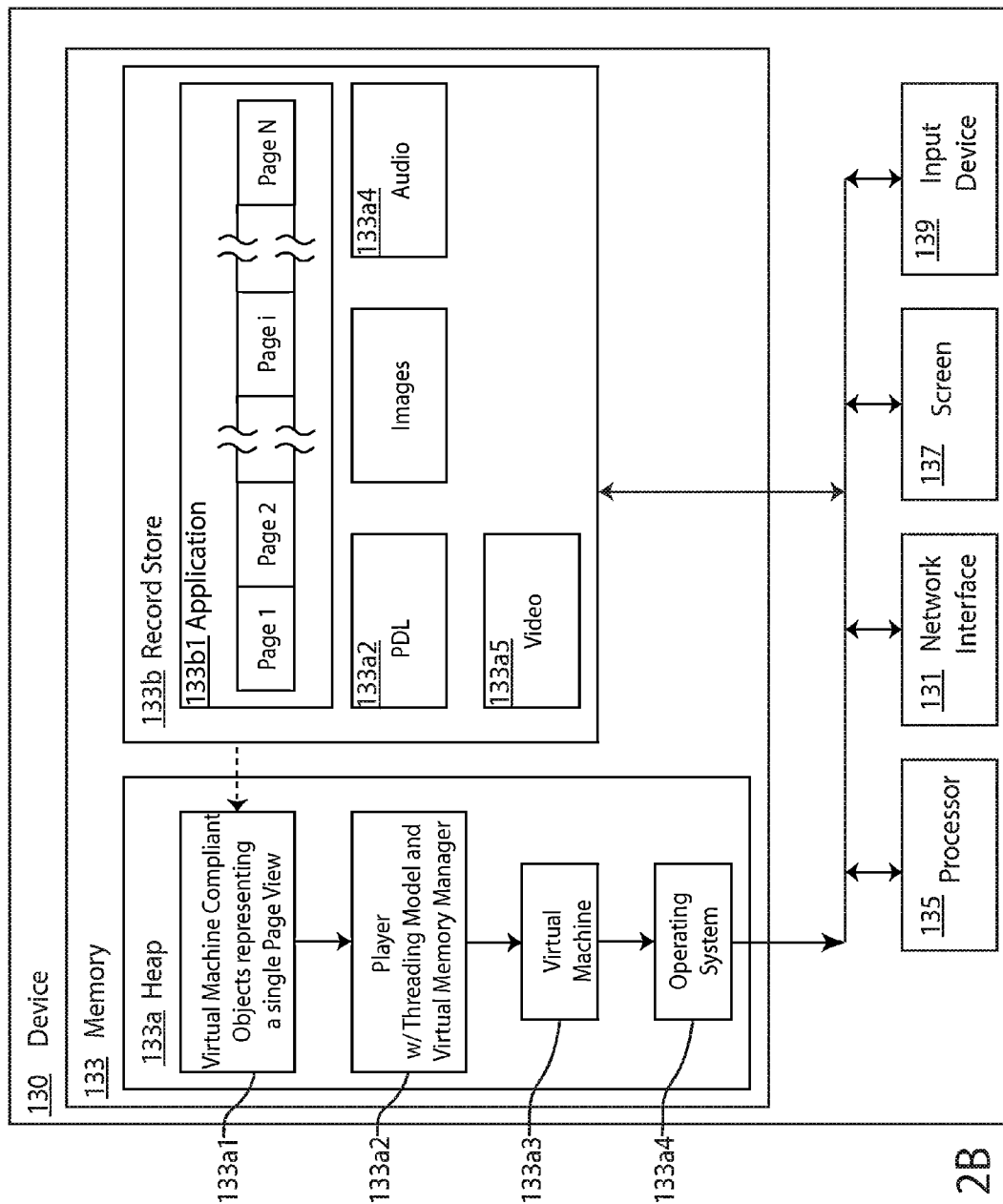
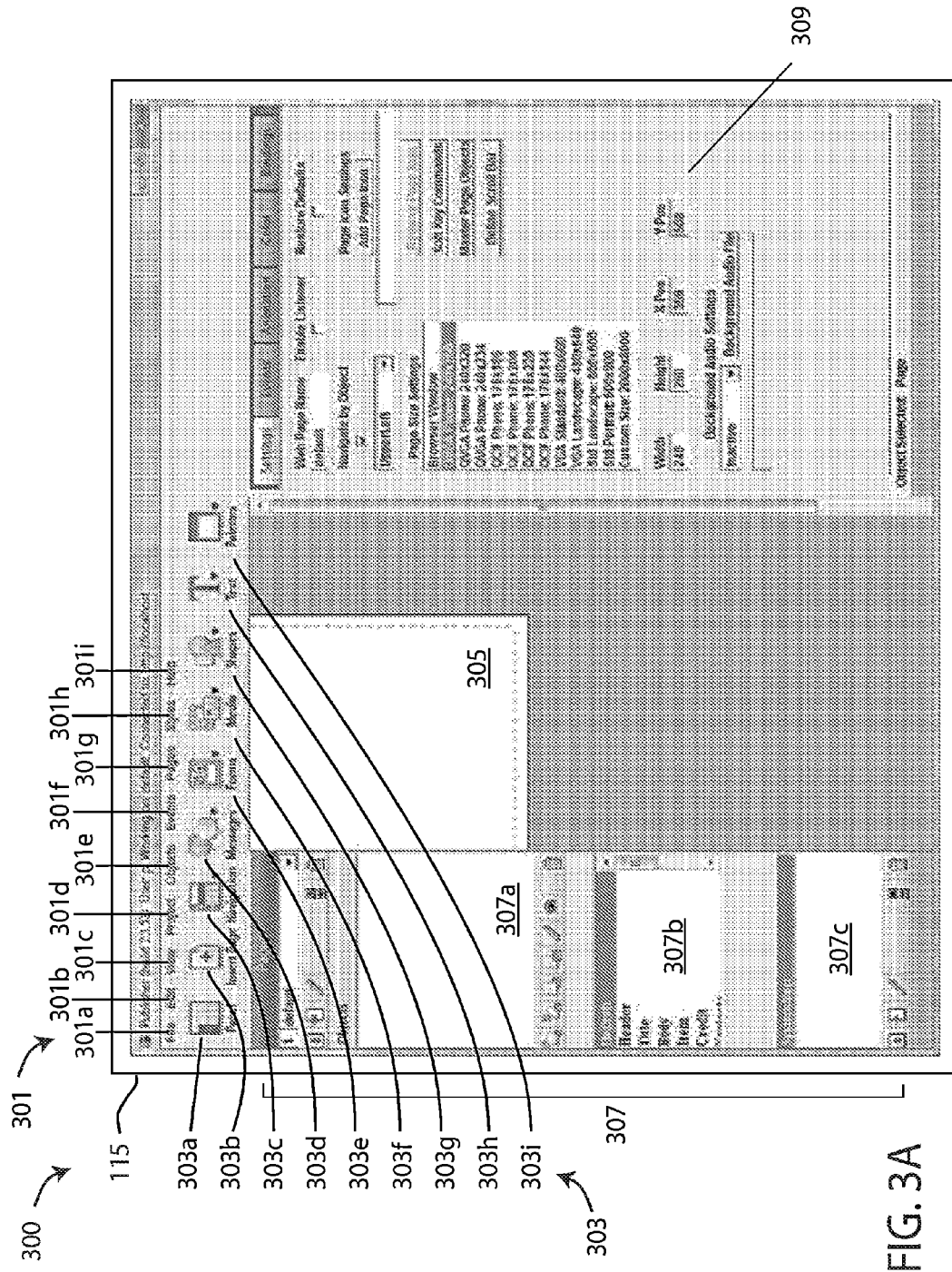


FIG. 2B



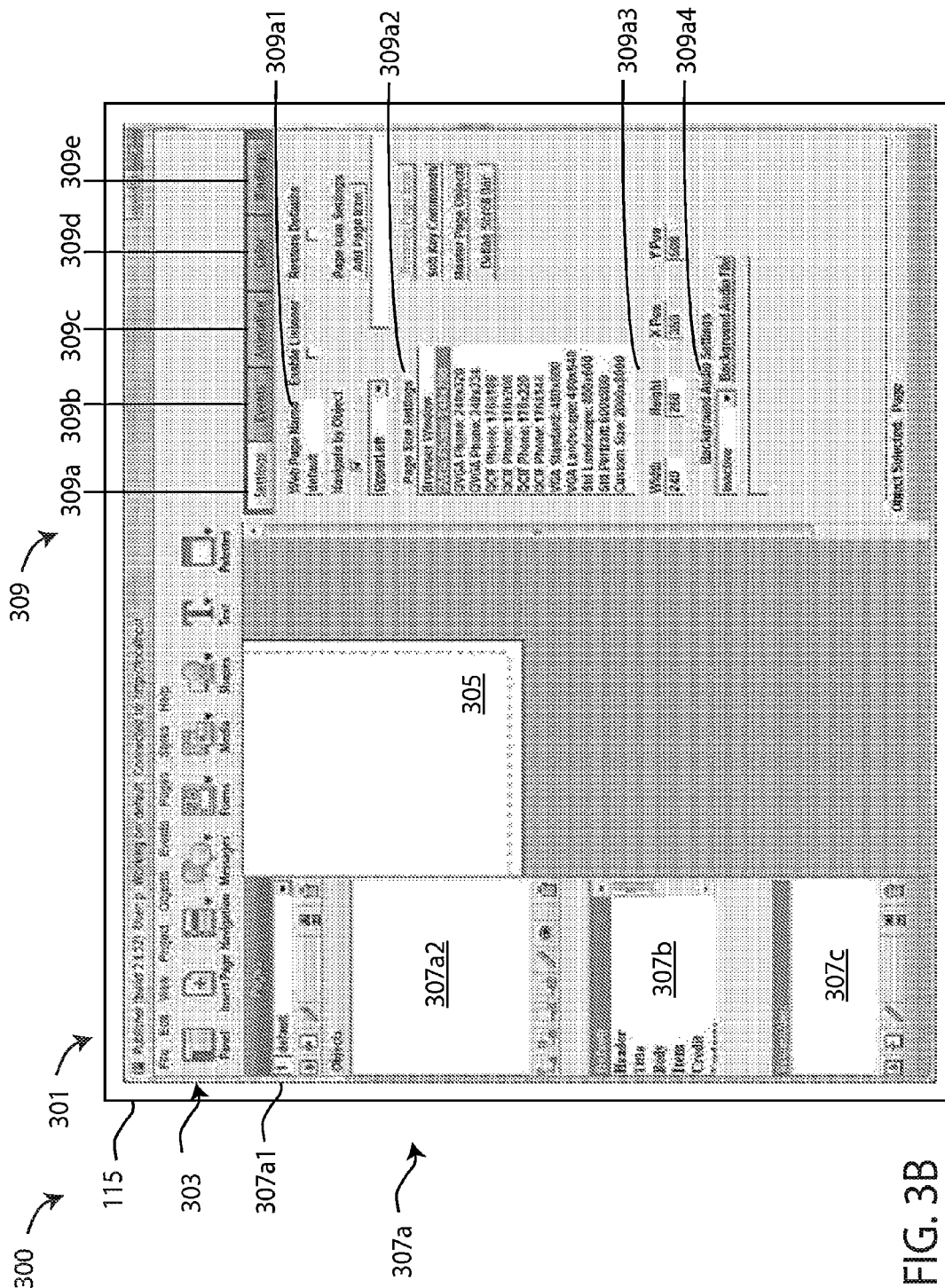


FIG. 3B

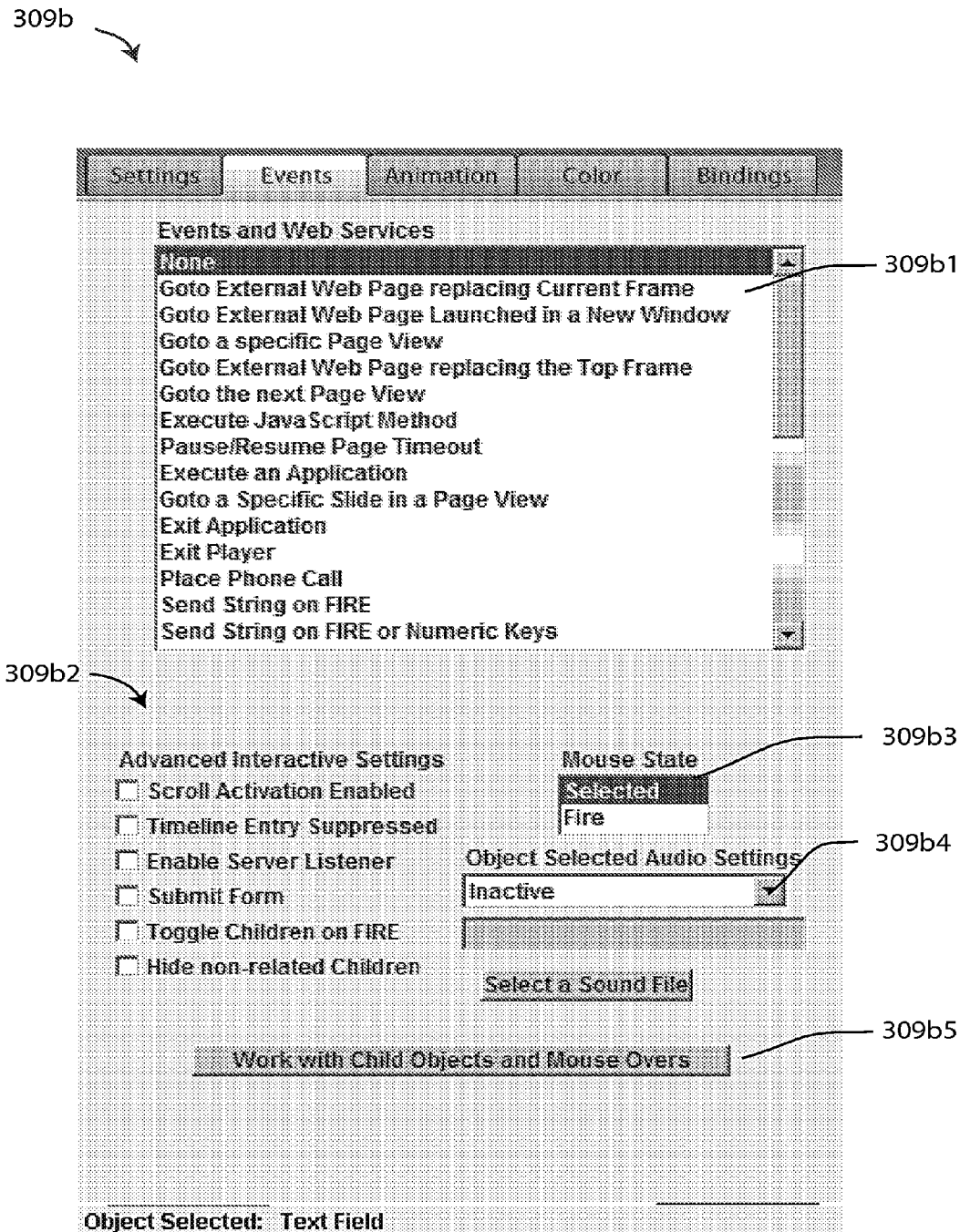


FIG. 3C

309c

Settings Events Animation Color Bindings

Object Entry Timeline Specifications

Activate Timeline Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

Object Animation Specifications

Delay (Sec)	Direction	Movement	Duration (Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	.1	2
2	Scroll Right	Fade In	.2	3
3	Custom	Fade Out	.3	4
4	Multi-Point		.4	5
5	Seek Cursor		.5	6
6	Attach		.6	7
7	Deposit		.7	8
8	Send Home		.8	9
9	Carom N		.9	10
10	Carom NE			11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles Custom Zoom % Avoid Cursor Dampen Anim.

1 0 ☒ ☐

Object Exit Timeline Specifications

Activate Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

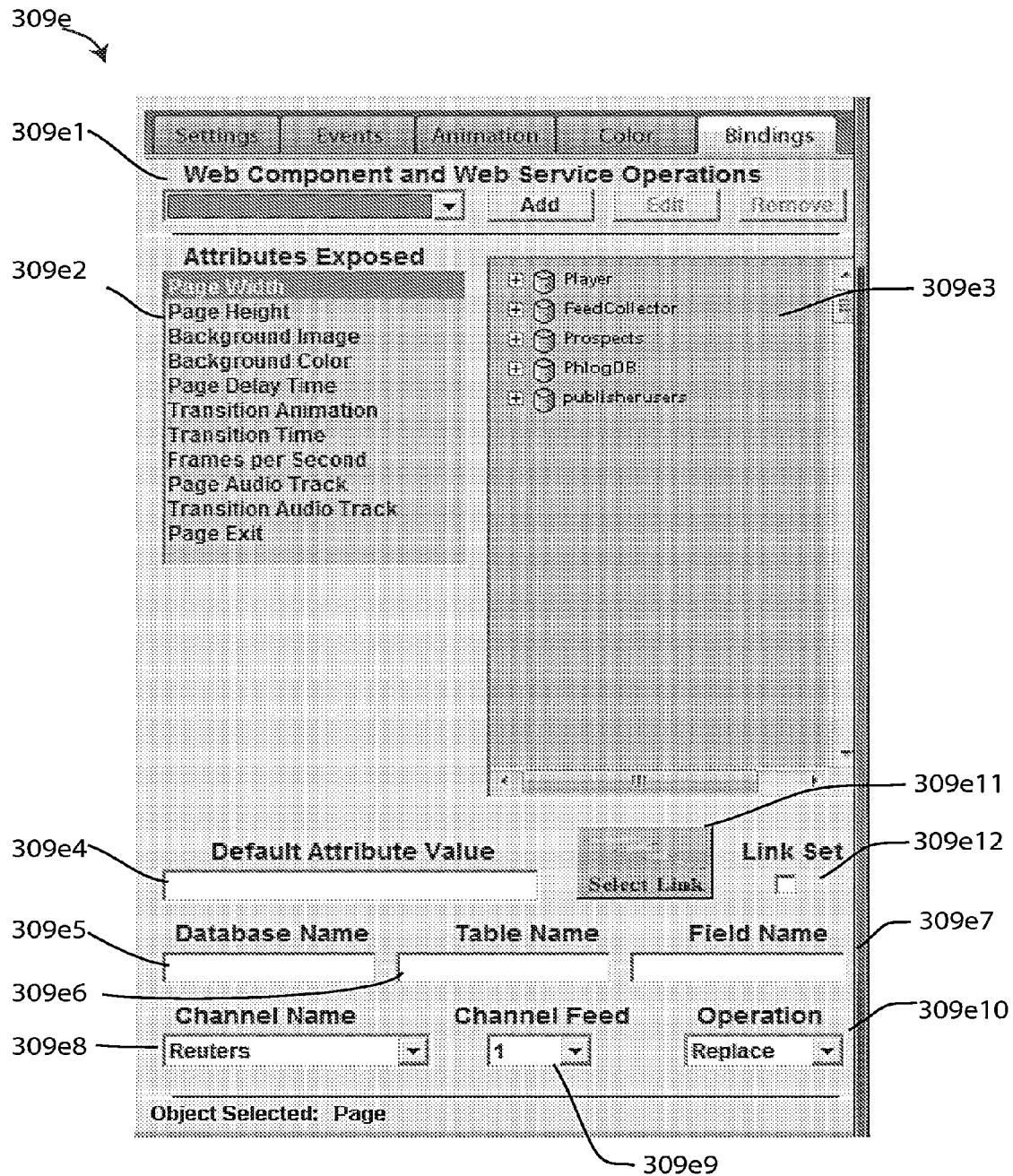


FIG. 3E

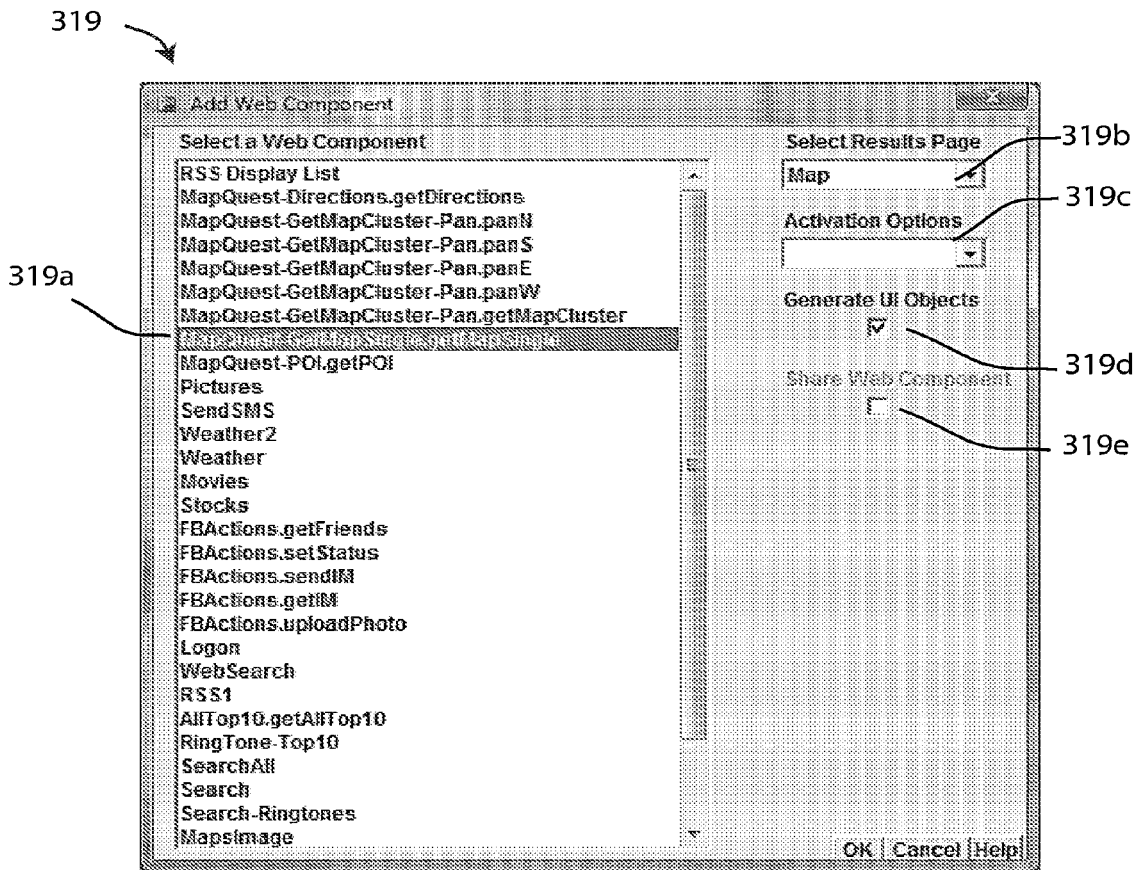


FIG. 3F

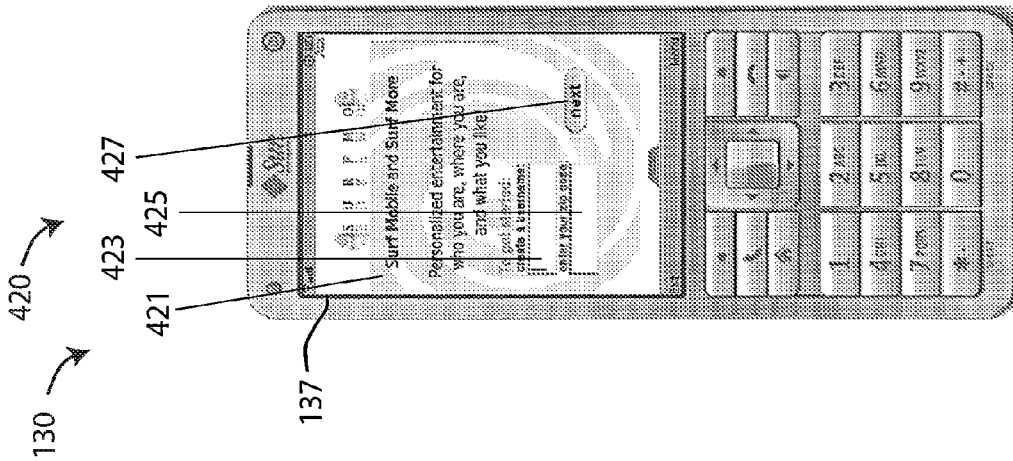


FIG. 4B

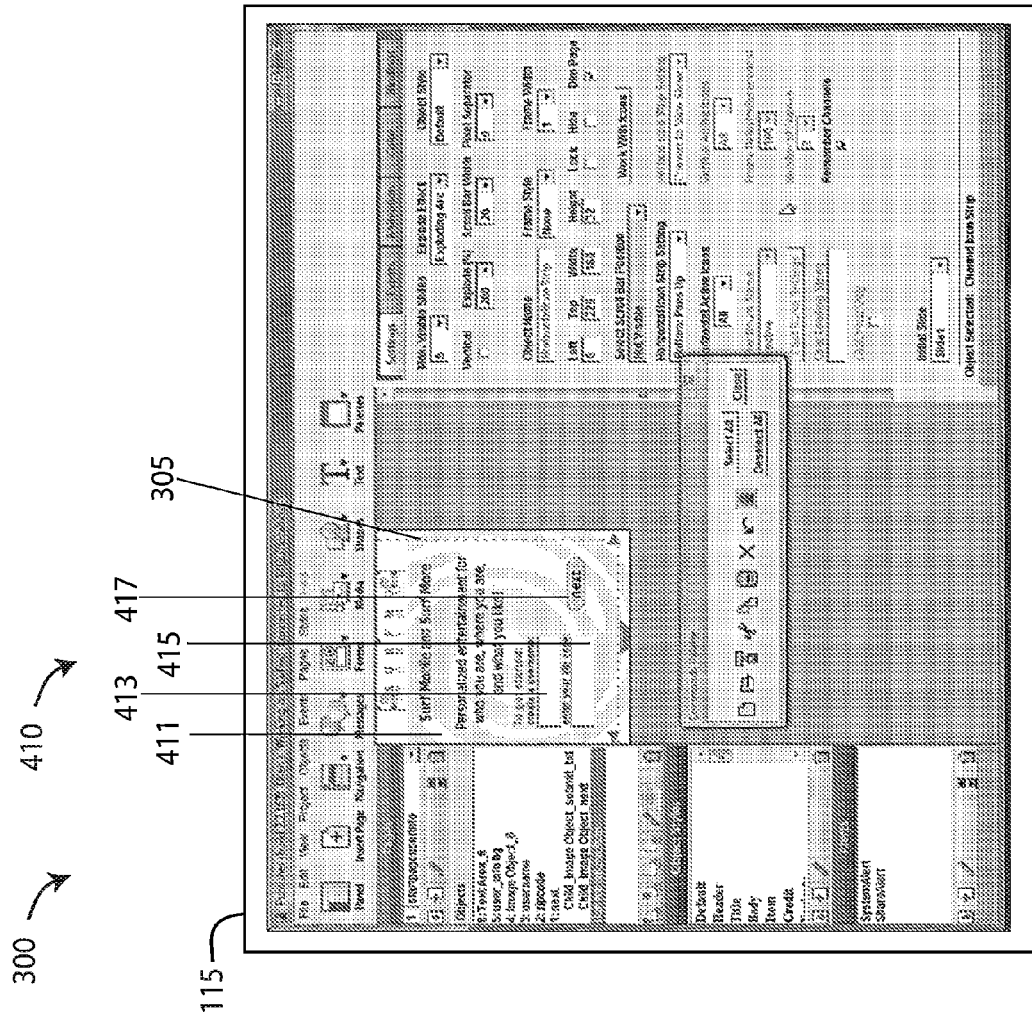


FIG. 4A

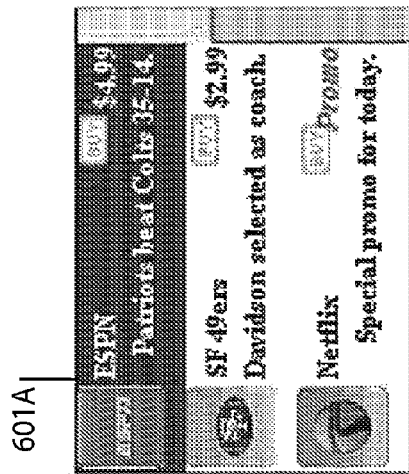


FIG. 6A

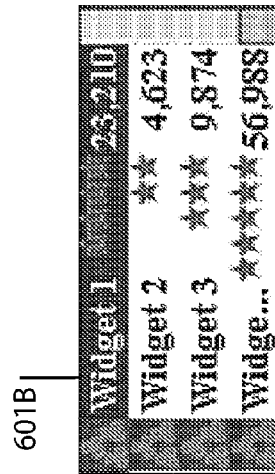


FIG. 6B

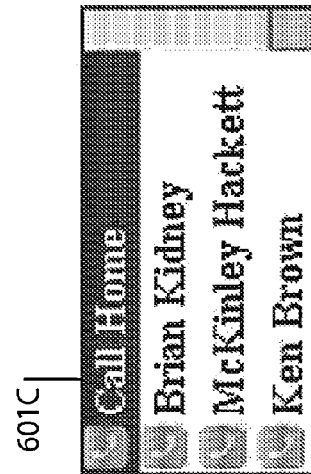


FIG. 6C

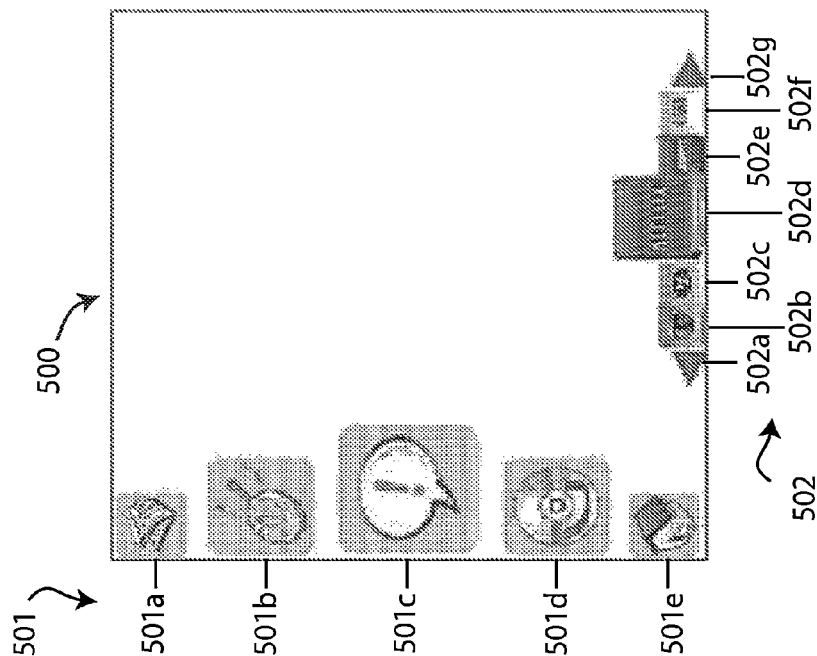


FIG. 5

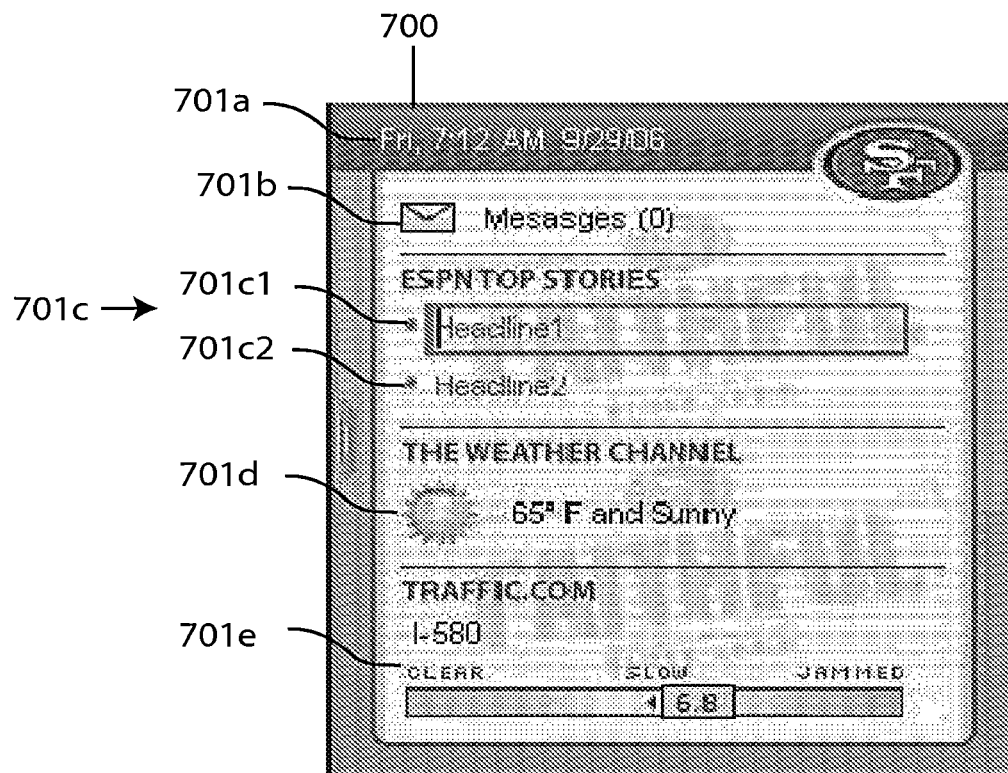


FIG. 7

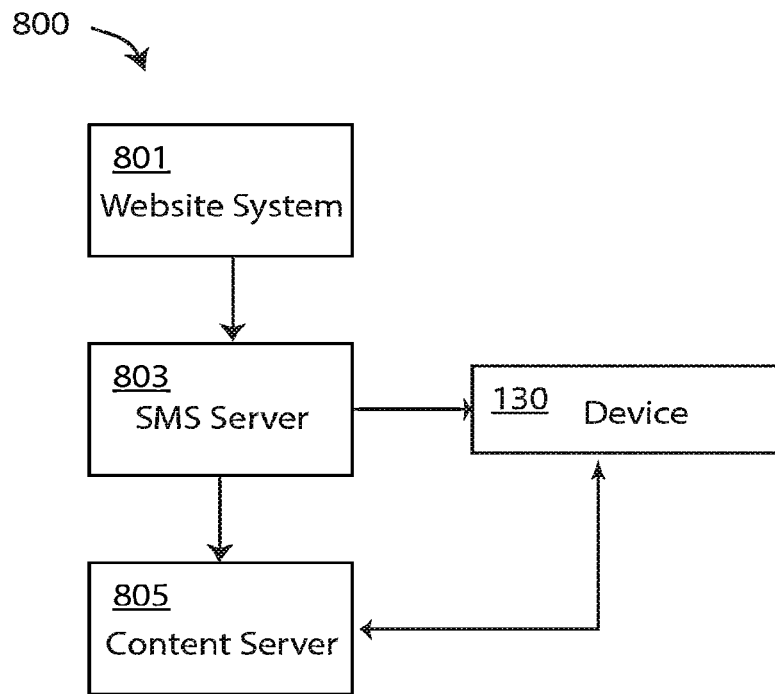


FIG. 8

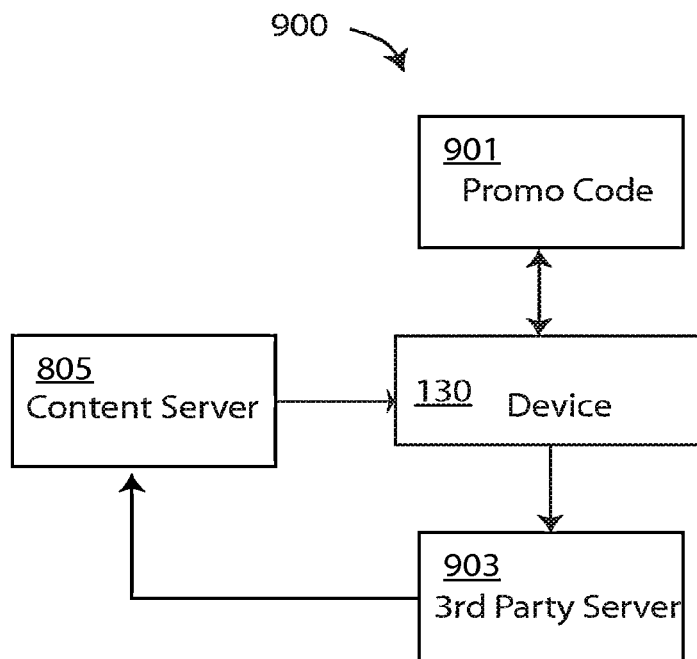


FIG. 9

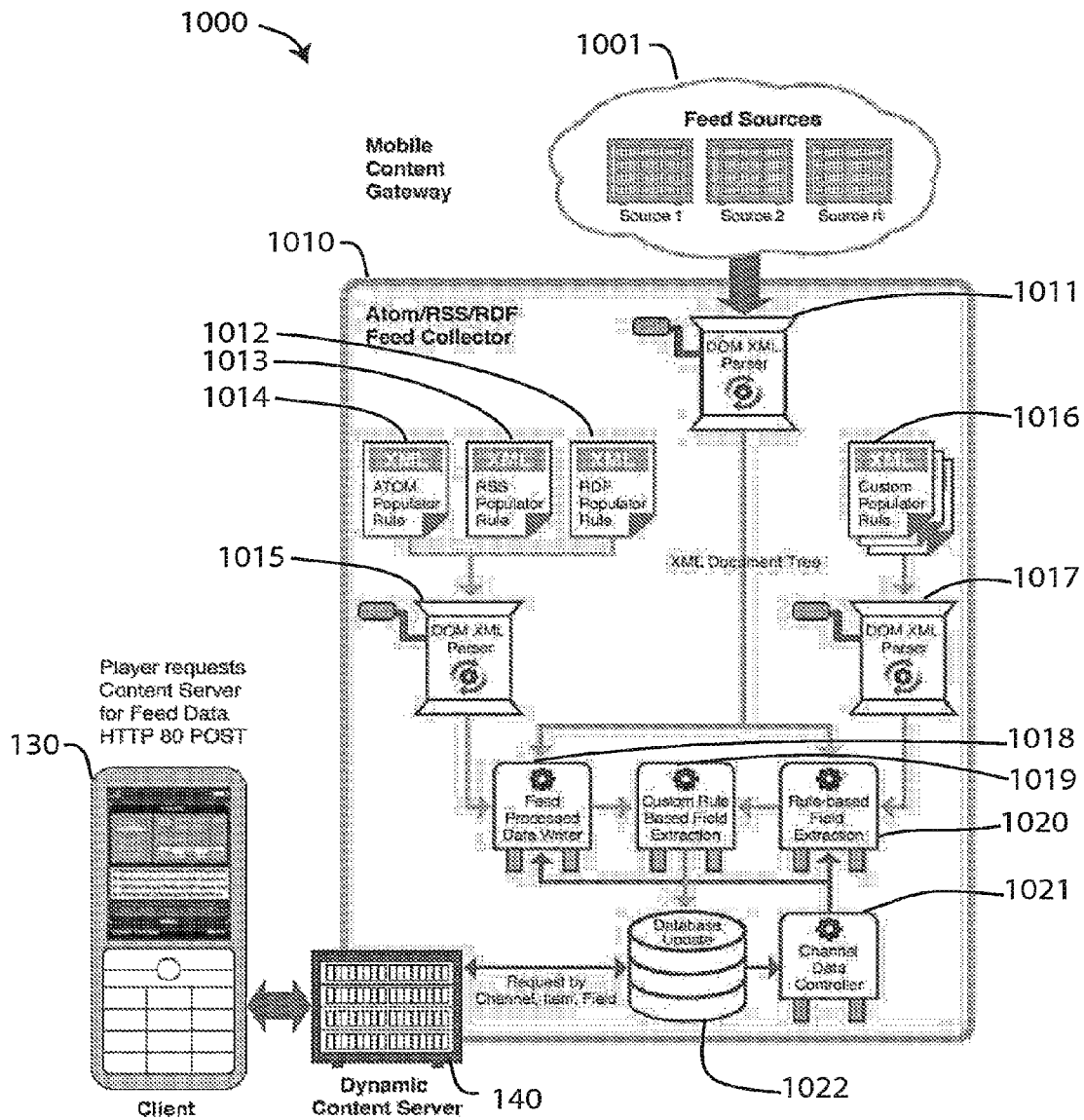
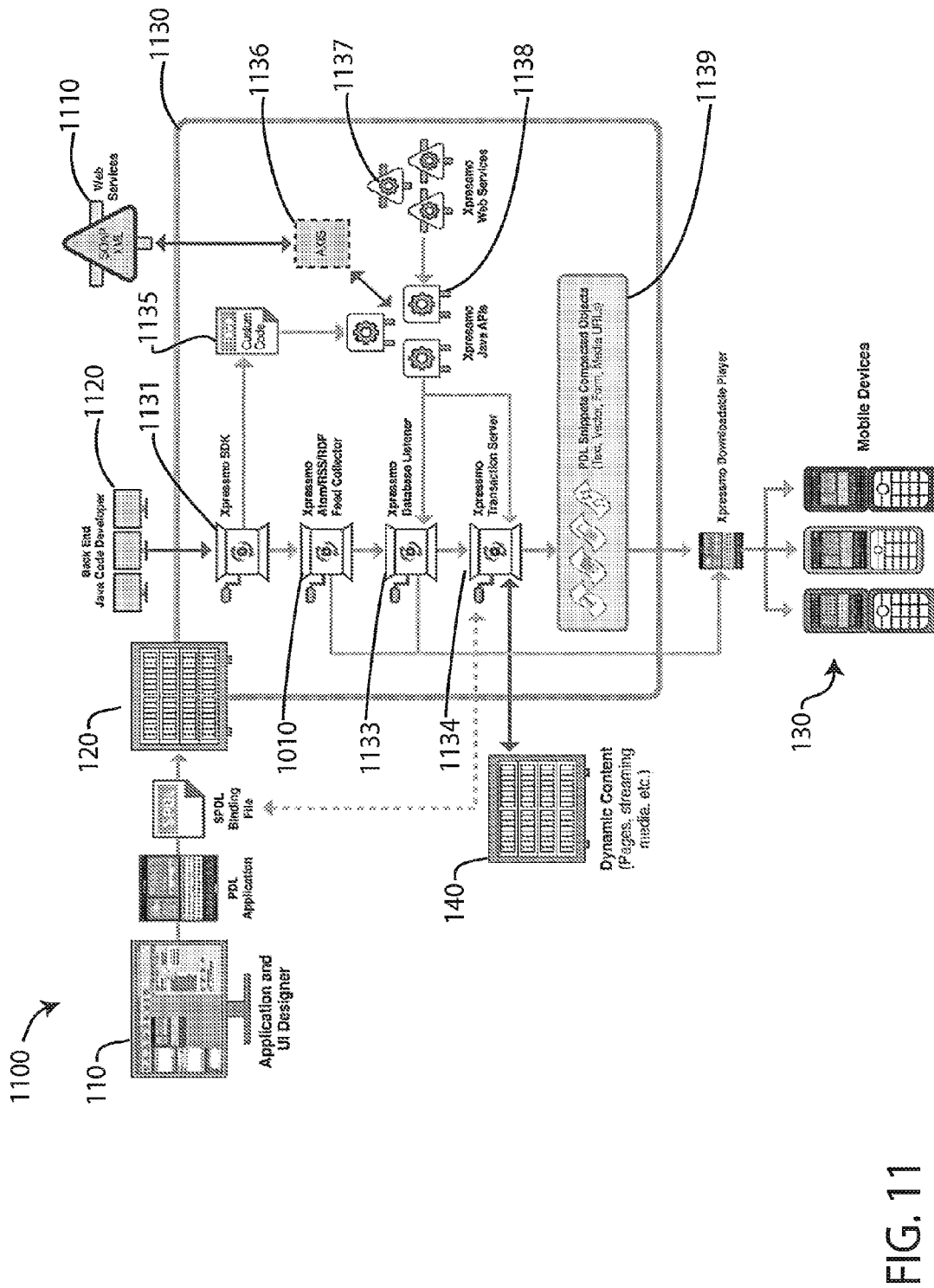


FIG. 10



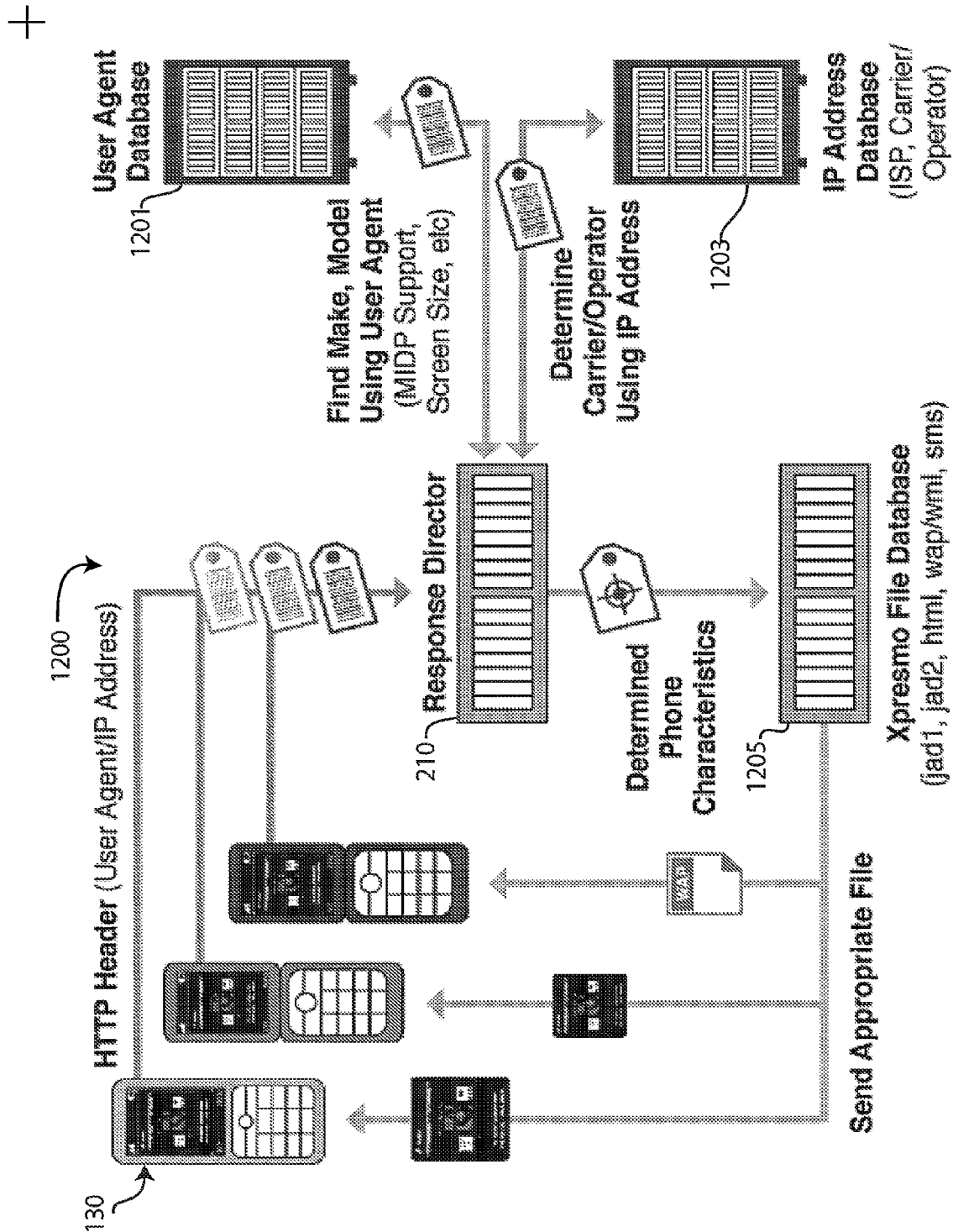


FIG. 12

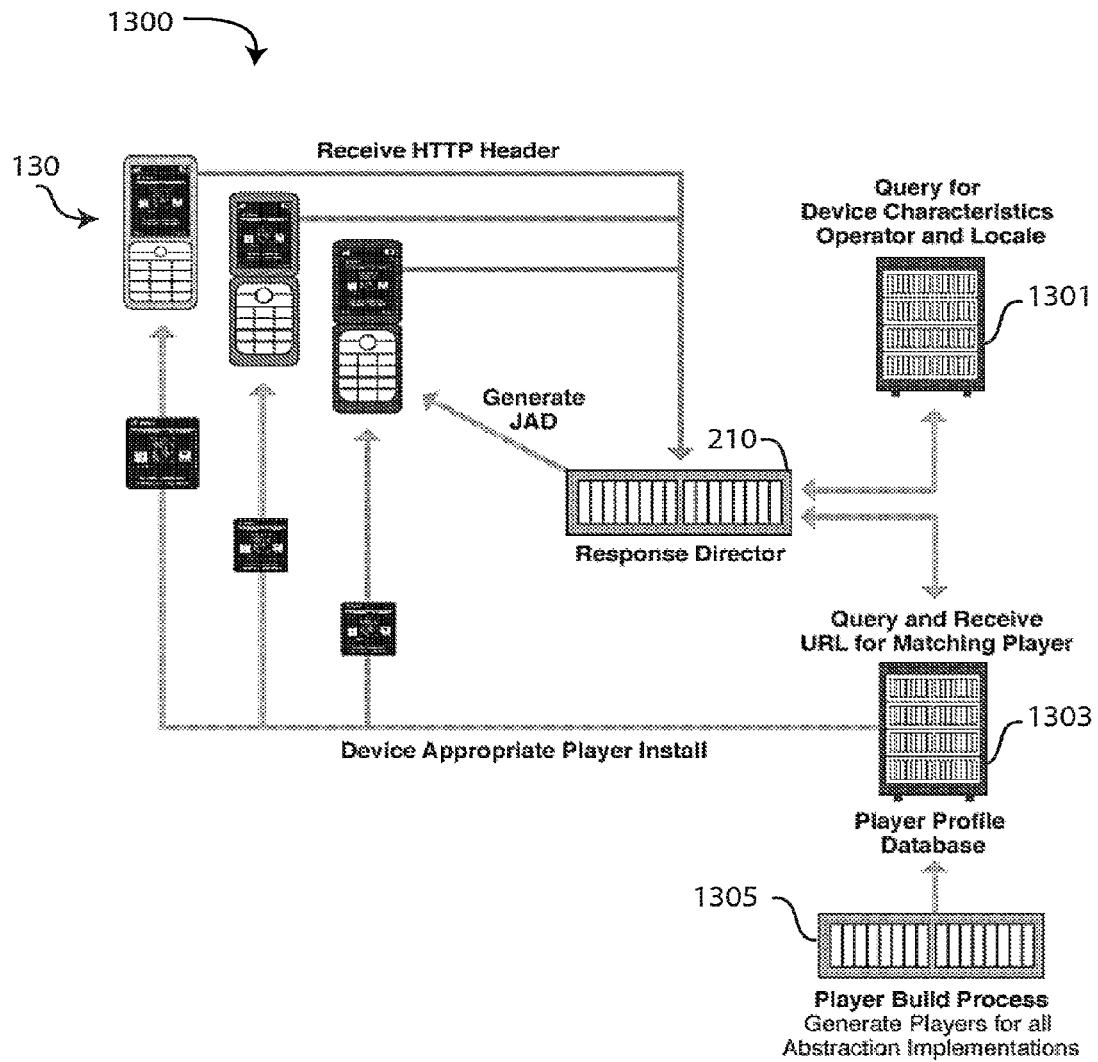


FIG. 13

1

SYSTEMS AND METHODS FOR INTEGRATING WIDGETS ON MOBILE DEVICES

TECHNICAL FIELD

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

BACKGROUND ART

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

DISCLOSURE OF INVENTION

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab' FIG. 3D illustrates one embodiment of an Animation Tab; FIG. 3E illustrates one embodiment of Bindings Tab; FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips; FIG. 6A is a display of a Channel Selection List; FIG. 6B is a display of a Widget Selection List; FIG. 6C is a display of a Phone List; FIG. 7 shows a display of a mash-up; FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

MODE(S) FOR CARRYING OUT THE INVENTION

FIG. 1A is an illustrative schematic of one embodiment of a system **100** including an authoring platform **110** and a server **120** for providing programming instructions to a

3

device 130 over a network N. In one embodiment, device 130 is a wireless device, and network N includes wireless communication to the device. Alternatively, system 100 may provide access over network N to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform 110 may produce programming instructions or files that may be transmitted over network N to operate device 130, including instructions or files that are sent to device 130 and/or server 120. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices 130 and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network N, and that, in particular, the location is not limited to any particular server, for example.

Device 130 may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface 131, a memory 133, a processor 135, a screen 137, and an input device 139. Network interface 131 is used by device 130 to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory 133 includes programming required to operate device 130 (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface 131, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen 137 is a touch screen, providing the functions of the screen and input device 139.

Authoring platform 110 includes a computer or computer system having a memory 111, a processor 113, a screen 115, and an input device 117. It is to be understood that memory 111, processor 113, screen 115, and input device 117 are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory 111 may also instruct authoring platform 110 to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory 111 is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool 112. In one embodiment, authoring tool 112 is a graphical system for designing the layout of features as a display that is to appear on screen 137. One example of authoring tool 112 is the CDER™ publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device 130 may include an operating system having a platform that can interpret certain routines. Memory 111 may optionally include programming referred to herein, and without limitation, as routines 114 that are executable on device 130.

Routines 114 may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices 130, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for

4

devices using routines provided on a platform. Thus as an example, routines 114 may include Java API's and an authoring tool System Development Kit (SDK) for specific devices 130.

Server 120 is a computer or computer system that includes a network interface 121, a memory 123, and a processor 125. Is to be understood that network interface 121, memory 123, and processor 125 are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform 110; accept input and/or provide output through network interface 121 over network N to network interface 131; or store information from authoring platform 110 or from device 130 for transmission to another device or system at a later time.

In one embodiment, authoring platform 110 permits a user to design desired displays for screen 137 and actions of device 130. In other words, authoring platform 110 is used to program the operation of device 130. In another embodiment, authoring platform 110 allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory 123 and may then be sent, when requested by device 130 or when the device is otherwise accessible, over network N, through network interface 130 for storage in memory 133.

In an alternative embodiment, analytics information from devices 130 may be returned from device 130, through network N and server 120, back to authoring platform 110, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server 140 is a computer or computer system that includes a network interface 141, a memory 143, and a processor 145. It is to be understood that network interface 141, memory 143, and processor 145 are configured such that a stored program in the memory may be executed by the processor to accepts requests R from device 130 and provide content C over a network, such as web server content the Internet, to device 130.

FIG. 1B is schematic of an alternative embodiment system 100 for providing programming instructions to device 130 over a network N that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system 100 may include multiple servers 120 and/or multiple devices 130.

In the embodiment of FIG. 1B, system 100 is shown as including two or more servers 120, shown illustratively and without limitation as servers 120a and 120b. Thus some of the programming or information between authoring platform 110 and one or more devices 130 may be stored, routed, updated, or controlled by more than one server 120. In particular, the systems and methods described herein may be executed on one or more server 120.

Also shown in FIG. 1B are a plurality of devices 130, shown illustratively and without limitation as device 130-1, 130-1, . . . 130-N. System 100 may thus direct communication between individual server(s) 120 and specific device(s) 130.

As described subsequently, individual devices 130 may be provided with program instructions which may be stored in each device's memory 133 and where the instructions are

5

executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different "device platforms." Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices 130 are provided with some programming from authoring system 100 that is particular to the device.

In one embodiment, system 100 provides permits a user of authoring platform 110 to provide instructions to each of the plurality of devices 130 in the form of a device- or device-platform specific instructions for processor 135 of the device, referred to herein and without limitation as a "Player," and a device-independent program, referred to herein and without limitation as an "Application." Thus, for example, authoring platform 110 may be used to generate programming for a plurality of devices 130 having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device 130 utilizes a Player and an Application to execute programming from authoring platform 110. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device 130 ("activated") through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device 130. In yet another alternative embodiment, Player is activated through a signal to device 130 by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory 133, the functioning of device 130 may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory 133 and which, when executed by processor 135, generate the designed displays on screen 137. The Application and Player may also include programming instructions which may be stored in memory 133 and which provide instructions to processor 135 to accept input from input device 139.

Authoring tool 112 may, for example, produce and store within memory 111 a plurality of Players (for different devices 130) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers 120 and then provided to individual devices 130. In general, Applications are provided to device 130 for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device 130.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device 130. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen 137, and the Player may interpret the Java and instruct processor 135 to produce the display according to the Application for execu-

6

tion on a specific device 130 according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen 137, instructions for accepting input from input device 139, instructions for interacting with a user of device 130, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device's Player interprets or executes the Application to generate one or more "pages" ("Applications Pages") on a display as defined by the PDL. The Player may include code that is device-specific—that is, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool 112 may thus be used to design one or more device-independent Applications and may also include information on one or more different devices 130 that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system 100 provides Players and Applications to one server 120, as in FIG. 1A. In another embodiment, system 100 provides Players to a first server 120a and Applications to a second server 120b, as in FIG. 1B.

In one embodiment, authoring tool 112 may be used to program a plurality of different devices 130, and routines 114 may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a "thin client"—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform 110 allows user to arrange objects for display on screen. A graphical user interface ("GUI," or "UI") is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform 110 also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform 110 also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor 135 to store or modify information in memory 133, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool 112, the Application, and Player. Thus while either designing the Application with the authoring tool 112, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into

7

a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in co-pending U.S. Pat. No. 6,546,397 to Rempell ("Rempell"), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the

8

display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server 120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring

platform 110 which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service 230 to a GUI component of the Application as will be displayed on screen 137. In addition, multiple components of one or more web service 230 can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service 230, are stored in the PDL. The content server 140 handles all communication between device 130 and the web service 230 and can be automatically deployed as a web application archive to any content server.

Device 130, upon detecting an event in which a component of a web service 230 has been defined, assembles and sends all related inputs to content server 240, which proxies the request to web service 230 and returns the requested information to device 130. The Player on device 130 then takes the outputs of web service 230 and binds the data to the UI components in the Application, as displayed on screen 137.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform 110 wants to present an ATOM feed on device 130, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server 120 and processed by device 130 at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services 230 are available either to the user of authoring platform 110 or otherwise accessible through the SDK and Java APIs of routines 114. System 200 permits an expanding set of components of web services 230 including, but not limited to: server pages from content server 120; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services 230 defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System 200 also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen 137. Thus, for example, a user of authoring platform 110 may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device 130, using system 200. Authoring platform 110 may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices 130. In addition, Authoring platform 110 may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services 230 that may be provided using system 200, a user of authoring platform 110 can place, for example, Yahoo maps into device 130 by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform 110. System 200 also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device 130 includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform 110 having access to platform dependent routines 114, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform 110 will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device 130, much like how a web browser downloads an HTML page through a external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services 230 include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each

11

with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform 110, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined Icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device 130 illustrating an embodiment of the programming generated by authoring platform 110. Memory 133 may include several different logical portions, such as a heap 133a, a record store 133b and a filesystem (not shown).

As shown in FIG. 2B, heap 133a and record store 133b include programming and/or content. In general, heap 133a is readily accessible by processor 135 and includes, but is not limited to portions that include the following programming: a portion 133a1 for virtual machine compliant objects representing a single Page View for screen 137; a portion 133a2 for a Player; a portion 133a3 for a virtual machine; and a portion 133a4 for an operating system.

Record store 133b (or alternatively the filesystem) includes, but is not limited to, portions 133b1 for Applications and non-streaming content, which may include portions 133a2 for images, portions 133a4 for audio, and/or portions 133a5 for video, and portions 133b2 for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlayed on the current page view. Other content, such as streaming content may be provided from network interface 131 directly to the Media Codec of device 130 with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion 133a1 as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieve from the Record store, converted into virtual machine compliant objects (by processor 135 and according to operation by the Player, Virtual Machine, etc), and stored in heap 133a. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions 133a1, 133a2, 133a3, 133a4, record store 133b and processor 135 are illustrative of the logical connection between the different types of programming stored in Heap 133a and record store 133b, that is, how data is processed by processor 135.

The Player determines which of the plurality of Application Pages in portion 133b1 is required next. This may be determined by input actions from the Input Device 139, or from instructions from the current Application Page. The Player instructs processor 135 to extract the PDF from that Applications Page and store it in portion 133a1. The Player then interprets the Application Page extracted from PDL

12

which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions 133a3, 133a4, 133a5, respectively.

The Virtual Machine in portion 133a3 processes the Player output, the Operating System in portion 133a3 processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion 133a4.

In another embodiment, the Player is a native program that interacts directly with the operating system.

Embodiments of a Publishing Environment

In one embodiment, authoring platform 110 includes a full-featured authoring tool 112 that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool 112 permits a user to design an Application by placing objects on canvas 305 and optionally assigning actions to the objects and save the Application. System 100 then provides the Application and Player to a device 130. The Application as it runs on device 130 has the same look and operation as designed on authoring platform 110. In certain embodiments, authoring platform 110 is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform 110 produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device 130 may, according to its contents, modify what is displayed on screen 137 or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool 112 allows a user to place one or more objects on canvas 305 and associate the objects with an Applications Pages. Authoring platform 110 maintains a database of object data in memory 111, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool 112, which are also maintained in memory 111. Authoring tool 112 also allows a user to define more than one Applications Page.

In another embodiment, authoring tool 112, provides Java programming functions of the Java API for specific devices 130 as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform 110 to position objects that, after being provided as an Application to device 130, activate such Java functions on the device.

In certain embodiments, authoring platform 110, as part of system 100, permits designers to include features of advanced web and web services Applications for access by users of device 130. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform 110 through interactive object libraries.

In certain embodiments, authoring platform 110, as part of system 100, allows the inclusion of child objects which may eventually be activated on device 130 by the user of the device or by time. The uses of the child objects on device 130 include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

13

In certain other embodiments, authoring platform 110, as part of system 100, provides advanced interactive event models on device 130, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform 110 may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device 130.

In certain embodiments, authoring platform 110, as part of system 100, provides full style inheritance on device 130. Thus, for example and without limitation, both master page inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool 112 may create various text objects on canvas 305 using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool 112 changes the font color of a specific button to green. If later, the user of authoring tool 112 changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform 110 provides page view, style, object, widget and Application template libraries. Authoring platform 110 may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform 110 to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours as seen on device 130.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface 300 as it appears, for example and without limitation, on screen 115 while executing authoring tool 112. In one embodiment, publisher interface 300 includes a Menu bar 301, a Tool bar 303, a Canvas 305, a Layer Inspector 307 having subcomponents of a page/object panel 307a, an object style panel 307b, and a page alert panel 307c, and a Resource Inspector 309.

In general, publisher interface 300 permits a user of authoring platform 110 to place objects on canvas 305 and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface 300 permits a user to program a graphical interface for the screen 137 of device 130 on screen 115 of

14

authoring platform 110, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device 130 when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool 112 maintains, in memory 111, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool 112 further maintains, in memory 111, a listing of the objects. As the user selects objects, publisher interface 300 provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory 111.

In one embodiment, publisher interface 300 is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas 305 placed and manipulated by the various commands within publisher interface 300 or inputs such as an input device 117 which may be a keyboard or mouse. As described herein, the contents of canvas 305 may be saved as an Application that, through system 100, provide the same or a similar placement of objects on screen 137 and have actions defined within publisher interface 300. Objects placed on canvas 305 are intended for interaction with user of device 130 and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface 300 may assign or associate actions or web bindings to UI objects placed on canvas 305 with result in the programming device 130 that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items)).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool 112, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select)	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select) Text Button
String	Text Field (Alpha)	Any	Text Field (Alpha)	Any
multilineString	Text Area	Text Area	Text Area	Text Area Paragraph
ImageURL	N/A	N/A	Image	Image Slide Show
VideoURL	N/A	N/A	Video	Video Slide Show
List	Single Item List	Single Item List Multi-Select List Complex List Choice Slide Show	Single Item List	Any List Type Any Choice Type (see Complex List Specification)
ComplexList	Complex List	Single Item List Multi-Select List Complex List	Single Item List	Any List Type (see Complex List Specification)
Slideshow	Slide Show	Slide Show	Slide Show	Slide Show
SearchResponseList	N/A	N/A	Search Response List	Search Response List Control List Complex List Choice RSS Display List Control List Complex List Choice Choice Complex List
RSSList	N/A	N/A	RSS Display List	RSS Display List Control List Complex List Choice Choice Complex List
SingleSelectionList	Choice	Choice Complex List	Choice	Choice Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image Video Slide Show
ChannelDescription	N/A	N/A	Text Area	Text Area Paragraph Text Field Text Button List Choice Text Field Text Button Paragraph Text Area List Choice
ChannelTitle	N/A	N/A	Text Field	Text Field Text Button Paragraph Text Area List Choice Text Field (URL request)
URL			Text Field (URL request)	Text Field (URL request)
Audio URL			Text Field (Audio URL request)	Text Field (Audio URL request)
Purchase URL			Text Field (Purchase URL request)	Text Field (Purchase URL request)
Image Data			Image	Image Slide Show
Image List Data			Slide Show	Slide Show Image
Persistent Variable	N/A	N/A	N/A	N/A
Pipeline Multiple Select	Multi-select List	Multi-select List Complex List Slide Show	N/A	N/A
Phone Number	Text Field (numeric type)	Text Field Text Button	Text Field (numeric type)	Text Field Text Button
Hidden Attribute	Complex List	Complex List Slide Show	Complex List	Complex List Slide Show
Collection List	N/A	N/A	Slide Show	Complex List Slide Show

17

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated patent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or

18

style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

- control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

- control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

- control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

- place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or

19

areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **301i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c** displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303j** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within an Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types

20

and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshowing) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define a unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not lim-

ited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services

Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert. "Backend Synchronization"
Goto a specific Internal Web Page	Goto Widget Object
Goto the next Internal Web Page	Generate Alert. "With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children, Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen,

Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319d**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

23

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and/or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3rd party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. **10**.

Referring again to FIG. **3E**, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define the record. If the Feed Collector data base is selected, for example, then the RSS "Channel Name" and the "Channel Feed" drop down menus will be available for symbolically selected the record. For other data bases the RSS "Channel Name" and the "Channel Feed" drop down menus are replaced by a "Record ID" text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected.

Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by "Record ID" if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the "Select Link" will cause the current data base selections to begin the blink in some manner, and the "Select Link" will change to "Create Link". The user could still change the data base and attribute choices. Touching the "Create Link" will set the "Link Set" checkbox and the "Create Link" will be replaced by "Delete Link" if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. **4A** and **4B** according to the system **100**, where FIG. **4A** shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. **4B** shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform

24

110 is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. **4B**. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. **4A**, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes.

Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways. Launch Strips and Graphical List Templates Launch Strips

Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. **5** shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector

25

309 with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip 502 also includes UI objects 502a and 503g, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device 130. Launch strip 502 may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List 601a, in FIG. 6B as a display of a Widget Selection List 601b, and in FIG. 6C as a display of a Phone List 601c. Lists 601a, 601b, and 601c may be displayed on canvas 305 or on screen 137 of device 130 having the proper Player and Application. As illustrated, graphical lists 601a, 601b, and 601c may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service 250 or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list 601a, 601b, and 601c has several individual entries that are each linked to specific actions. Thus Channel Selection List 601a shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List 601b includes several objects presenting different widgets for selecting. Phone List 601c includes a list phone number objects of names that, when selected by a user of device 130 cause the number to be dialed Entries in Phone List 601c may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List 601c may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform 110 allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

Personalization and Temporal Adoption

System 100 and 200 permit for the personalization of device 130 by a variety of means. Specifically, what is displayed on screen 137 may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the

26

phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display 700 of a mash-up which may be on displayed canvas 305 or on screen 137 of device 130 having the proper Player and Application. Display 700 includes several object 701 that have been dynamically bound, including an indication of time 701a, an indication of unread text messages 701b, an RSS news feed 701c (including 2 "ESPN Top Stories" 701c1 and 701c2), components 701d from two Web Services—a weather report ("The Weather Channel"), and a traffic report 701e ("TRAFFIC.COM").

In assembling the information of display 700, device 130 is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device 130 has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be maintained as persistent storage on device 130 when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base. Push Capable Systems

In another embodiment system 100 or 200 is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device 130 can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device 130.

FIG. 8 is a schematic of an embodiment of a push enabled system 800. System 800 is generally similar to system 100 or 200. Device 130 is shown as part of a schematic of a push capable system 800 in FIG. 8. System 800 includes a website system 801 hosting a website 801, a server 803 and a content server 805. System 801 is connected to servers 803 and/or 805 through the Internet. Server 803 is generally similar to server 120, servers 805 is generally similar to server 140.

In one embodiment, a user sets up a weekly SMS update from website system 801. System 801 provides user information to server 803, which is an SMS server, when an update is ready for delivery. Server 803 provides device 130 with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website 801 also provides content server 805 with the content of the update. When a user of device 130 responds to the SMS query, the response is provided to content server 805, which provides device 130 with updates including the subscribed content.

In an alternative embodiment of system 800, server 803 broadcasts alerts to one or more devices 130, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system 900. System 900 is generally similar to system 100, 200, or 800. In system 900 a user requests information using an SMS code, which is delivered to device

27

130. System 900 includes a promotional code 901, a third-party server 903, and content server 805. Server 803 is connected to servers 803 and/or 805 through the Internet, and is generally similar to server 120.

A promotional code 901 is provided to a user of device 130, for example and without limitation, on print media, such as on a cereal box. The use of device 130 sends the code server 903. Server 903 then notifies server 805 to provide certain information to device 130. Server 805 then provides device 130 with the requested information.

Device Routines

Device routines 114 may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server 140; an expanding set of web services 250 available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector 1010 and content gateway 1130.

Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform 110 SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some filed value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response.

RSS/Atom and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule

28

Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database.

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010, database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can to presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made at on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation

process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

- 1: A file format version number, used for upgrading database in future releases.
- 2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.
- 3: Whether the Application is a web site.
- 4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.
- 5: Web page and styles high watermarks.
- 6: The Websitename.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects ((i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of

31

line segments per paragraph line and contains the values used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requir-

32

ing dynamic web page and object resizing, virtual screen resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.
- 3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Website-name".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first

33

web page, then they are flagged for compression and inclusion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Website-name".class, customized run time engine file, and the "Website-name".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Website-name".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Website-name".bat and "Website-namelib".bat files are written. The "Website-name".bat and "Website-name".bat files are then executed under DOS, creating compressed "Website-name".cab and "Website-namelib".cab files and compressed "Website-name" jar and "Website-namelib" jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

Displaying Content on a Device Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reassembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform 110.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device 130. Specifically, device 130 operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device 130 for display on screen 137.

Response Director

In one embodiment, system 100 includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

34

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. 12 illustrates one embodiment of a system 1200 that includes a response director 210, a user agent database 1201, an IP address database 1203, and a file database 1205. System 1200 is generally similar to system 100, 200, 800, 900, 1000, or 1100.

Databases 1201, 1203, and 1205 may reside on server 120, 210, or any computer system in communication with response director 210. System 1200, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database 1201 includes user agent information regarding individual devices 130 that are used to identify the operating system on the device. IP address database 1203 identifies the carrier/operator of each device 130. File database 1205 includes data files that may operate on each device 130.

The following is an illustrative example of the operation of response director 210. First, a device 1300 generates an SMS message, which automatically sends an http:// stream that includes handset information and its phone number to response director 210. Response director 210 then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database 1201 which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address 1203 identifies the carrier/operator.

File database 1205 contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device 130. A list of available Applications are returned to a decision tree, which then returns, to device 130, the Application that is appropriate for the respective device. For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director 210 generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of.

If there is an Application that has a data type that device 130 cannot support, for example, video, response director 210 sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director 210 is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algo-

35

rithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform **110** automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128×128, 176×208, 240×260, 220×220, and many other customized sizes in between.

FIG. **13** is a schematic of an embodiment of a system **1300**. System **1300** is generally similar to system **1200**. System **1300** is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System **1300** includes response director **210**, a device characteristics operator and local database **1301**, a player profile database **1303** and a player build process **1305**, which may be authoring platform **110**.

As an example of system **1300**, when response director **210** receives an SMS message from device **130**, the response director identifies the device characteristics operator and locale from database **1301** and a Player URL from database **1303** and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director **210** by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform **110** may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is

36

scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a magnetic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described

37

in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases “in one embodiment,” “in an embodiment,” or “in certain embodiments” in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term “comprising” shall be synonymous with “including,” “containing,” or “characterized by,” is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. “Comprising” is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. “Comprising” leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:
computer memory storing a registry of:

a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the symbolic name, and has a preferred UI object, and

b) an address of the web service;

an authoring tool configured to:

define a (UI) object for presentation on the display,

where said defined UI object corresponds to a web component included in said registry selected from a group consisting of an input of the web service

38

and an output of the web service, where each defined UI object is either: 1) selected by a user of the authoring tool; or 2) automatically selected by the system as the preferred UI object corresponding to the symbolic name of the web component selected by the user of the authoring tool,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object, where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name, and

produce an Application including the selected symbolic name of the defined UI object, where said Application is a device-independent code; and

a Player, where said Player is a device-dependent code, wherein, when the Application and Player are provided to the device and executed on the device, and when the user of the device provides one or more input values associated with an input symbolic name to an input of the defined UI object,

1) the device provides the user provided one or more input values and corresponding input symbolic name to the web service,

2) the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

3) said Player receives the output symbolic name and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and

generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

define a SMS field or list; and

generate code that, when executed on the device, allows a user to supply an SMS address to said SMS field or list.

39

10. The system of claim 1,
 where said code includes three or more codes, where one
 of said three or more codes is device specific, and
 where two of said three or more codes is device
 independent. 5

11. The system of claim 1, where said code is provided
 over said network.

12. The system of claim 1, wherein said defined UI object
 corresponds to a widget.

13. The system of claim 1, where said Player is activated 10
 and runs in a web browser.

14. The system of claim 1, where said Player is a native
 program.

15. A method of displaying content on a display of a
 device having a Player, where said Player is a device- 15
 dependent code, said method comprising:
 defining a user interface (UI) object for presentation on
 the display, where said UI object corresponds to a web
 component included in a registry of one or more web
 components selected from a group consisting of an 20
 input of a web service and an output of the web service,
 where each web component includes a plurality of
 symbolic names of inputs and outputs associated with
 each web service, and where the registry includes: a)
 symbolic names required for evoking one or more web 25
 components each related to a set of inputs and outputs
 of the web service obtainable over a network, where the
 symbolic names are character strings that do not con-
 tain either a persistent address or pointer to an output
 value accessible to the web service, and b) an address 30
 of the web service, and where each defined UI object is
 either: 1) selected by a user of an authoring tool; or 2)
 automatically selected by a system as a preferred UI
 object corresponding to a symbolic name of the web
 component selected by the user of the authoring tool; 35
 selecting the symbolic name from said web component
 corresponding to the defined UI object, where the
 selected symbolic name has an associated data format
 class type corresponding to a subclass of UI objects that
 support the data format type of the symbolic name, and 40
 has the preferred UI object;
 associating the selected symbolic name with the defined
 UI object; and
 producing an Application including the selected symbolic
 name of the defined UI object, where said Application 45
 is a device-independent code, wherein, when the Appli-
 cation and Player are provided to the device and
 executed on the device, and when the user of the device
 provides one or more input values associated with an
 input symbolic name to an input of the defined UI 50
 object,

40

1) the device provides the user provided one or more input
 values and corresponding input symbolic name to the
 web service,
 2) the web service utilizes the input symbolic name and
 the user provided one or more input values for gener-
 ating one or more output values having an associated
 output symbolic name,
 3) said Player receives the output symbolic name and
 corresponding one or more output values and provides
 instructions for the display of the device to present an
 output value in the defined UI object.

16. The method of claim 15, where said registry includes
 definitions of input and output related to said web service.

17. The method of claim 15, where said web component
 is a text chat, a video chat, an image, a slideshow, a video,
 or an RSS feed.

18. The method of claim 15, where said UI object is an
 input field for a chat.

19. The method of claim 15, where said UI object is an
 input field for a web service.

20. The method of claim 15, where said UI object is an
 input field usable to obtain said web component, where said
 input field includes a text field, a scrolling text box, a check
 box, a drop down-menu, a list menu, or a submit button.

21. The method of claim 15, where said web component
 is an output of a web service, is the text provided by one or
 more simultaneous chat sessions, is the video of a video chat
 session, is a video, an image, a slideshow, an RSS display,
 or an advertisement.

22. The method of claim 15, further comprising:
 defining a phone field or list; and
 generating code that, when executed on the device, allows
 a user to supply a phone number to said phone field or
 list.

23. The method of claim 15, further comprising:
 defining a SMS field or list; and
 generating code that, when executed on the device, allows
 a user to supply an SMS address to said SMS field or
 list.

24. The method of claim 15, and such that said Player
 interprets dynamically received, device independent values
 of the web component defined in the Application.

25. The method of claim 15, further comprising:
 providing said Application and Player over said network.

26. The method of claim 15, wherein said UI object
 corresponds to a widget.

27. The method of claim 15, where said Player is activated
 and runs in a web browser.

28. The method of claim 15, where said Player is a native
 program.

* * * * *

EXHIBIT E



(12) **United States Patent**
Rempell et al.

(10) **Patent No.:** **US 9,928,044 B2**
(45) **Date of Patent:** ***Mar. 27, 2018**

(54) **SYSTEMS AND METHODS FOR
PROGRAMMING MOBILE DEVICES**

(71) Applicant: **Express Mobile, Inc.**, Novato, CA (US)

(72) Inventors: **Steven H. Rempell**, Novato, CA (US);
David Chrobak, Clayton, CA (US);
Ken Brown, San Martin, CA (US)

(73) Assignee: **Express Mobile, Inc.**, Novato, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **15/706,746**

(22) Filed: **Sep. 17, 2017**

(65) **Prior Publication Data**

US 2018/0004493 A1 Jan. 4, 2018

Related U.S. Application Data

(63) Continuation of application No. 15/370,990, filed on Dec. 6, 2016, now Pat. No. 9,766,864, which is a (Continued)

(51) **Int. Cl.**
G06F 3/048 (2013.01)
G06F 9/44 (2018.01)
H04L 29/08 (2006.01)
G06F 3/0484 (2013.01)
G06F 3/0482 (2013.01)
H04L 12/58 (2006.01)
H04L 29/06 (2006.01)

(52) **U.S. Cl.**

CPC **G06F 8/38** (2013.01); **G06F 3/0482** (2013.01); **G06F 3/04842** (2013.01); **G06F 8/34** (2013.01); **G06F 9/4443** (2013.01); **H04L 51/046** (2013.01); **H04L 65/60** (2013.01); **H04L 67/02** (2013.01)

(58) **Field of Classification Search**

CPC G06F 3/048
See application file for complete search history.

(56)

References Cited

U.S. PATENT DOCUMENTS

2004/0055017 A1 3/2004 Delpuch et al.
2004/0163020 A1 8/2004 Sidman
(Continued)

OTHER PUBLICATIONS

Stina Nylander et al. "The Ubiquitous Interactor-Device Independent Access to Mobile Services" (Computer-Aided Design for User Interfaces IV, Proceedings of the Fifth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Jan. 2004, pp. 271-282).*

(Continued)

Primary Examiner — Xuyang Xia

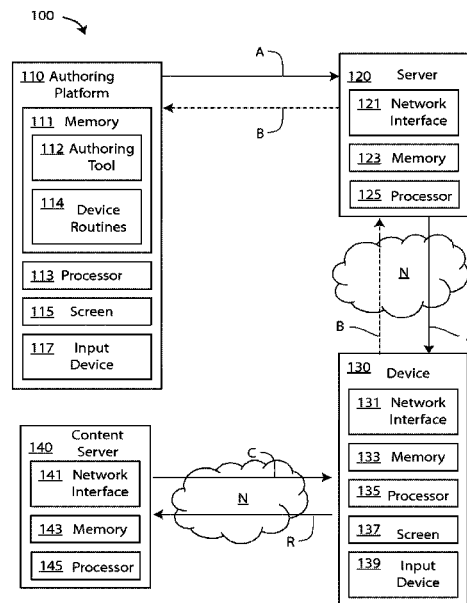
(74) *Attorney, Agent, or Firm* — Steven R. Vosen

(57)

ABSTRACT

Embodiments of a system and method are described for generating and distributing programming to mobile devices over a network. Devices are provided with Players specific to each device and Applications that are device independent. Embodiments include a full-featured WYSIWYG authoring environment, including the ability to bind web components to objects.

28 Claims, 18 Drawing Sheets



Related U.S. Application Data

continuation of application No. 14/708,094, filed on May 8, 2015, now Pat. No. 9,542,163, which is a continuation of application No. 12/936,395, filed as application No. PCT/US2009/039695 on Apr. 6, 2009, now Pat. No. 9,063,755.

- (60) Provisional application No. 61/123,438, filed on Apr. 7, 2008, provisional application No. 61/113,471, filed on Nov. 11, 2008, provisional application No. 61/166,651, filed on Apr. 3, 2009.

- (56) **References Cited**

U.S. PATENT DOCUMENTS

2004/0199614 A1* 10/2004 Shenfield H04L 29/06
709/220

2005/0149935 A1 7/2005 Benedetti
2005/0273705 A1* 12/2005 McCain G06F 17/24
715/234
2006/0063518 A1 3/2006 Paddon et al.

OTHER PUBLICATIONS

International Search Report and Written Opinion—PCT/US2009/039695—Aug. 21, 2009.
International Preliminary Report on Patentability and Written Opinion—PCT/US2009/039695—Oct. 21, 2010.
Rempell et al, co-owned U.S. Pat. No. 9,471,287, Issue date of Oct. 18, 2016.
Rempell et al, co-owned U.S. Pat. No. 9,507,571, Issue date of Nov. 29, 2016.
Rempell et al, co-owned U.S. Pat. No. 9,542,163, Issue date of Nov. 29, 2016.
Rempell et al, co-owned U.S. Pat. No. 9,766,864, Issue date of Sep. 19, 2017.

* cited by examiner

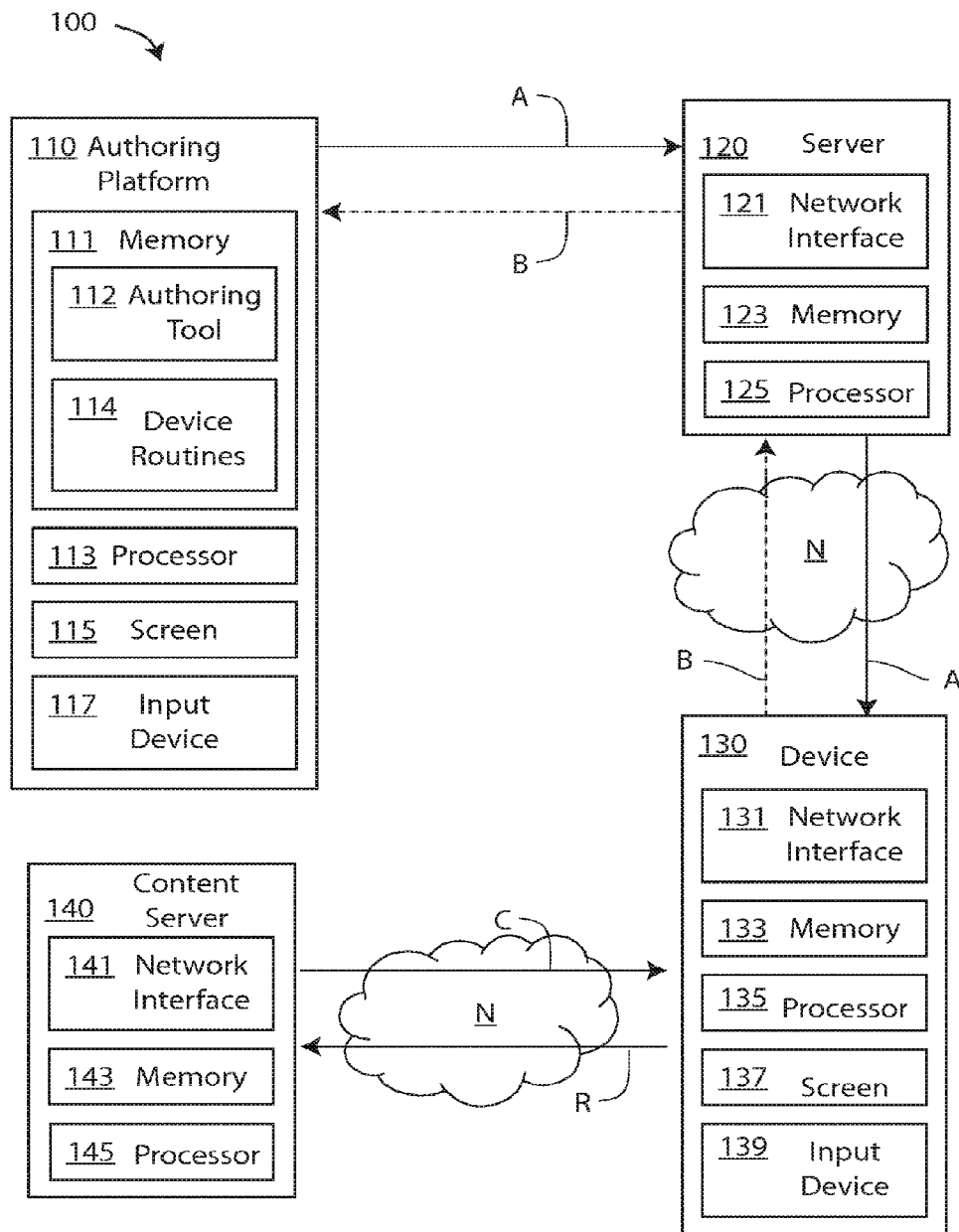


FIG. 1A

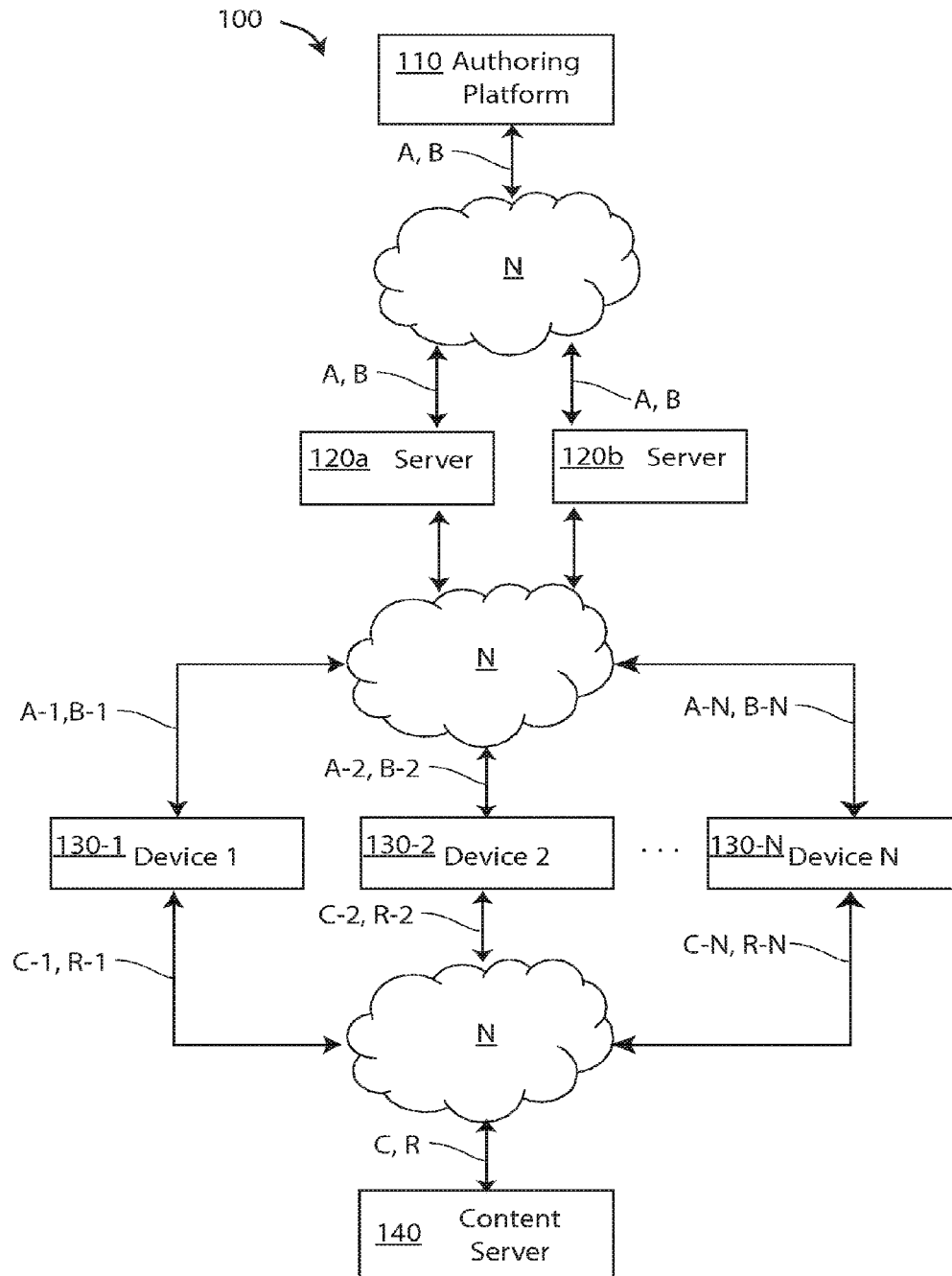


FIG. 1B

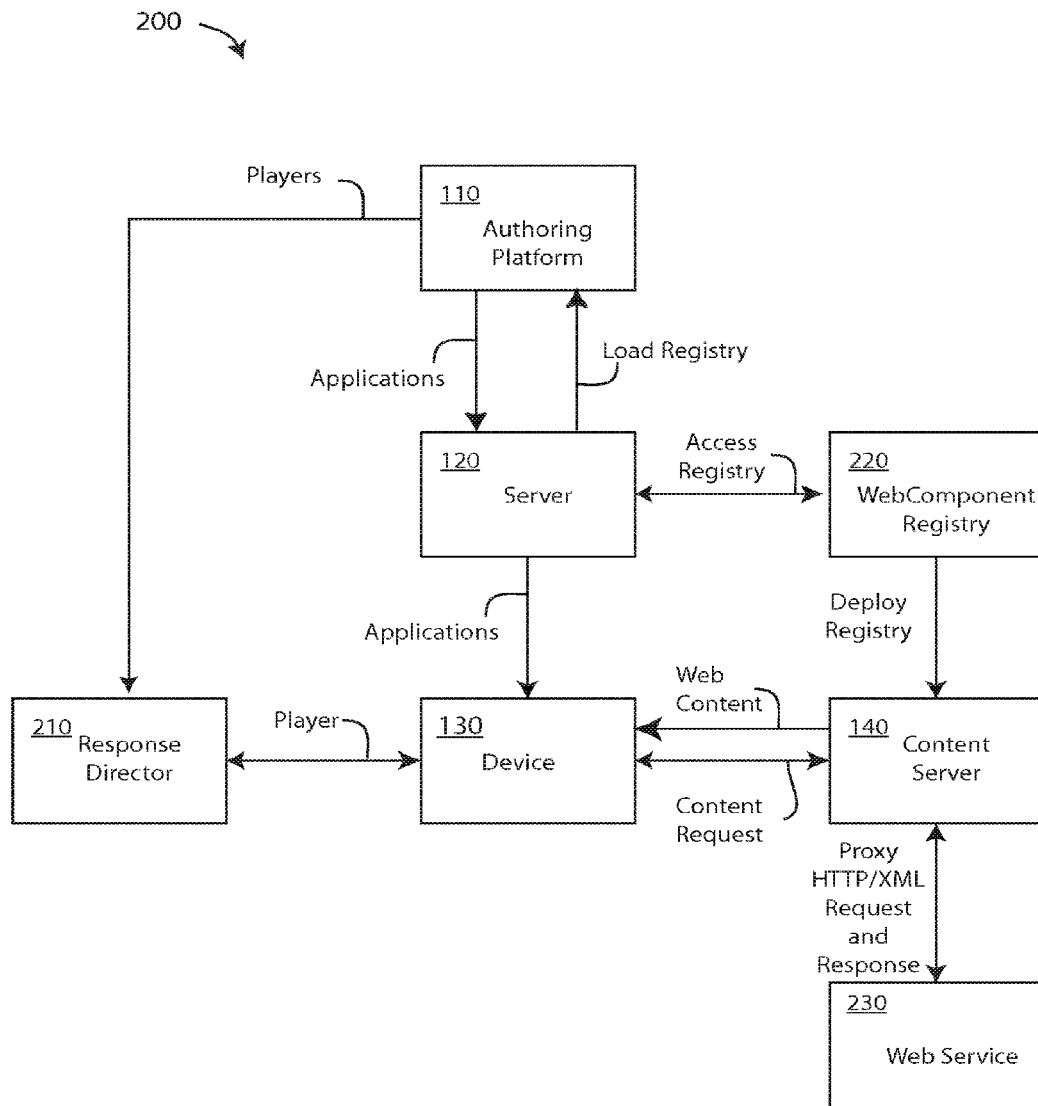
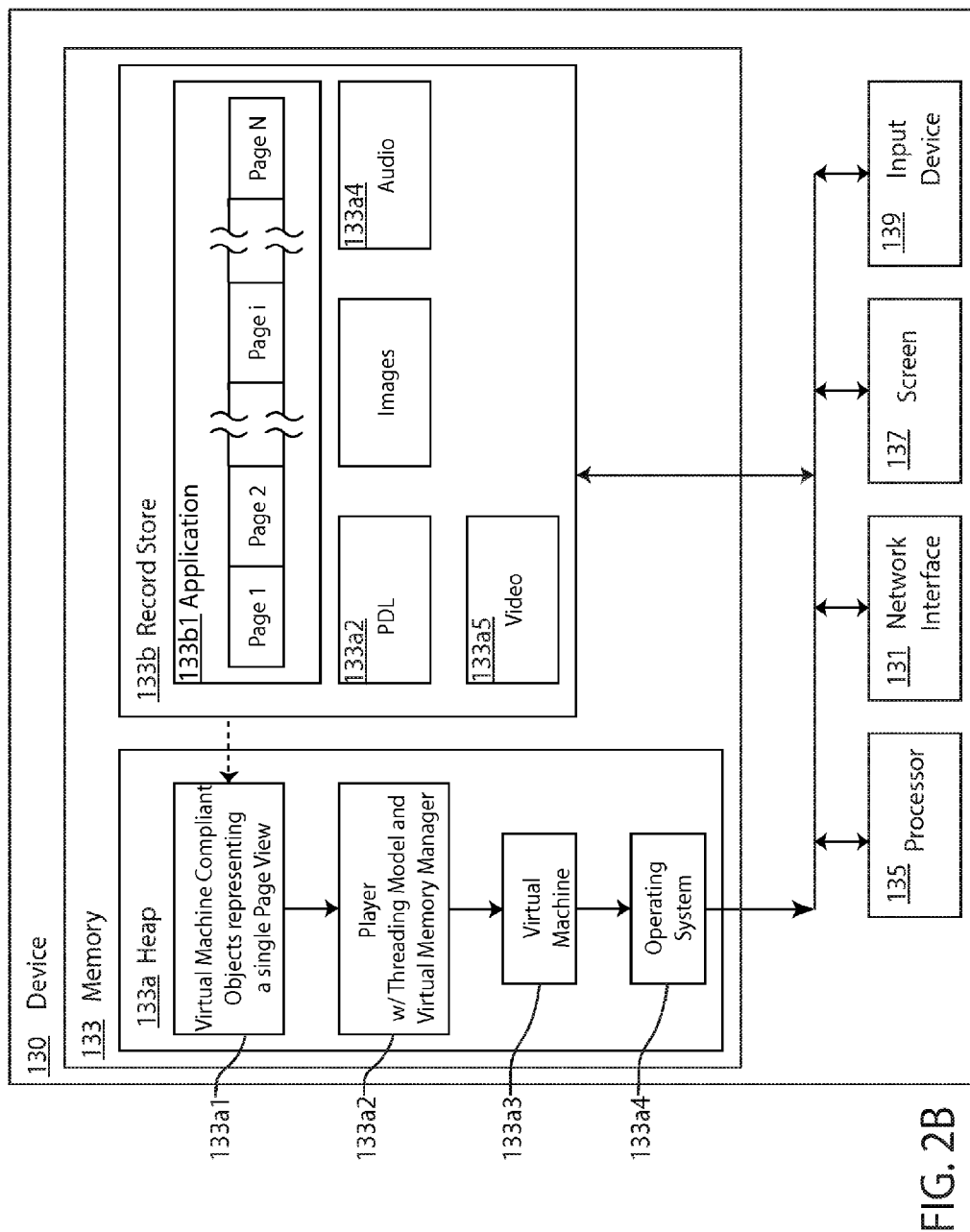
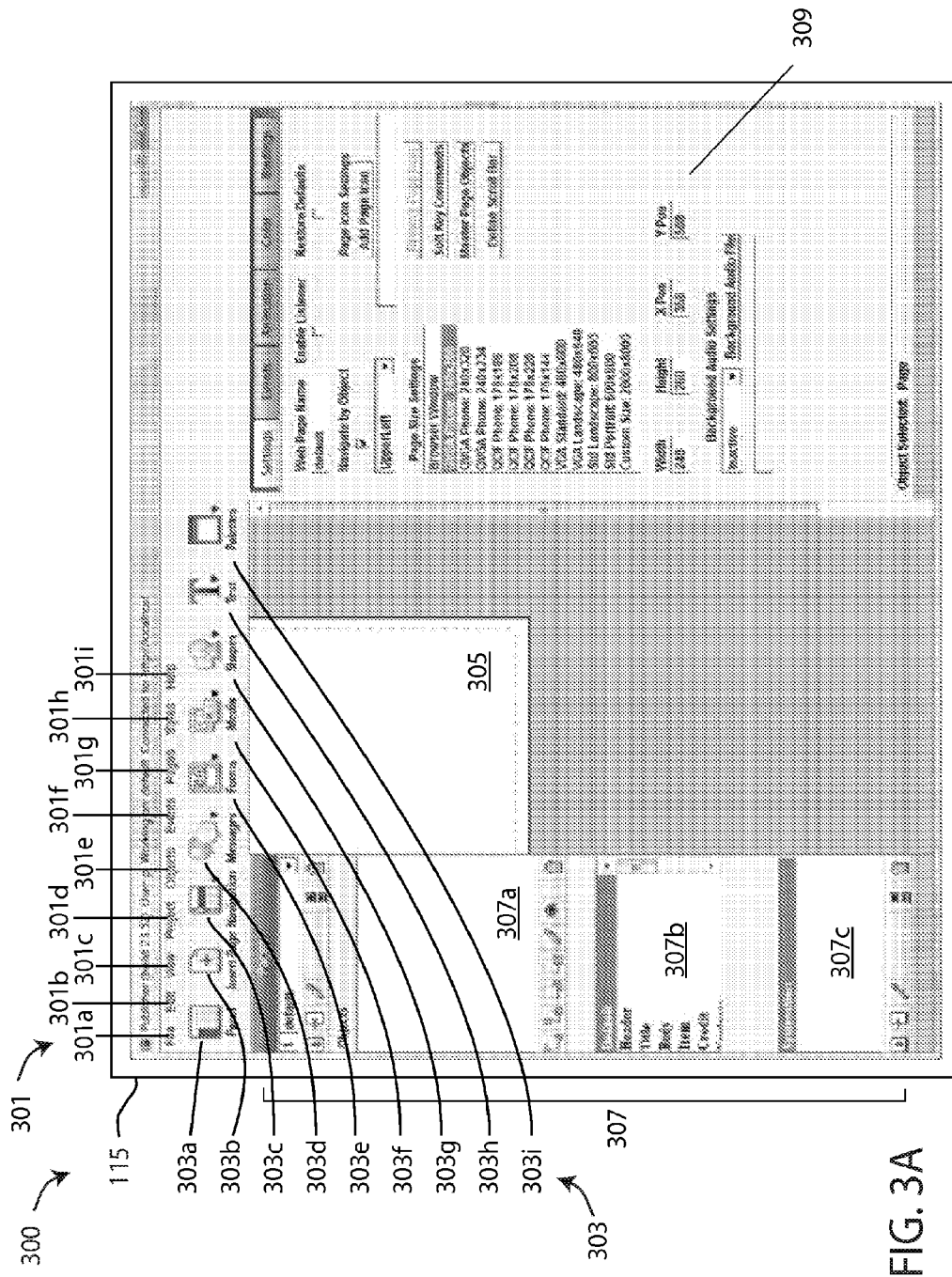


FIG. 2A





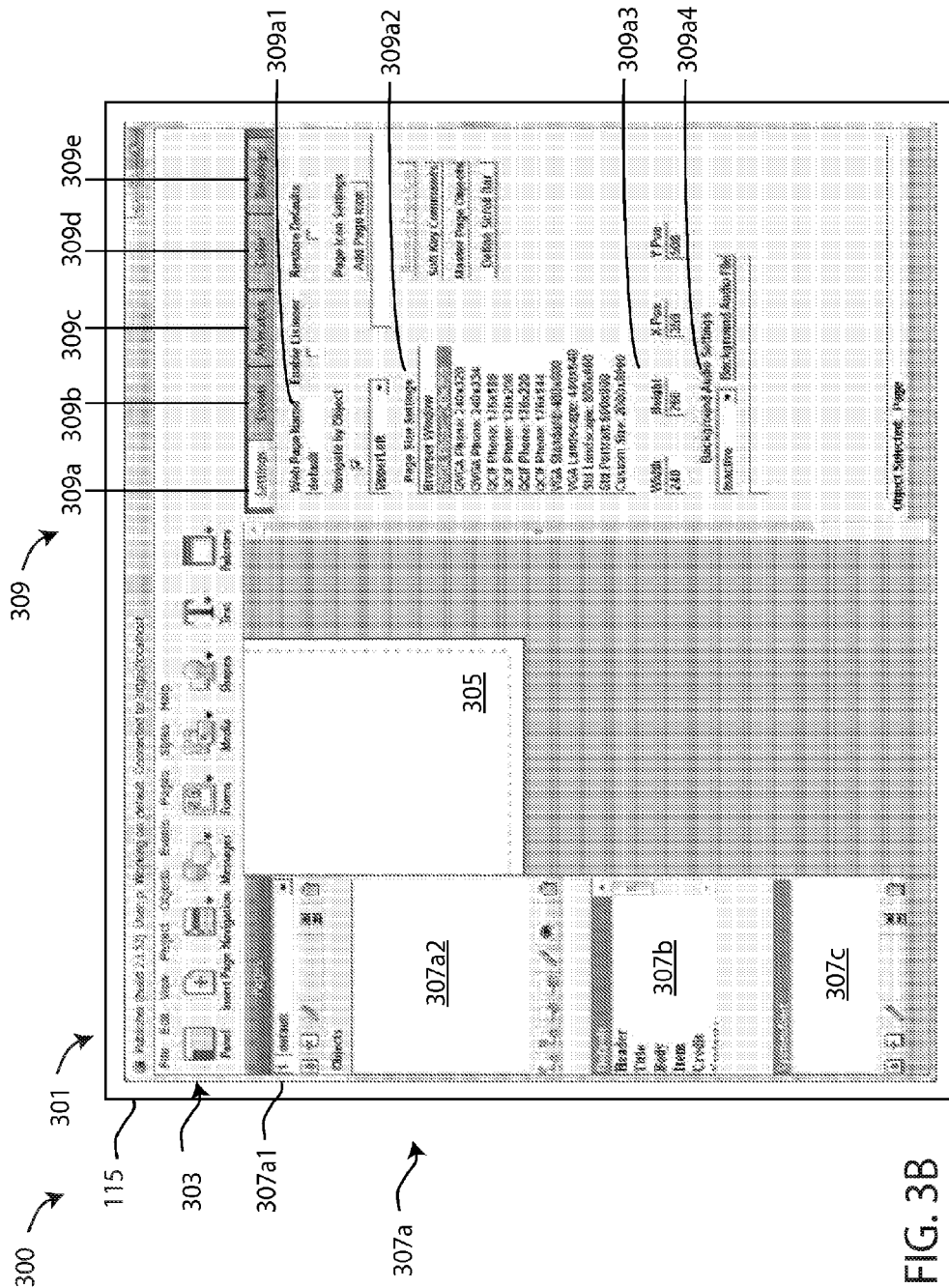


FIG. 3B

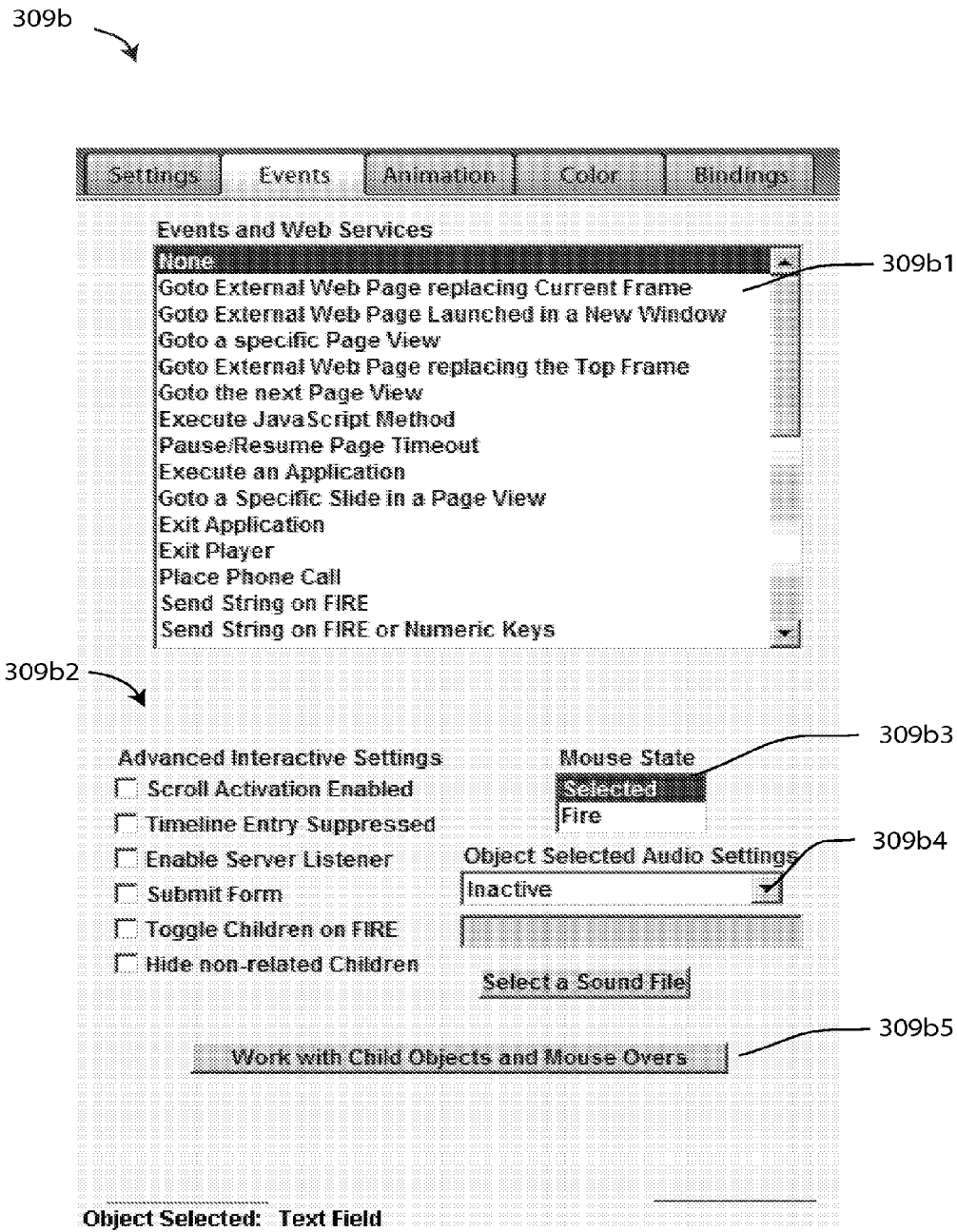


FIG. 3C

309c

Settings Events Animation Color Bindings

Activate Object Entry Timeline Specifications

Timeline Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Entry Animation Audio Track

Inactive Entry Audio File

Object Animation Specifications

Delay (Sec)	Direction	Movement	Duration (Sec)	Frames
0	None	None	0	1
1	Scroll Left	Fade	1	2
2	Scroll Right	Fade In	2	3
3	Custom	Fade Out	3	4
4	Multi-Point		4	5
5	Seek Cursor		5	6
6	Attach		6	7
7	Deposit		7	8
8	Send Home		8	9
9	Carom N		9	10
10	Carom NE		10	11

Pathname for this Object's Animation Audio Track

Inactive Main Audio File

Animation Cycles Custom Zoom % Avoid Cursor Dampen Anim.

1 0 ☒ ☐

Object Exit Timeline Specifications

Activate Delay (Sec) Direction Movement Duration (Sec) Frames

☐ 0 .0 None None 2 .0 10

Specifications for this Object's Exit Animation Audio Track

Inactive Exit Audio File

FIG. 3D

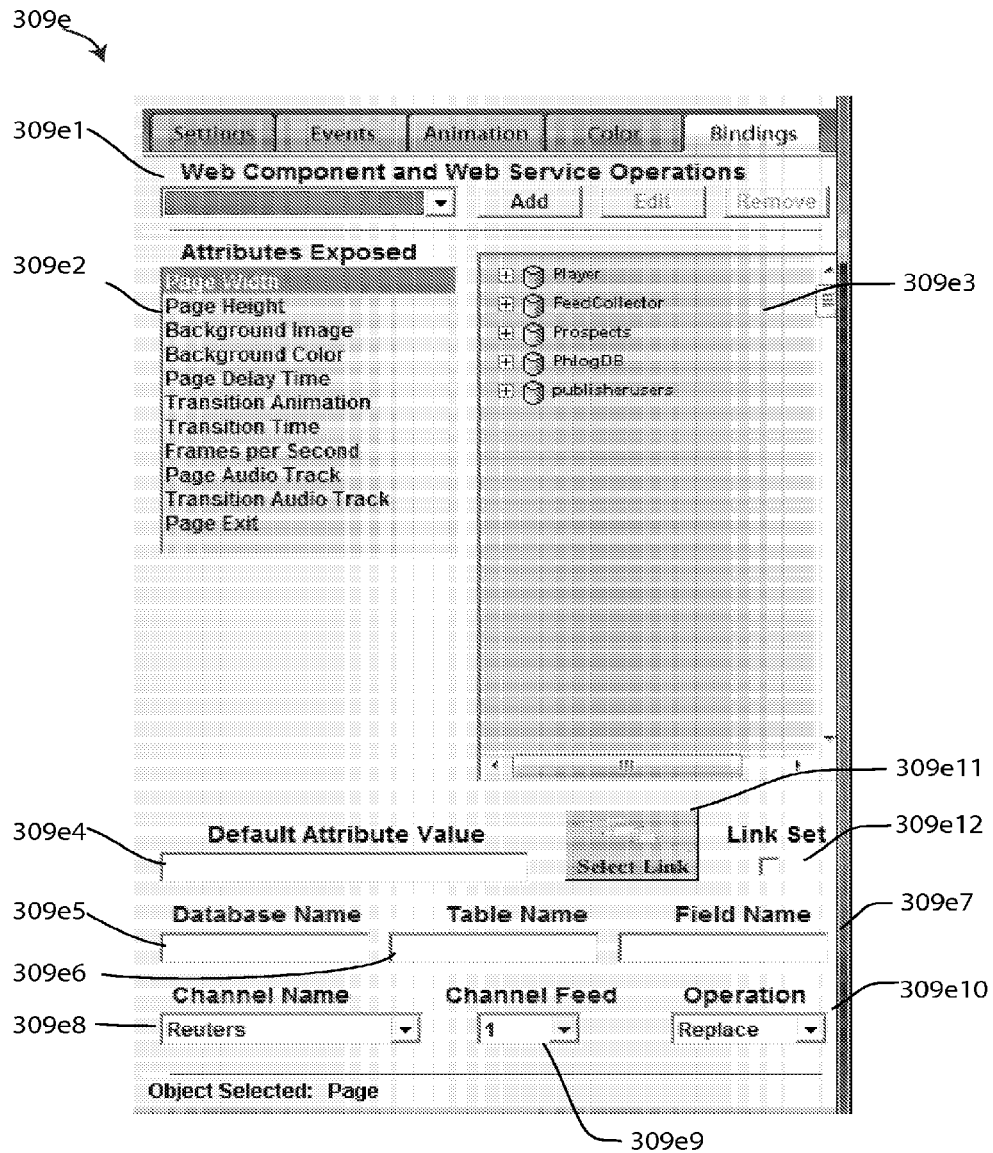


FIG. 3E

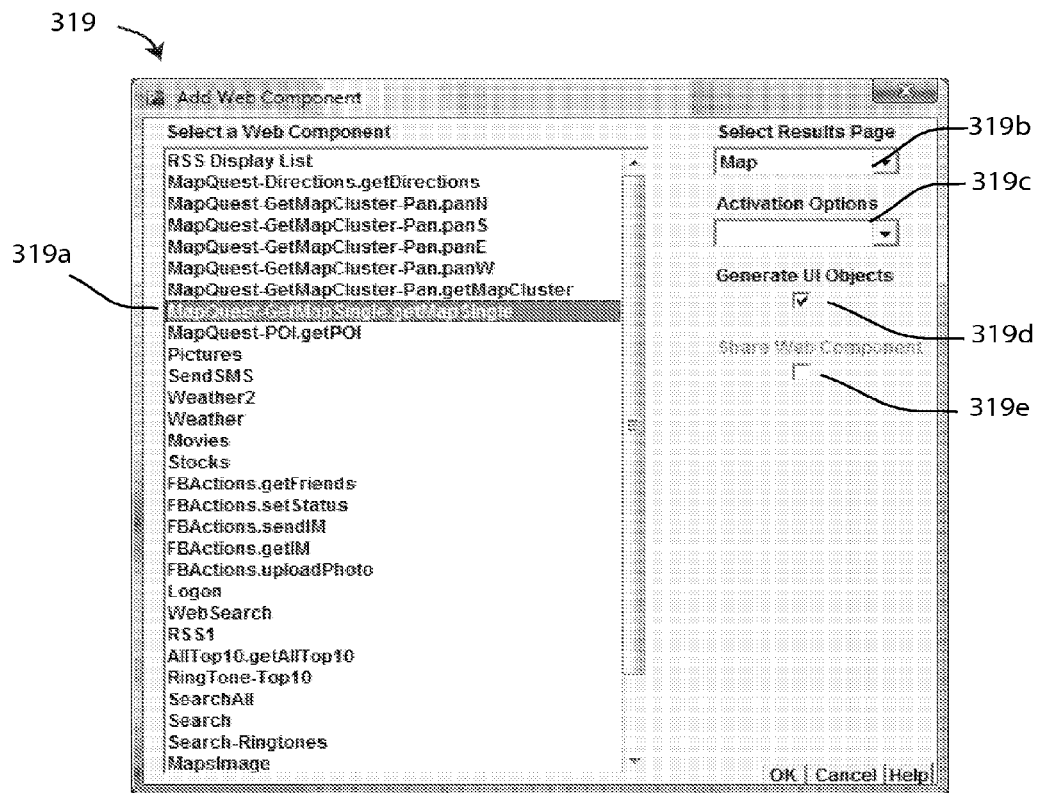


FIG. 3F

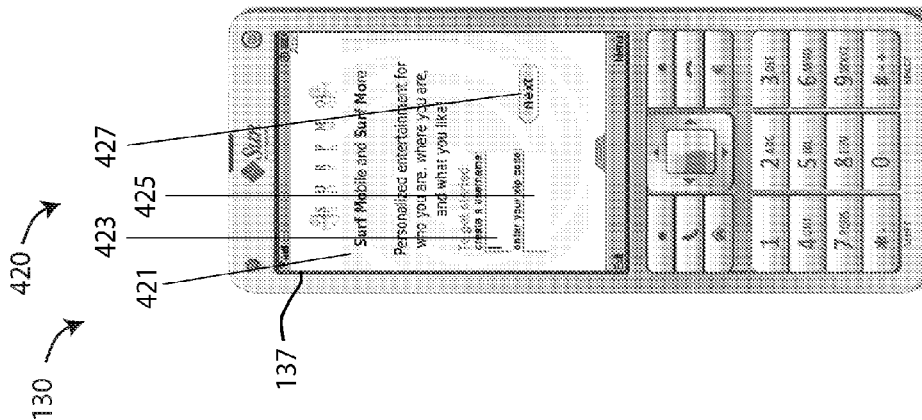


FIG. 4B

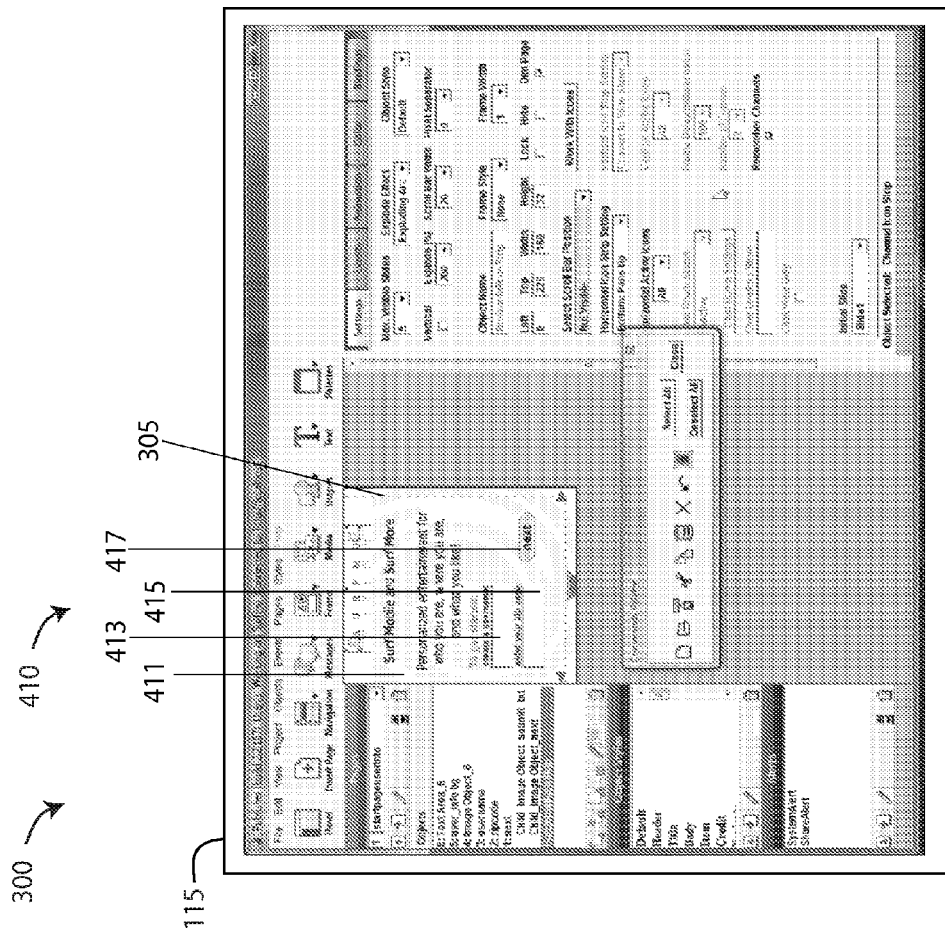


FIG. 4A

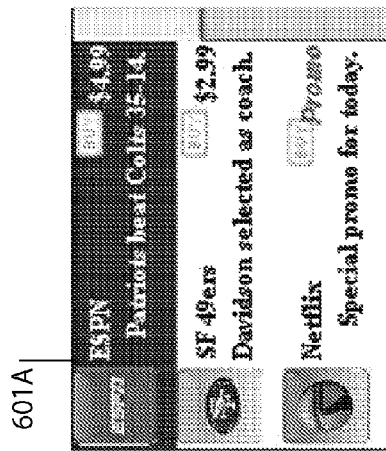


FIG. 6A

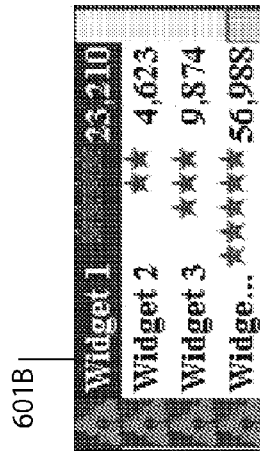


FIG. 6B



FIG. 6C

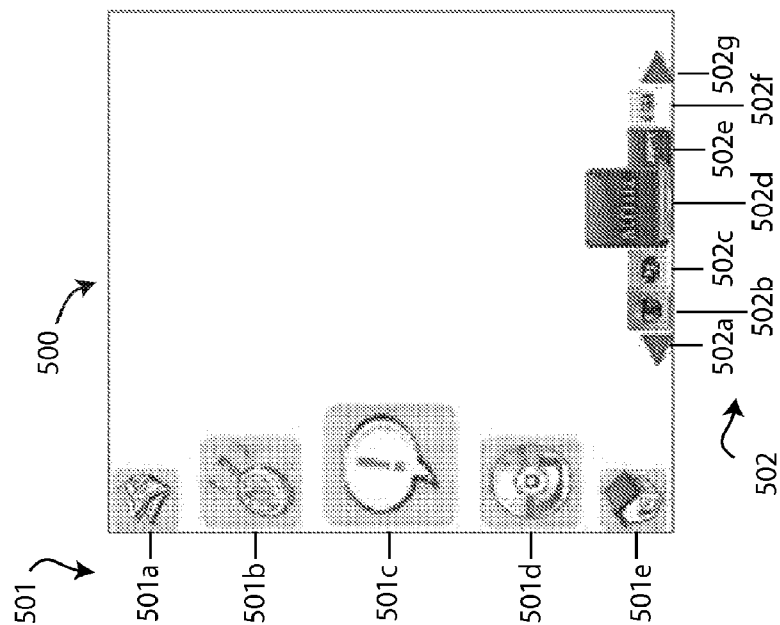


FIG. 5

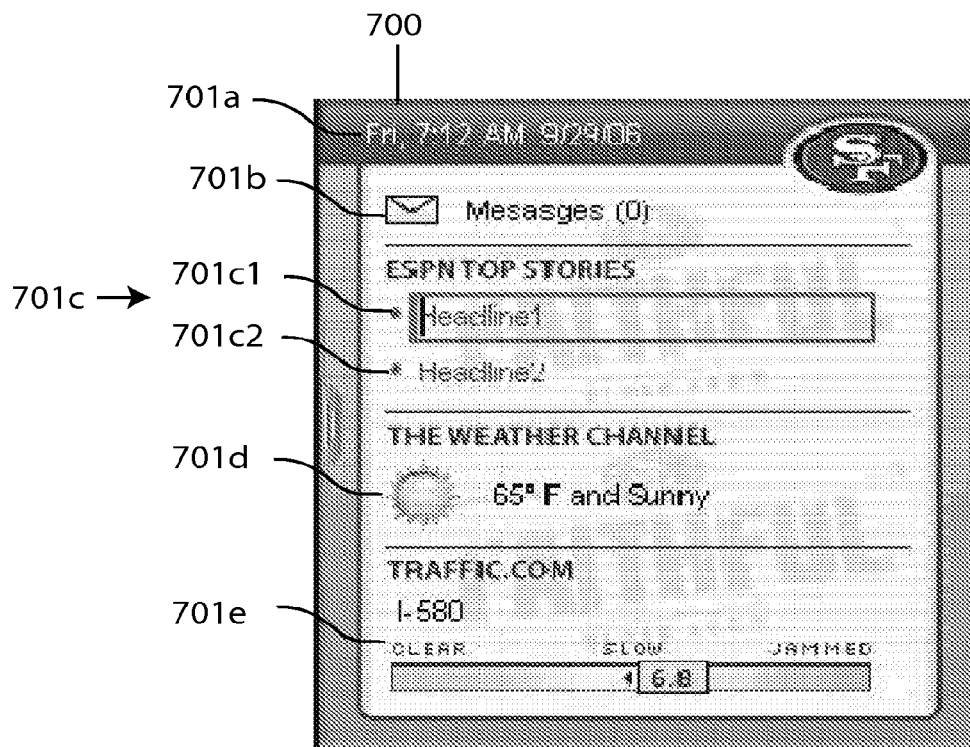


FIG. 7

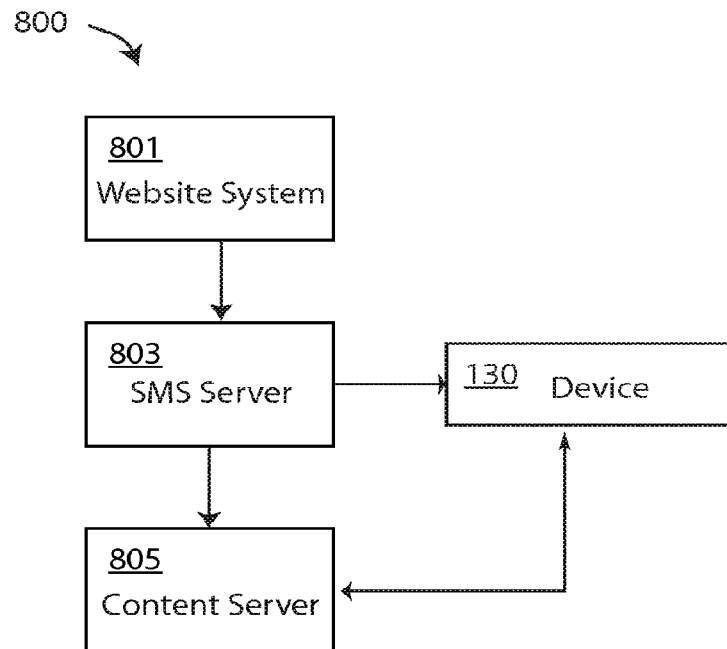


FIG. 8

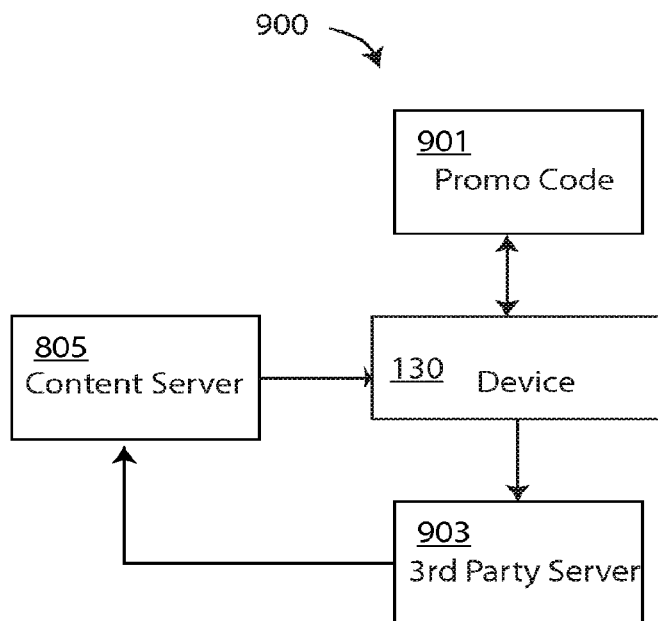


FIG. 9

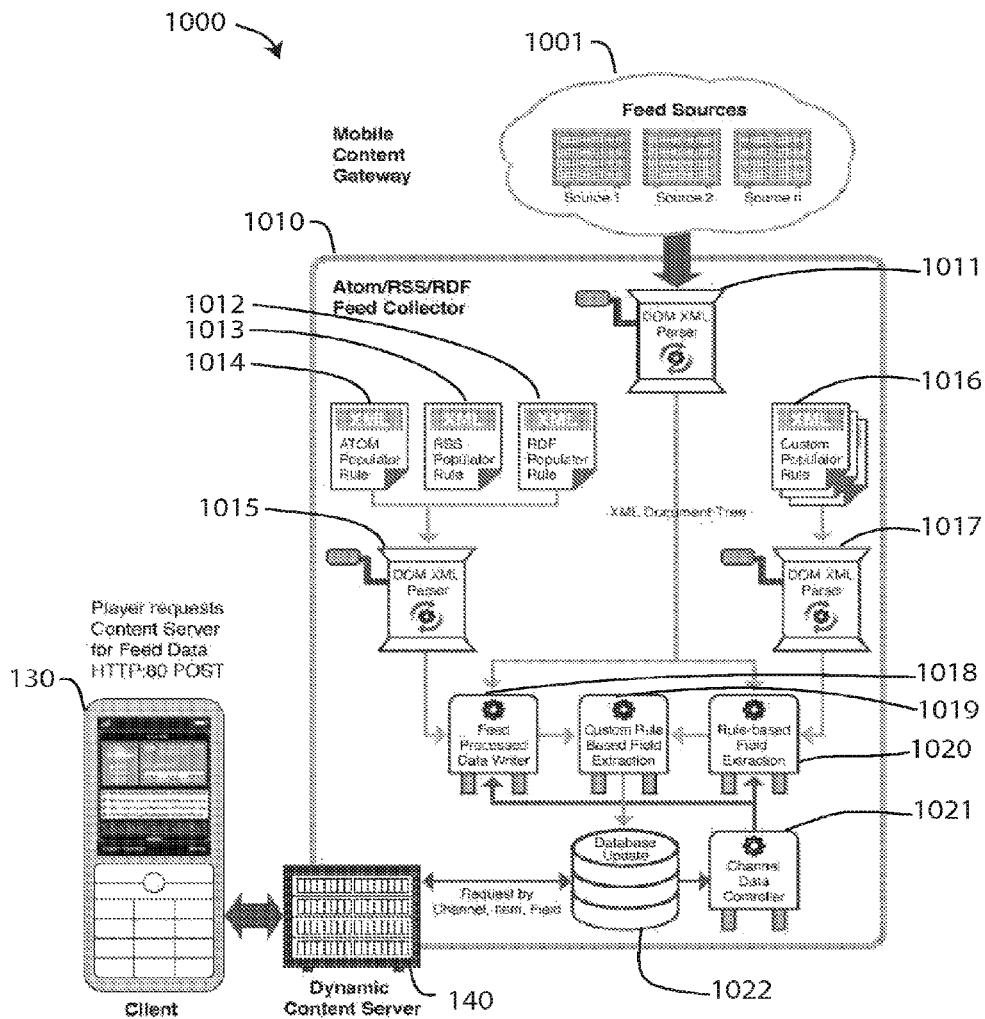


FIG. 10

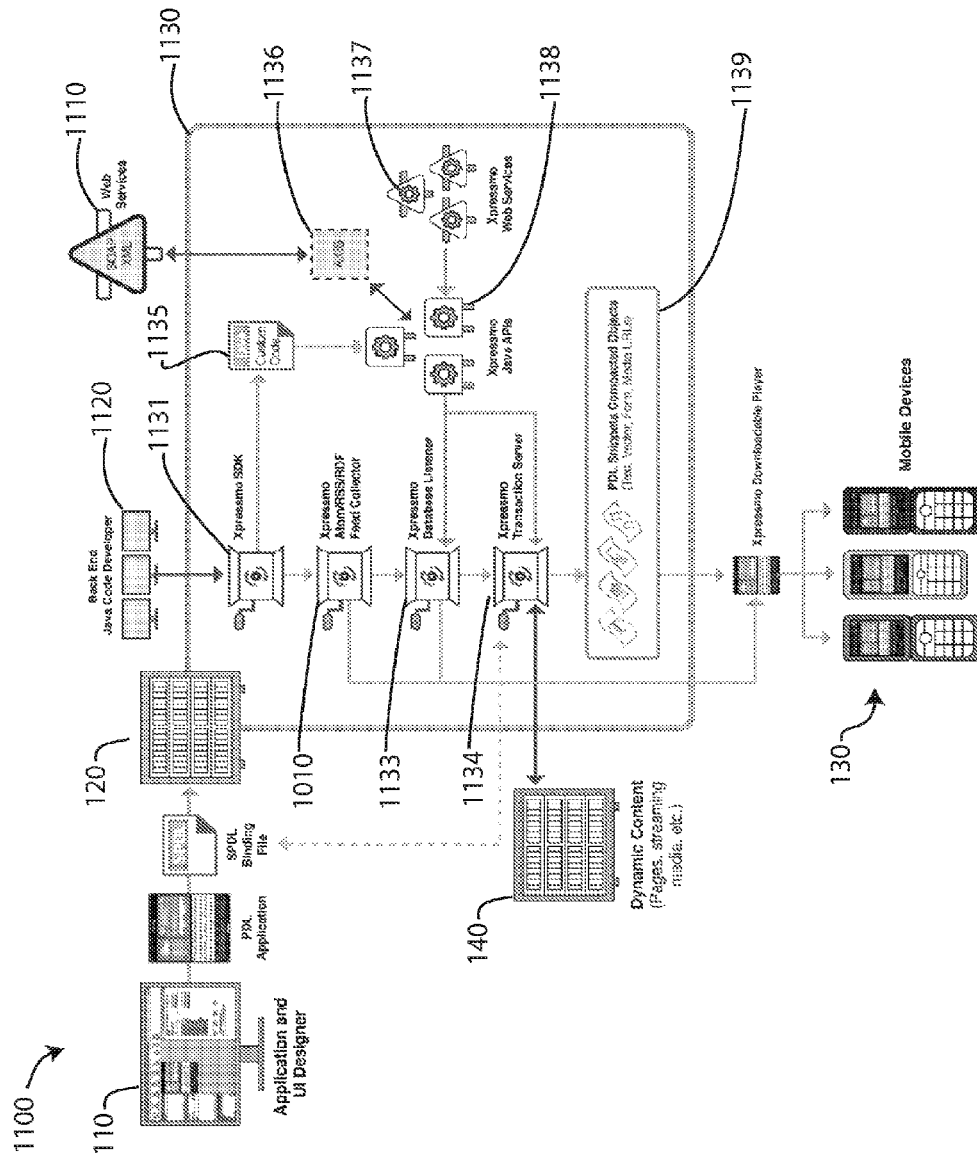


FIG. 11

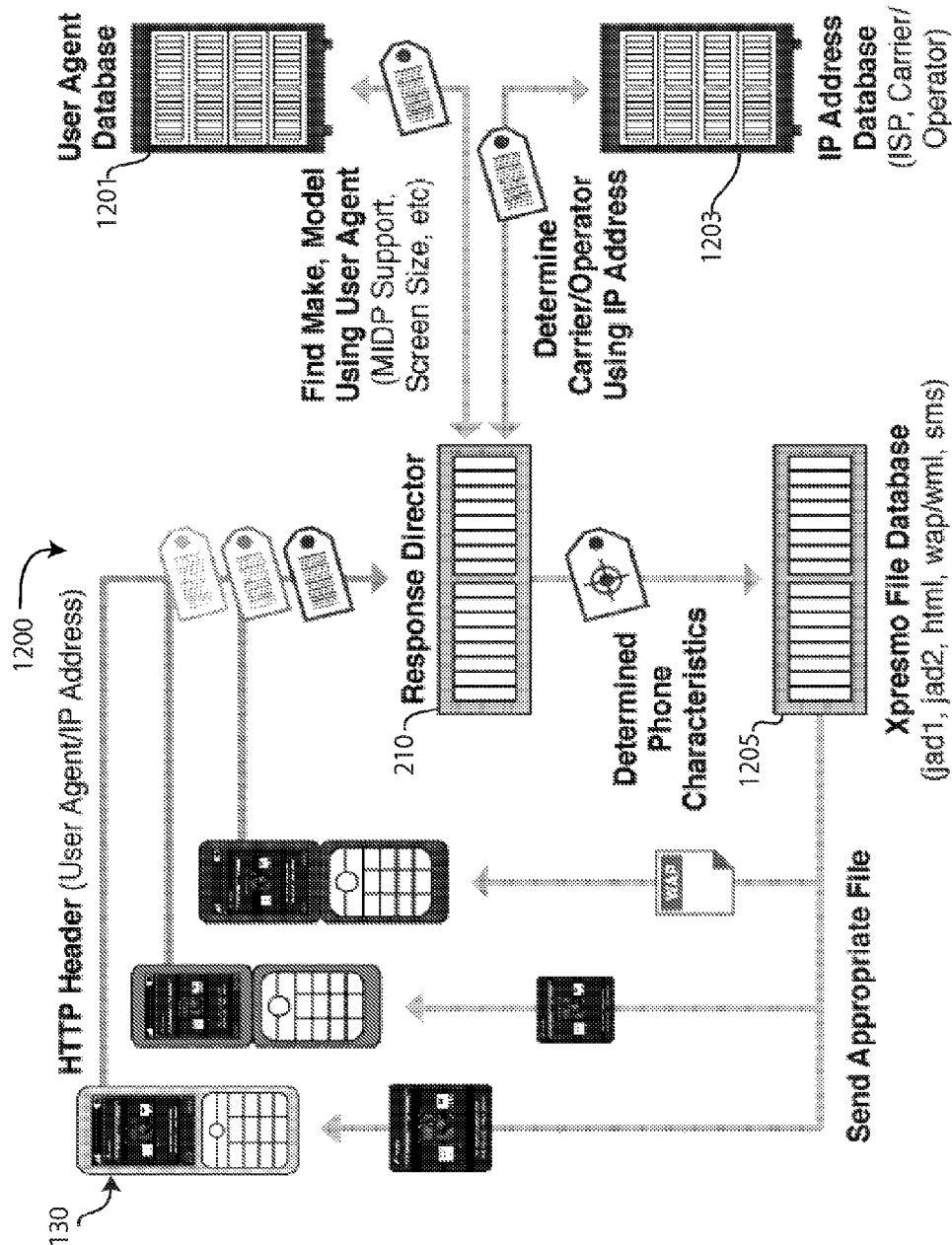


FIG. 12

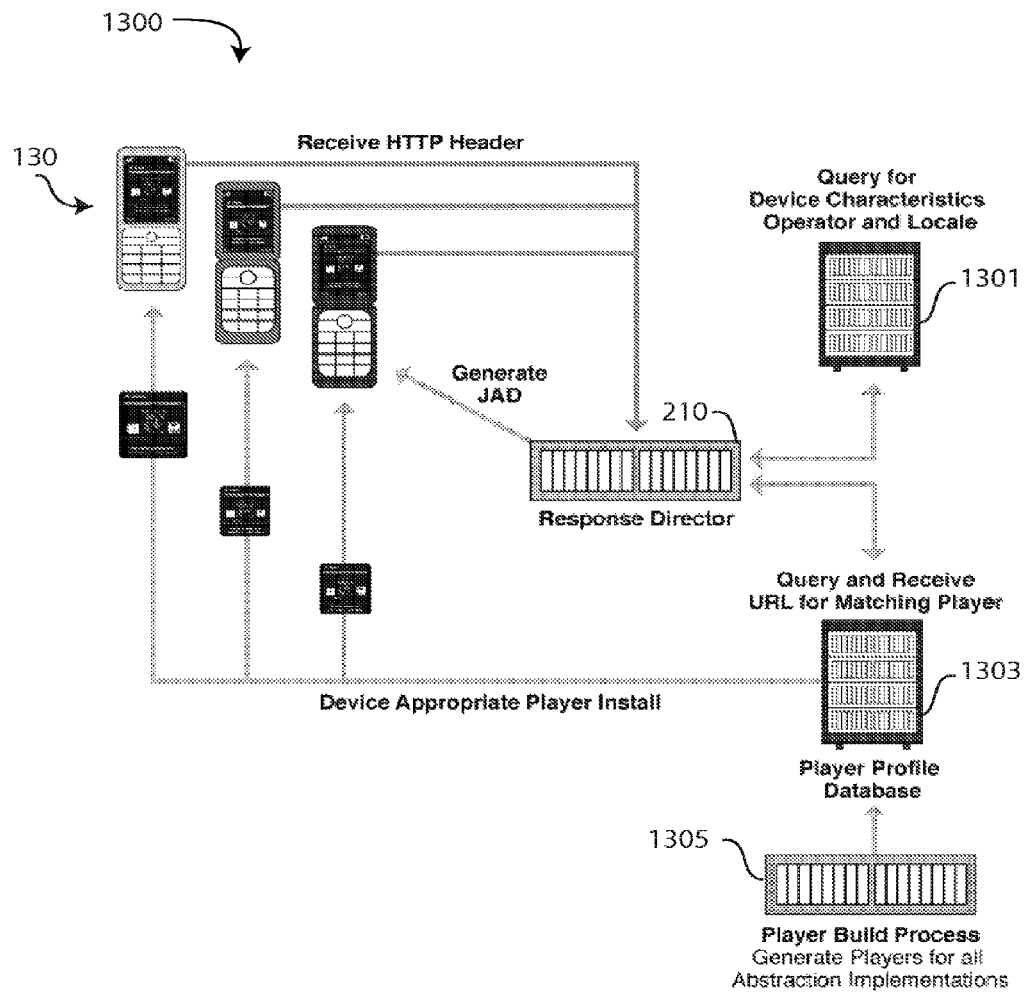


FIG. 13

1

SYSTEMS AND METHODS FOR PROGRAMMING MOBILE DEVICES

TECHNICAL FIELD

The present invention generally relates to providing software for mobile devices, and more particularly to a method and system for authoring Applications for devices.

BACKGROUND ART

Internet-connected mobile devices are becoming ever more popular. While these devices provide portability to the Internet, they generally do not have the capabilities of non-mobile devices including computing, input and output capabilities.

In addition, the mobility of the user while using such devices provides challenges and opportunities for the use of the Internet. Further, unlike non-mobile devices, there are a large number of types of devices and they tend to have a shorter lifetime in the marketplace. The programming of the myriad of mobile devices is a time-consuming and expensive proposition, thus limiting the ability of service providers to update the capabilities of mobile devices.

Thus there is a need in the art for a method and apparatus that permits for the efficient programming of mobile devices. Such a method and apparatus should be easy to use and provide output for a variety of devices.

DISCLOSURE OF INVENTION

In certain embodiments, a system is provided to generate code to provide content on a display of a platform. The system includes a database of web services obtainable over a network and an authoring tool. The authoring tool is configured to define an object for presentation on the display, select a component of a web service included in said database, associate said object with said selected component, and produce code that, when executed on the platform, provides said selected component on the display of the platform.

In certain other embodiments, a method is provided for providing information to platforms on a network. The method includes accepting a first code over the network, where said first code is platform-dependent; providing a second code over the network, where said second code is platform-independent; and executing said first code and said second code on the platform to provide web components obtained over the network.

In certain embodiments, a method for displaying content on a platform utilizing a database of web services obtainable over a network is provided. The method includes: defining an object for presentation on the display; selecting a component of a web service included in said database; associating said object with said selected component; and producing code that, when executed on the platform, provides said selected component on the display of the platform.

In one embodiment, one of the codes is a Player, which is a thin client architecture that operates in a language that manages resources efficiently, is extensible, supports a robust application model, and has no device specific dependencies. In another embodiment, Player P is light weight and extends the operating system and/or virtual machine of the device to: Manage all applications and application upgrades, and resolve device, operating system, VM and language fragmentation.

2

In another embodiment, one of the codes is an Application that is a device independent code that interpreted by the Player.

These features together with the various ancillary provisions and features which will become apparent to those skilled in the art from the following detailed description, are attained by the system and method of the present invention, preferred embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is an illustrative schematic of one embodiment of a system including an authoring platform and a server for providing programming instructions to a device over a network;

FIG. 1B is schematic of an alternative embodiment system for providing programming instructions to device over a network;

FIG. 2A is a schematic of an embodiment of system illustrating the communications between different system components;

FIG. 2B is a schematic of one embodiment of a device illustrating an embodiment of the programming generated by authoring platform;

FIGS. 3A and 3B illustrate one embodiment of a publisher interface as it appears, for example and without limitation, on a screen while executing an authoring tool;

FIG. 3C illustrates an embodiment of the Events Tab, FIG. 3D illustrates one embodiment of an Animation Tab; FIG. 3E illustrates one embodiment of Bindings Tab; FIG. 3F illustrates one embodiment of a pop-up menu for adding web components;

FIG. 4A shows a publisher interface having a layout on a canvas; and FIG. 4B shows a device having the resulting layout on a device screen;

FIG. 5 shows a display of launch strips;

FIG. 6A is a display of a Channel Selection List;

FIG. 6B is a display of a Widget Selection List;

FIG. 6C is a display of a Phone List;

FIG. 7 shows a display of a mash-up;

FIG. 8 is a schematic of an embodiment of a push capable system;

FIG. 9 is a schematic of an alternative embodiment of a push capable system;

FIG. 10 is a schematic of one embodiment of a feed collector;

FIG. 11 is a schematic of an embodiment of a Mobile Content Gateway;

FIG. 12 is a schematic of one embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database; and

FIG. 13 is a schematic of another embodiment of a system that includes a response director, a user agent database, an IP address database, and a file database.

Reference symbols are used in the Figures to indicate certain components, aspects or features shown therein, with reference symbols common to more than one Figure indicating like components, aspects or features shown therein.

MODE(S) FOR CARRYING OUT THE INVENTION

FIG. 1A is an illustrative schematic of one embodiment of a system 100 including an authoring platform 110 and a server 120 for providing programming instructions to a

3

device **130** over a network **N**. In one embodiment, device **130** is a wireless device, and network **N** includes wireless communication to the device. Alternatively, system **100** may provide access over network **N** to other information, data, or content, such as obtainable as a web service over the Internet. In general, a user of authoring platform **110** may produce programming instructions or files that may be transmitted over network **N** to operate device **130**, including instructions or files that are sent to device **130** and/or server **120**. The result of the authoring process is also referred to herein, and without limitation, as publishing an Application.

Embodiments include one or more databases that store information related to one or more devices **130** and/or the content provided to the devices. It is understood that such databases may reside on any computer or computer system on network **N**, and that, in particular, the location is not limited to any particular server, for example.

Device **130** may be, for example and without limitation, a cellular telephone or a portable digital assistant, includes a network interface **131**, a memory **133**, a processor **135**, a screen **137**, and an input device **139**. Network interface **131** is used by device **130** to communication over a wireless network, such as a cellular telephone network, a WiFi network or a WiMax network, and then to other telephones through a public switched telephone network (PSTN) or to a satellite, or over the Internet. Memory **133** includes programming required to operate device **130** (such as an operating system or virtual machine instructions), and may include portions that store information or programming instructions obtained over network interface **131**, or that are input by the user (such as telephone numbers or images from a device camera (not shown)). In one embodiment screen **137** is a touch screen, providing the functions of the screen and input device **139**.

Authoring platform **110** includes a computer or computer system having a memory **111**, a processor **113**, a screen **115**, and an input device **117**. It is to be understood that memory **111**, processor **113**, screen **115**, and input device **117** are configured such a program stored in the memory may be executed by the processor to accept input from the input device and display information on the screen. Further, the program stored in memory **111** may also instruct authoring platform **110** to provide programming or information, as indicated by the line labeled "A" and to receive information, as indicated by the line labeled "B."

Memory **111** is shown schematically as including a stored program referred to herein, and without limitation, as an authoring tool **112**. In one embodiment, authoring tool **112** is a graphical system for designing the layout of features as a display that is to appear on screen **137**. One example of authoring tool **112** is the CDER™ publishing platform (Express Mobile, Inc., Novato, Calif.).

In another embodiment, which is not meant to limit the scope of the present invention, device **130** may include an operating system having a platform that can interpret certain routines. Memory **111** may optionally include programming referred to herein, and without limitation, as routines **114** that are executable on device **130**.

Routines **114** may include device-specific routines—that is, codes that are specific to the operating system, programming language, or platform of specific devices **130**, and may include, but are not limited to, Java, Windows Mobile, Brew, Symbian OS, or Open Handset Alliance (OHA). Several examples and embodiments herein are described with reference to the use of Java. It is to be understood that the invention is not so limited, except as provided in the claims, and that one skilled in the art could provide Players for

4

devices using routines provided on a platform. Thus as an example, routines **114** may include Java API's and an authoring tool System Development Kit (SDK) for specific devices **130**.

Server **120** is a computer or computer system that includes a network interface **121**, a memory **123**, and a processor **125**. Is to be understood that network interface **121**, memory **123**, and processor **125** are configured such that a program stored in the memory may be executed by the processor to: accept input and/or provide output to authoring platform **110**; accept input and/or provide output through network interface **121** over network **N** to network interface **131**; or store information from authoring platform **110** or from device **130** for transmission to another device or system at a later time.

In one embodiment, authoring platform **110** permits a user to design desired displays for screen **137** and actions of device **130**. In other words, authoring platform **110** is used to program the operation of device **130**. In another embodiment, authoring platform **110** allows a user to provide input for the design of one or more device displays and may further allow the user to save the designs as device specific Applications. The Applications may be stored in memory **123** and may then be sent, when requested by device **130** or when the device is otherwise accessible, over network **N**, through network interface **130** for storage in memory **133**.

In an alternative embodiment, analytics information from devices **130** may be returned from device **130**, through network **N** and server **120**, back to authoring platform **110**, as indicated by line B, for later analysis. Analytics information includes, but is not limited to, user demographics, time of day, and location. The type of analytic content is only limited by which listeners have been activated for which objects and for which pages. Analytic content may include, but is not limited to, player-side page view, player-side forms-based content, player-side user interactions, and player-side object status.

Content server **140** is a computer or computer system that includes a network interface **141**, a memory **143**, and a processor **145**. It is to be understood that network interface **141**, memory **143**, and processor **145** are configured such that a stored program in the memory may be executed by the processor to accepts requests **R** from device **130** and provide content **C** over a network, such as web server content the Internet, to device **130**.

FIG. 1B is schematic of an alternative embodiment system **100** for providing programming instructions to device **130** over a network **N** that is generally similar to the system of FIG. 1A. The embodiment of FIG. 1B illustrates that system **100** may include multiple servers **120** and/or multiple devices **130**.

In the embodiment of FIG. 1B, system **100** is shown as including two or more servers **120**, shown illustratively and without limitation as servers **120a** and **120b**. Thus some of the programming or information between authoring platform **110** and one or more devices **130** may be stored, routed, updated, or controlled by more than one server **120**. In particular, the systems and methods described herein may be executed on one or more server **120**.

Also shown in FIG. 1B are a plurality of devices **130**, shown illustratively and without limitation as device **130-1**, **130-1**, . . . **130-N**. System **100** may thus direct communication between individual server(s) **120** and specific device(s) **130**.

As described subsequently, individual devices **130** may be provided with program instructions which may be stored in each device's memory **133** and where the instructions are

5

executed by each device's processor 135. Thus, for example, server(s) 120 may provide device(s) 130 with programming in response to the input of the uses of the individual devices. Further, different devices 130 may be operable using different sets of instructions, that is having one of a variety of different "device platforms." Differing device platforms may result, for example and without limitation, to different operating systems, different versions of an operating system, or different versions of virtual machines on the same operating system. In some embodiments, devices 130 are provided with some programming from authoring system 100 that is particular to the device.

In one embodiment, system 100 provides permits a user of authoring platform 110 to provide instructions to each of the plurality of devices 130 in the form of a device- or device-platform specific instructions for processor 135 of the device, referred to herein and without limitation as a "Player," and a device-independent program, referred to herein and without limitation as an "Application." Thus, for example, authoring platform 110 may be used to generate programming for a plurality of devices 130 having one of several different device platforms. The programming is parsed into instructions used by different device platforms and instructions that are independent of device platform. Thus in one embodiment, device 130 utilizes a Player and an Application to execute programming from authoring platform 110. A device having the correct Player is then able to interpret and be programmed according to the Application.

In one alternative embodiment, the Player is executed the first time by device 130 ("activated") through an Application directory. In another alternative embodiment, the Player is activated by a web browser or other software on device 130. In yet another alternative embodiment, Player is activated through a signal to device 130 by a special telephone numbers, such as a short code.

When the Application and the Player are provided to memory 133, the functioning of device 130 may occur in accordance with the desired programming. Thus in one embodiment, the Application and Player includes programming instructions which may be stored in memory 133 and which, when executed by processor 135, generate the designed displays on screen 137. The Application and Player may also include programming instructions which may be stored in memory 133 and which provide instructions to processor 135 to accept input from input device 139.

Authoring tool 112 may, for example, produce and store within memory 111 a plurality of Players (for different devices 130) and a plurality of Applications for displaying pages on all devices. The Players and Applications are then stored on one or more servers 120 and then provided to individual devices 130. In general, Applications are provided to device 130 for each page of display or a some number of pages. A Player need be provided once or updated as necessary, and thus may be used to display a large number of Applications. This is advantageous for the authoring process, since all of the device-dependent programming is provided to a device only once (or possibly for some small number of upgrades), permitting a smaller Application, which is the same for each device 130.

Thus, for example and without limitation, in one embodiment, the Player transforms device-independent instructions of the Application into device-specific instructions that are executable by device 130. Thus, by way of example and without limitation, the Application may include Java programming for generating a display on screen 137, and the Player may interpret the Java and instruct processor 135 to produce the display according to the Application for execu-

6

tion on a specific device 130 according to the device platform. The Application may in general include, without limitation, instructions for generating a display on screen 137, instructions for accepting input from input device 139, instructions for interacting with a user of device 130, and/or instructions for otherwise operating the device, such as to place a telephone call.

The Application is preferably code in a device-independent format, referred to herein and without limitation as a Portable Description Language (PDL). The device's Player interprets or executes the Application to generate one or more "pages" ("Applications Pages") on a display as defined by the PDL. The Player may include code that is device-specific—that is, each device is provided with a Player that is used in the interpretation and execution of Applications. Authoring tool 112 may thus be used to design one or more device-independent Applications and may also include information on one or more different devices 130 that can be used to generate a Player that specific devices may use to generate displays from the Application.

In one embodiment, system 100 provides Players and Applications to one server 120, as in FIG. 1A. In another embodiment, system 100 provides Players to a first server 120a and Applications to a second server 120b, as in FIG. 1B.

In one embodiment, authoring tool 112 may be used to program a plurality of different devices 130, and routines 114 may include device-specific routines. In another embodiment, the Player is of the type that is commonly referred to as a "thin client"—that is, software for running on the device as a client in client-server architecture with a device network which depends primarily on a central server for processing activities, and mainly focuses on conveying input and output between the user and the server.

In one embodiment, authoring platform 110 allows user to arrange objects for display on screen. A graphical user interface ("GUI," or "UI") is particularly well suited to arranging objects, but is not necessary. The objects may correspond to one or more of an input object, an output object, an action object, or may be a decorative display, such as a logo, or background color or pattern, such as a solid or gradient fill. In another embodiment, authoring platform 110 also permits a user to assign actions to one or more of an input object, an output object, or an action object. In yet another embodiment, authoring platform 110 also permits a user to bind one or more of an input object, an output object, or an action object with web services or web components, or permits a user to provide instructions to processor 135 to store or modify information in memory 133, to navigate to another display or service, or to perform other actions, such as dialing a telephone number.

In certain embodiments, the applicant model used in developing and providing Applications is a PDL. The PDL can be conceptually viewed as a device, operating system and virtual machine agnostic representation of Java serialized objects. In certain embodiments, the PDL is the common language for authoring tool 112, the Application, and Player. Thus while either designing the Application with the authoring tool 112, or programming with the SDK, the internal representation of the programming logic is in Java. In one embodiment the SDK is used within a multi-language software development platform comprising an IDE and a plug-in system to extend it, such as the Eclipse Integrated Development Environment (see, for example, <http://www.eclipse.org/>). At publish time the Java code is translated into

7

a PDL. This translation may also occur in real-time during the execution of any Web Services or backend business logic that interacts with the user.

One embodiment for compacting data that may be used is described in co-pending U.S. Pat. No. 6,546,397 to Rempell ("Rempell"), the contents of which are incorporated herein by reference. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

The use of a PDL, as described in Rempell, permits for efficient code and data compaction. Code, as well as vector, integer and Boolean data may be compacted and then compressed resulting in a size reduction of 40 to 80 times that of the original Java serialized objects. This is important not only for performance over the network but for utilizing the virtual memory manager of the Player more efficiently. As an example, the reassembled primitives of the Java objects may first undergo logical compression, followed by LZ encoding.

The use of a PDL also provides virtual machine and operating system independence. Since the reassembled primitives of the Application no longer have any dependencies from the original programming language (Java) that they were defined in. The PDL architecture takes full advantage of this by abstracting all the virtual machine and/or operating system interfaces from the code that processes the PDL.

In one embodiment, the PDL is defined by the means of nested arrays of primitives. Accordingly, the use of a PDL provides extensibility and compatibility, with a minimal amount of constraints in extending the Player seamlessly as market demands and device capabilities continue to grow. Compatibility with other languages is inherent based on the various Player abstraction implementations, which may be, for example and without limitation, Java CDC, J2SE or MIDP2 implementations.

In one embodiment, the architecture of Player P includes an abstraction interface that separates all device, operating system and virtual machine dependencies from the Player's Application model business logic (that is, the logic of the server-side facilities) that extend the Application on the Player so that it is efficiently integrated into a comprehensive client/server Application. The use of an abstraction interface permits the more efficient porting to other operating systems and virtual machines and adding of extensions to the Application model so that a PDL can be implemented once and then seamlessly propagated across all platform implementations. The Application model includes all the currently supported UI objects and their attributes and well as all of the various events that are supported in the default Player. Further, less robust platforms can be augmented by extending higher end capabilities inside that platform's abstraction interface implementation.

In one embodiment, authoring platform 110 provides one or more pages, which may be provided in one Application, or a plurality of Applications, which are stored in memory 123 and subsequently provided to memory 133. In certain embodiments, the Application includes instructions R to request content or web services C from content server 140. Thus, for example and without limitation, the request is for information over the network via a web service, and the request R is responded to with the appropriate information for display on device 130. Thus, for example, a user may request a news report. The Application may include the layout of the display, including a space for the news, which is downloaded from content server 140 for inclusion on the

8

display. Other information that may be provided by content server 140 may include, but is not limited to, pages, Applications, multimedia, and audio.

FIG. 2A is a schematic of a system 200 of an embodiment of system 100 illustrating the communications between different system components. System includes a response director 210, a web component registry 220, and a web service 230. System 200 further includes authoring platform 110, server 120, device 130 and content server 140 are which are generally similar to those of the embodiments of FIGS. 1A and 1B, except as explicitly noted.

Response director 210 is a computer or computer system that may be generally similar to server 120 including the ability to communicate with authoring platform 110 and one or more devices 130. In particular, authoring platform 110 generates one or more Players (each usable by certain devices 130) which are provided to response director 210. Devices 130 may be operated to provide response director 210 with a request for a Player and to receive and install the Player. In one embodiment, device 130 provides response director 210 with device-specific information including but not limited to make, model, and/or software version of the device. Response director 210 then determines the appropriate Player for the device, and provides the device with the Player over the network.

Web service 230 is a plurality of services obtainable over the Internet. Each web service is identified and/or defined as an entry in web component registry 230, which is a database, XML file, or PDL that exists on a computer that may be a server previously described or another server 120. Web component registry 230 is provided through server 120 to authoring platform 110 so that a user of the authoring platform may bind web services 230 to elements to be displayed on device 130, as described subsequently.

In one embodiment, authoring platform 110 is used in conjunction with a display that provides a WYSIWYG environment in which a user of the authoring platform can produce an Application and Player that produces the same display and the desired programming on device 130. Thus, for example, authoring tool 112 provides a display on screen 115 that corresponds to the finished page that will be displayed on screen 137 when an Application is intercepted, via a Player, on processor 135 of device 130.

Authoring platform 110 further permits a user of the authoring platform to associate objects, such as objects for presenting on screen 137, with components of one or more web services 230 that are registered in web component registry 220. In one embodiment, information is provided in an XML file to web component registry 220 for each registered components of each web service 230. Web component registry 220 may contain consumer inputs related to each web service 230, environmental data such as PIM, time or location values, persistent variable data, outputs related to the web service, and/or optional hinting for improving the user's productivity.

A user of authoring platform 110 of system 200 may define associations with web services as WebComponent Bindings. In one embodiment, authoring platform 110 allows a user to associate certain objects for display that provide input or output to components of web service 230. The associated bindings are saved as a PDL in server 120.

In one embodiment, an XML web component registry 220 for each registered web service 230 is loaded into authoring platform 110. The user of system 200 can then assign components of any web service 230 to an Application without any need to write code. In one embodiment, a component of web service 230 is selected from authoring

platform **110** which presents the user with WYSIWYG dialog boxes that enable the binding of all the inputs and outputs of component of web service **230** to a GUI component of the Application as will be displayed on screen **137**. In addition, multiple components of one or more web service **230** can be assigned to any Object or Event in order to facilitate mashups. These Object and/or Event bindings, for each instance of a component of any web service **230**, are stored in the PDL. The content server **140** handles all communication between device **130** and the web service **230** and can be automatically deployed as a web application archive to any content server.

Device **130**, upon detecting an event in which a component of a web service **230** has been defined, assembles and sends all related inputs to content server **240**, which proxies the request to web service **230** and returns the requested information to device **130**. The Player on device **130** then takes the outputs of web service **230** and binds the data to the UI components in the Application, as displayed on screen **137**.

In one embodiment, the mechanism for binding the outputs of the web service to the UI components is through symbolic references that matches each output to the symbolic name of the UI component. The outputs, in one embodiment, may include meta-data which could become part of the inputs for subsequent interactions with the web service.

For example, if a user of authoring platform **110** wants to present an ATOM feed on device **130**, they would search through a list of UI Components available in the authoring platform, select the feed they want to use, and bind the output of the feed summary to a textbox. The bindings would be saved into the PDL on server **120** and processed by device **130** at runtime. If the ATOM feed does not exist a new one can be added to the web component registry that contains all the configuration data required, such as the actual feed URL, the web component manager URL, and what output fields are available for binding.

In another embodiment, components of web services **230** are available either to the user of authoring platform **110** or otherwise accessible through the SDK and Java APIs of routines **114**. System **200** permits an expanding set of components of web services **230** including, but not limited to: server pages from content server **120**; third-party web services including, but not limited to: searching (such through Google or Yahoo), maps (such as through MapQuest and Yahoo), storefronts (such as through ThumbPlay), SMS share (such as through clickatel), stock quotes, social networking (such as through FaceBook), stock quotes, weather (such as through Accuweather) and/or movie trailers. Other components include web services for communication and sharing through chats and forums and rich messaging alerts, where message alerts are set-up that in turn could have components of Web Services **230** defined within them, including the capture of consumer generated and Web Service supplied rich media and textual content.

System **200** also permits dynamic binding of real-time content, where the inputs and outputs of XML web services are bound to GUI components provided on screen **137**. Thus, for example, a user of authoring platform **110** may bind attributes of UI Objects to a particular data base field on a Server. When running the Application, the current value in the referenced data base will be immediately applied. During the Application session, any other real time changes to these values in the referenced data base will again be immediately displayed.

As an example of dynamic binding of real-time content, an RSS feeds and other forms of dynamic content may be inserted into mobile Applications, such as device **130**, using system **200**. Authoring platform **110** may include a "RSS display" list which permits a user to select RSS channels and feeds from an extensible list of available dynamic content. Meta data, such as titles, abstracts and Images can be revealed immediately by the user as they traverse this RSS display list, bringing the PC experience completely and conveniently to mobile devices **130**. In addition, Authoring platform **110** may include a dialog box that dynamically links objects to data and feeds determined by RSS and chat databases. Any relevant attribute for a page view and/or object can be dynamically bound to a value in a server-side database. This includes elements within complex objects such as: any icon or text element within a graphical list; any icon within a launch strip; any feature within any geographical view of a GIS service object; and/or any virtual room within a virtual tour.

As an example of third-party web services **230** that may be provided using system **200**, a user of authoring platform **110** can place, for example, Yahoo maps into device **130** by binding the required component of the Yahoo Maps Web Service, such as Yahoo Map's Inputs and/or Outputs to appropriate Objects of authoring platform **110**. System **200** also provides binding to web services for text, image and video searching by binding to components of those web services.

In one embodiment, an Application for displaying on device **130** includes one or more Applications Pages, each referred to herein as an "XSP," that provides functionality that extends beyond traditional web browsers. The XSP is defined as a PDL, in a similar manner as any Application, although it defines a single page view, and is downloaded to the Player dynamically as required by the PDL definition of the Application. Thus, for example, while JSPs and ASPs, are restricted to the functionality supported by the web browser, the functionality of XSPs can be extended through authoring platform **110** having access to platform dependent routines **114**, such as Java APIs. Combined with dynamic binding functionality, an XSP, a page can be saved as a page object in an author's "pages" library, and then can be dynamically populated with real-time content simultaneously as the page is downloaded to a given handset Player based on a newly expanded API. XSP Server Pages can also be produced programmatically, but in most cases authoring platform **110** will be a much more efficient way to generate and maintain libraries of dynamically changing XSPs.

With XSPs, Applications Pages that have dynamic content associated with them can be sent directly to device **130**, much like how a web browser downloads an HTML page through an external reference. Without XSPs, content authors would have to define each page in the Application. With XSPs, no pages need to be defined. Thus, for example, in a World Cup Application, one page could represent real-time scores that change continuously on demand. With polling (for example, a prompt to the users asking who they predict will win a game), a back-end database would tabulate the information and then send the results dynamically to the handsets. With a bar chart, the Application would use dynamic PDL with scaling on the fly. For example, the server would recalibrate the bar chart for every ten numbers.

Other combinations of components of web services **230** include, but are not limited to, simultaneous video chat sessions, inside an integrated page view, with a video or television station; multiple simultaneous chat sessions, each

11

with a designated individual and/or group, with each of the chat threads visible inside an integrated page view.

Another extension of an XSP is a widget object. Widgets can be developed from numerous sources including, but not limited to, authoring platform 110, a Consumer Publishing Tool, and an XML to Widget Conversion Tool where the SDK Widget Libraries are automatically populated and managed, or Widget Selection Lists that are available and can be populated with author defined Icons.

Applications, Players, and Processing in a Device

FIG. 2B is a schematic of one embodiment of a device 130 illustrating an embodiment of the programming generated by authoring platform 110. Memory 133 may include several different logical portions, such as a heap 133a, a record store 133b and a filesystem (not shown).

As shown in FIG. 2B, heap 133a and record store 133b include programming and/or content. In general, heap 133a is readily accessible by processor 135 and includes, but is not limited to portions that include the following programming: a portion 133a1 for virtual machine compliant objects representing a single Page View for screen 137; a portion 133a2 for a Player; a portion 133a3 for a virtual machine; and a portion 133a4 for an operating system.

Record store 133b (or alternatively the filesystem) includes, but is not limited to, portions 133b1 for Applications and non-streaming content, which may include portions 133a2 for images, portions 133a4 for audio, and/or portions 133a5 for video. and portions 133b2 for non-Application PDLs, such as a Master Page PDL for presenting repeating objects, and Alerts, which are overlayed on the current page view. Other content, such as streaming content may be provided from network interface 131 directly to the Media Codec of device 130 with instructions from Player on how to present the audio or video.

In one embodiment, the Player includes a Threading Model and a Virtual Memory Manager. The Threading Model first manages a queue of actions that can be populated based on Input/Output events, Server-side events, time-based events, or events initiated by user interactions. The Threading Model further manages the simultaneous execution of actions occurring at the same time. The Virtual Memory Manager includes a Logical Virtual Page controller that provides instructions from the record store to the heap, one page at time. Specifically, the Virtual Memory Manager controls the transfer of one of the Application Pages and its virtual machine compliant objects into portion 133a1 as instructions readable by the Player or Virtual Machine. When the Player determines that a new set of instructions is required, the information (such as one Application Page is retrieve from the Record store, converted into virtual machine compliant objects (by processor 135 and according to operation by the Player, Virtual Machine, etc). and stored in heap 133a. Alternatively, the Player may augment virtual machine compliant objects with its own libraries for managing user interactions, events, memory, etc.

The connection of portions 133a1, 133a2, 133a3, 133a4, record store 133b and processor 135 are illustrative of the logical connection between the different types of programming stored in Heap 133a and record store 133b, that is, how data is processed by processor 135.

The Player determines which of the plurality of Application Pages in portion 133b1 is required next. This may be determined by input actions from the Input Device 139, or from instructions from the current Application Page. The Player instructs processor 135 to extract the PDF from that Applications Page and store it in portion 133a1. The Player then interprets the Application Page extracted from PDL

12

which in turn defines all of the virtual machine compliant Objects, some of which could have attributes that refer to images, audio, and/or video stored in portions 133a3, 133a4, 133a5, respectively.

The Virtual Machine in portion 133a3 processes the Player output, the Operating System in portion 133a3 processes the Virtual Machine output which results in machine code that is processed by the Operating System in portion 133a4.

In another embodiment, the Player is a native program that interacts directly with the operating system.

Embodiments of a Publishing Environment

In one embodiment, authoring platform 110 includes a full-featured authoring tool 112 that provides a what-you-see-is-what-you-get (WYSIWYG) full featured editor. Thus, for example, authoring tool 112 permits a user to design an Application by placing objects on canvas 305 and optionally assigning actions to the objects and save the Application. System 100 then provides the Application and Player to a device 130. The Application as it runs on device 130 has the same look and operation as designed on authoring platform 110. In certain embodiments, authoring platform 110 is, for example and without limitation, a PC-compatible or a Macintosh computer.

Authoring platform 110 produces an Application having one or more Applications Pages, which are similar to web pages. That is, each Applications Page, when executed on device 130 may, according to its contents, modify what is displayed on screen 137 or cause programming on the device to change in a manner similar to how web pages are displayed and navigated through on a website.

In one embodiment, authoring tool 112 allows a user to place one or more objects on canvas 305 and associate the objects with an Applications Pages. Authoring platform 110 maintains a database of object data in memory 111, including but not limited to type of object, location on which page, and object attributes. The user may add settings, events, animations or binding to the object, from authoring tool 112, which are also maintained in memory 111. Authoring tool 112 also allows a user to define more than one Applications Page.

In another embodiment, authoring tool 112, provides Java programming functions of the Java API for specific devices 130 as pull-down menus, dialog boxes, or buttons. This permits a user of authoring platform 110 to position objects that, after being provided as an Application to device 130, activate such Java functions on the device.

In certain embodiments, authoring platform 110, as part of system 100, permits designers to include features of advanced web and web services Applications for access by users of device 130. Some of the features of advanced web and web services include, but are not limited to: slide shows, images, video, audio, animated transitions, multiple chats, and mouse interaction; full 2-D vector graphics; GIS (advanced LBS), including multiple raster and vector layers, feature sensitive interactions, location awareness, streaming and embedded audio/video, virtual tours, image processing and enhancement, and widgets. In other embodiments the features are provided for selection in authoring platform 110 through interactive object libraries.

In certain embodiments, authoring platform 110, as part of system 100, allows the inclusion of child objects which may eventually be activated on device 130 by the user of the device or by time. The uses of the child objects on device 130 include, but are not limited to: mouse over (object selection), hover and fire events and launching of object-specific, rich-media experiences.

13

In certain other embodiments, authoring platform 110, as part of system 100, provides advanced interactive event models on device 130, including but not limited to: user-, time- and/or location-initiated events, which allow content developers to base interactivity on specific user interactions and/or instances in time and space; timelines, which are critical for timing of multiple events and for animations when entering, on, or exiting pages of the Application; waypoints, which act similar to key frames, to allow smooth movement of objects within pages of the Application. Waypoints define positions on a page object's animation trajectory. When an object reaches a specific waypoint other object timelines can be initiated, creating location-sensitive multiple object interaction, and/or audio can be defined to play until the object reaches the next waypoint.

Authoring platform 110 may also define a Master Page, which acts as a template for an Applications Page, and may also define Alert Pages, which provide user alerts to a user of device 130.

In certain embodiments, authoring platform 110, as part of system 100, provides full style inheritance on device 130. Thus, for example and without limitation, both master page inheritance (for structural layout inheritance and repeating objects) and object styles (for both look and feel attribute inheritance) are supported. After a style has been defined for an object, the object will inherit the style. Style attributes include both the look and the feel of an object, including mouse interaction, animations, and timelines. Each page may include objects that may be a parent object or a child object. A child object is one that was created by first selecting a parent object, and then creating a child object. Child objects are always part of the same drawing layer as its parent object, but are drawn first, and are not directly selectable when running the Application. A parent object is any object that is not a child object, and can be selected when running the Application.

As an example, the user of authoring tool 112 may create various text objects on canvas 305 using a style that sets the font to red, the fonts of these objects will be red. Suppose user of authoring tool 112 changes the font color of a specific button to green. If later, the user of authoring tool 112 changes the style to blue; all other text objects that were created with that style will become blue except for the button that had been specifically set to green.

In certain other embodiments, authoring platform 110 provides page view, style, object, widget and Application template libraries. Authoring platform 110 may provide templates in private libraries (available to certain users of the authoring platform) and public libraries (available to all users of the authoring platform). Templates may be used to within authoring platform 110 to define the look and feel of the entire Application, specific pages, or specific slide shows and virtual tours as seen on device 130.

FIGS. 3A and 3B illustrate one embodiment of a publisher interface 300 as it appears, for example and without limitation, on screen 115 while executing authoring tool 112. In one embodiment, publisher interface 300 includes a Menu bar 301, a Tool bar 303, a Canvas 305, a Layer Inspector 307 having subcomponents of a page/object panel 307a, an object style panel 307b, and a page alert panel 307c, and a Resource Inspector 309.

In general, publisher interface 300 permits a user of authoring platform 110 to place objects on canvas 305 and then associate properties and/or actions to the object, which are stored in the Application. As described subsequently, publisher interface 300 permits a user to program a graphical interface for the screen 137 of device 130 on screen 115 of

14

authoring platform 110, save an Application having the programming instructions, and save a Player for the device. The intended programming is carried out on device 130 when the device, having the appropriate device platform Player, receives and executes the device-independent Application.

Thus, for example, authoring tool 112 maintains, in memory 111, a list of every type of object and any properties, actions, events, or bindings that may be assigned to that object. As objects are selected for an Application, authoring tool 112 further maintains, in memory 111, a listing of the objects. As the user selects objects, publisher interface 300 provides the user with a choice of further defining properties, actions, events, or bindings that may be assigned to each particular object, and continues to store the information in memory 111.

In one embodiment, publisher interface 300 is a graphical interface that permits the placement and association of objects in a manner typical of, for example, vector graphics editing programs (such as Adobe Illustrator). Objects located on canvas 305 placed and manipulated by the various commands within publisher interface 300 or inputs such as an input device 117 which may be a keyboard or mouse. As described herein, the contents of canvas 305 may be saved as an Application that, through system 100, provide the same or a similar placement of objects on screen 137 and have actions defined within publisher interface 300. Objects placed on canvas 305 are intended for interaction with user of device 130 and are referred to herein, without limitation, as objects or UI (user interface) objects. In addition, the user of interface 300 may assign or associate actions or web bindings to UI objects placed on canvas 305 with result in the programming device 130 that cause it to respond accordingly.

Objects include, but are not limited to input UI objects, response UI objects. Input UI objects include but are not limited to: text fields (including but not limited to alpha, numeric, phone number, or SMS number); text areas; choice objects (including but not limited to returning the selected visible string or returning a numeric hidden attribute); single item selection lists (including but not limited to returning the selected visible string or returning a numeric hidden attribute); multi item selection lists (including but not limited to returning all selected items (visible text string or hidden attribute) or cluster item selection lists (returning the hidden attributes for all items)).

Other input UI objects include but are not limited to: check boxes; slide show (including but not limited to returning a numeric hidden attribute, returning a string hidden attribute, or returning the hidden attributes for all slides); and submit function (which can be assigned to any object including submit buttons, vectors, etc.).

Response UI Objects may include, but are not limited to: single line text objects, which include: a text Field (including but not limited to a URL, audio URL, or purchase URL), a text button, a submit button, or a clear button. Another response UI objects include: a multiple line text object, which may include a text area or a paragraph; a check box; an image; a video; a slide show (with either video or image slides, or both); choice objects; list objects; or control lists, which control all the subordinate output UI objects for that web component. Control list objects include, but are not limited to: list type or a choice type, each of which may include a search response list or RSS display list.

As a further example of objects that may be used with authoring tool 112, Table I lists Data Types, Preferred Input, Input Candidates, Preferred Output and Output Candidates for one embodiment of an authoring tool.

TABLE I

One embodiment of supported objects				
Data Types	Preferred Input	Input Candidates	Preferred Output	Output Candidates
boolean	Check Box	Check Box	Check Box	Check Box
Int	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select)	Text Field (integer)	Text Field (integer) Text Field (Phone #) Text Field (SMS #) Choice List (single select) Text Button
String	Text Field (Alpha)	Any	Text Field (Alpha)	Any
multilineString	Text Area	Text Area	Text Area	Text Area Paragraph
ImageURL	N/A	N/A	Image	Image Slide Show
VideoURL	N/A	N/A	Video	Video Slide Show
List	Single Item List	Single Item List Multi-Select List Complex List Choice Slide Show	Single Item List	Any List Type Any Choice Type (see Complex List Specification)
ComplexList	Complex List	Single Item List Multi-Select List Complex List	Single Item List	Any List Type (see Complex List Specification)
Slideshow	Slide Show	Slide Show	Slide Show	Slide Show
SearchResponseList	N/A	N/A	Search Response List	Search Response List Control List Complex List Choice
RSSList	N/A	N/A	RSS Display List	RSS Display List Control List Complex List Choice
SingleSelectionList	Choice	Choice Complex List	Choice	Choice Complex List
MultiSelectionList	Multi-Selection List	Multi-Selection List	Multi-Selection List	Multi-Selection List
ServiceActivation	Submit Button	Any	N/A	N/A
ChannelImageURL	N/A	N/A	Image	Image Video Slide Show
ChannelDescription	N/A	N/A	Text Area	Text Area Paragraph Text Field Text Button List Choice
ChannelTitle	N/A	N/A	Text Field	Text Field Text Button Paragraph Text Area List Choice
URL			Text Field (URL request)	Text Field (URL request)
Audio URL			Text Field (Audio URL request)	Text Field (Audio URL request)
Purchase URL			Text Field (Purchase URL request)	Text Field (Purchase URL request)
Image Data			Image	Image Slide Show
Image List Data			Slide Show	Slide Show Image
Persistent Variable	N/A	N/A	N/A	N/A
Pipeline Multiple Select	Multi-select List	Multi-select List Complex List Slide Show	N/A	N/A
Phone Number	Text Field (numeric type)	Text Field Text Button	Text Field (numeric type)	Text Field Text Button
Hidden Attribute	Complex List	Complex List Slide Show	Complex List	Complex List Slide Show
Collection List	N/A	N/A	Slide Show	Complex List Slide Show

17

In general, publisher interface **300** permits a user to define an Application as one or more Applications Pages, select UI objects from Menu bar **301** or Tool bar **303** and arrange them on an Applications Page by placing the objects canvas **305**. An Application Page is a page that is available to be visited through any navigation event. Application Pages inherit all the attributes of the Master Page, unless that attribute is specifically changed during an editing session.

Authoring platform **110** also stores information for each UI object on each Application Page of an Application. Layer Inspector **307** provides lists of Applications Pages, UI objects on each Applications Page, and Styles, including templates. Objects may be selected from canvas **305** or Layer Inspector **307** causing Resource Inspector **309** to provide lists of various UI objects attributes which may be selected from within the Resource Inspector. Publisher interface **300** also permits a user to save their work as an Application for layer transfer and operation of device **130**. Publisher interface **300** thus provides an integrated platform for designing the look and operation of device **130**.

The information stored for each UI object depends, in part, on actions which occur as the result of a user of device **130** selecting the UI object from the device. UI objects include, but are not limited to: navigational objects, such as widget or channel launch strips or selection lists; message objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields or a pop-up alert; text fields or areas; check boxes; pull down menus; selection lists and buttons; pictures; slide shows; video or LBS maps; shapes or text defined by a variety of tools; a search response; or an RSS display.

In certain embodiments, publisher interface **300** permits a user to assign action to UI objects, including but not limited to, programming of the device **130** or a request for information over network N. In one embodiment, for example and without limitation, publisher interface **300** has a selection to bind a UI object to a web service—that is, associate the UI object or a manipulation or selection of UI object with web services. Publisher interface **300** may also include many drawing and text input functions for generating displays that may be, in some ways, similar to drawing and/or word processing programs, as well as toolbars and for zooming and scrolling of a workspace.

Each UI object has some form, color, and display location associate with it. Further, for example and without limitation, UI objects may have navigational actions (such as return to home page), communications actions (such as to call the number in a phone number field), or web services (such as to provide and/or retrieve certain information from a web service). Each of these actions requires authoring platform **110** to store the appropriate information for each action. In addition, UI objects may have associated patent or child objects, default settings, attributes (such as being a password or a phone number), whether a field is editable, animation of the object, all of which may be stored by authoring platform **110**, as appropriate.

Menu bar **301** provides access features of publisher interface **300** through a series of pull-down menus that may include, but are not limited to, the following pull-down menus: a File menu **301a**, an Edit menu **301b**, a View menu **301c**, a Project menu **301d**, an Objects menu **301e**, an Events menu **301f**, a Pages menu **301g**, a Styles menu **301h**, and a Help menu **301i**.

File menu **301a** provides access to files on authoring platform **110** and may include, for example and without limitation, selections to open a new Application or master page, open a saved Application, Application template, or

18

style template, import a page, alert, or widget, open library objects including but not limited to an image, video, slide show, vector or list, and copying an Application to a user or to Server **120**.

Edit menu **301b** may include, but is not limited to, selections for select, cut, copy, paste, and edit functions.

View menu **301c** may include, but is not limited to, selections for zooming in and out, previewing, canvas **305** grid display, and various palette display selections.

Project menu **301d** may include, but is not limited to, selections related to the Application and Player, such as selections that require a log in, generate a universal Player, generate server pages, activate server APIs and extend Player APIs. A Universal Player will include all the code libraries for the Player, including those that are not referenced by the current Application. Server APIs and Player APIs logically extend the Player with Server-side or device-side Application specific logic.

Objects menu **301e** includes selections for placing various objects on canvas **305** including, but not limited to: navigation UI objects, including but not limited to widget or channel launch strips or selection lists; message-related UI objects, including but not limited to multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert; shapes, which provides for drawing tools; forms-related objects, including but not limited to text fields; scrolling text box, check box, drop-down menu, list menu, submit button or clear button; media-related UI objects such as pictures, slide shows, video or LBS maps; text-related UI objects such as buttons or paragraphs; and variables, including but not limited to time, date and audio mute control.

Events menu **301f** includes selections for defining child objects, mouse events, animations or timelines.

Pages menu **301g** includes selection for handling multi-page Applications, and may include selections to set a master page, delete, copy, add or go to Applications Pages.

Styles menu **301h** includes selections to handle styles, which are the underlying set of default appearance attributes or behaviors that define any object that is attached to a style. Styles are a convenient way for quickly creating complex objects, and for changing a whole collection of objects by just modifying their common style. Selections of Styles menu **301h** include, but not limited to, define, import, or modify a style, or apply a template. Help menu **301i** includes access a variety of help topics.

Tool bar **303** provides more direct access to some of the features of publisher interface **300** through a series of pull-down menus. Selections under tool bar **303** may include selections to:

- control the look of publisher interface **300**, such as a Panel selection to control the for hiding or viewing various panels on publisher interface **300**;

- control the layout being designed, such as an Insert Page selection to permit a user to insert and name pages;

- control the functionality of publisher interface **300**, such as a Palettes selection to choose from a variety of specialized palettes, such as a View Palette for zooming and controlling the display of canvas **305**, a Command Palette of common commands, and Color and Shape Palettes;

- place objects on canvas **305**, which may include selections such as: a Navigation selection to place navigational objects, such as widget or channel launch strips or selection lists), a Messages selection to place objects for communicating, such as a multiple chat, video chat, phone and/or SMS lists or fields, or a pop-up alert, a Forms selection to place objects such as text fields or

19

areas, check boxes, pull down menus, selection lists, and buttons, a Media selection to place pictures, slide shows, video or LBS maps, and a Shapes selection having a variety of drawing tools, a Text selection for placing text, a search response, or an RSS display, and Palettes.

In one embodiment, Tool bar **303** includes a series of pull-down menus that may include, but are not limited to, items from Menu bar **301** organized in the following pull-down menus: a Panel menu **303a**, an Insert Page menu **303b**, a Navigation menu **303c**, a Messages menu **303d**, a Forms menu **303e**, a Media menu **303f**, a Shapes menu **303g**, a Text menu **303h**, and a Palettes menu **301i**.

Panel menu **303a** permits a user of authoring platform **110** to change the appearance of interface **300** by, controlling which tools are on the interface or the size of canvas **305**. Insert Page menu **303b** permits a user of authoring platform **110** to open a new Application Page. Navigation menu **303c** displays a drop down menu of navigational-related objects such as a widget or channel launch strip or selection list. Messages menu **303d** displays a drop down menu of messaging-related objects such as multiple chat, video chat, phone or SMS lists or fields, and pop-up alerts. Forms menu **303e** displays a drop down menu of forms-related objects including, but not limited to, a text field, a text area, a check box, a drop down menu, a selection list, a submit button, and a clear button. Media menu **303f** displays a drop down menu of media-related objects including, but not limited to, a picture, slide show, video or LBS map. Shapes menu **303g** displays a drop down menu of draw tools, basic shapes, different types of lines and arrows and access to a shape library. Text menu **303h** displays a drop down menu of text-related objects, including but not limited to a text button, paragraph, search response, RSS display and variables such as time and date.

Palettes menu **301i** includes a selection of different palettes that can be moved about publisher interface **300**, where each palette has specialized commands for making adjustments or associations to objects easier. Palettes include, but are not limited to: a page view palette, to permit easy movement between Applications Pages; a view palette, to execute an Application or zoom or otherwise control the viewing of an Application; a commands palette having editing commands; a color palette for selection of object colors; and a shapes palette to facilitate drawing objects.

Layer inspector **307** permits a user of publisher interface **300** to navigate, select and manipulate UI objects on Applications Pages. Thus, for example, a Page/objects panel **307a** of layer inspector **307** has a listing that may be selected to choose an Applications Pages within an Application, and UI objects and styles within an Applications Page. An Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits selection of UI objects for operations to be performed on the objects.

Thus, for example, when objects from Menu bar **301** or Tool bar **303** are placed on canvas **305**, the name of the object appears in Page/objects panel **307a**. Page/objects panel **307a** includes a page display **307a1** and an objects display **307a2**. Page display **307a1** includes a pull down menu listing all Applications Pages of the Application, and objects display **307a2** includes a list of all objects in the Applications Page (that is, objects on canvas **305**).

In general, page/objects panel **307a** displays various associations with a UI object and permits various manipulations including, but not limited to, operations for parent and child objects that are assigned to a page, and operations for object styles, and permits navigating between page types

20

and object styles, such as switching between the master page and Application pages and deselecting object styles and alerts, opening an Edit Styles Dialog Box and deselecting any master, Application or alert page, or selecting an alert page and deselecting any Master Page or Application Page. A parent or child object can also be selected directly from the Canvas. In either case, the Resource Inspector can then be used for modifying any attribute of the selected object.

Examples of operations provided by page/objects panel **307a** on pages include, but are not limited to: importing from either a user's private page library or a public page library; deleting a page; inserting a new page, inheriting all the attributes of the Master Page, and placing the new page at any location in the Page List; editing the currently selected page, by working with an Edit Page Dialog Box. While editing all the functions of the Resource Inspector **309** are available, as described subsequently, but are not applied to the actual page until completing the editing process.

Examples of operations provided by of page/objects panel **307a** on objects, which may be user interface (UI) objects, include but are not limited to: changing the drawing order layer to: bring to the front, send to the back, bring to the front one layer, or send to the back one layer; hiding (and then reshowing) selected objects to show UI objects obstructed by other UI Objects, delete a selected UI Page Object, and editing the currently selected page, by working with a Edit Page Dialog Box.

Object styles panel **307b** of layer inspector **307** displays all styles on the Applications Page and permits operations to be performed on objects, and is similar to panel **307a**. Examples of operations provided by object style panel **307b** include, but are not limited to: importing from either a user's private object library or a public object library; inserting a new object style, which can be inherited from a currently selected object, or from a previously defined style object; and editing a currently selected object style by working with an Edit Style Dialog Box.

Style attributes can be assigned many attributes, including the look, and behavior of any object that inherits these objects. In addition, List Layout Styles can be created or changed as required. A layout style can define a unbounded set of Complex List Layouts, including but not limited to: the number of lines per item in a list, the number of text and image elements and their location for each line for each item in the last, the color and font for each text element, and the vertical and horizontal offset for each image and text element.

Alerts Panel **307c** provides a way of providing alert pages, which can have many of the attributes of Application Pages, but they are only activated through an Event such as a user interaction, a network event, a timer event, or a system variable setting, and will be superimposed onto whatever is currently being displayed. Alert Pages all have transparent backgrounds, and they function as a template overlay, and can also have dynamic binding to real time content.

Resource inspector **309** is the primary panel for interactively working with UI objects that have been placed on the Canvas **305**. When a UI object is selected on Canvas **305**, a user of authoring platform **110** may associate properties of the selected object by entering or selecting from resource inspector **309**. In one embodiment, resource inspector **309** includes five tab selections: Setting Tab **309a**, Events Tab **309b**, Animation Tab **309c**, Color Tab **309d** which includes a color palette for selecting object colors, and Bindings Tab **309e**.

Settings Tab **309a** provides a dialog box for the basic configuration of the selected object including, but not lim-

ited to, name, size, location, navigation and visual settings. Depending upon the type of object, numerous other attributes could be settable. As an example, the Setting Tab for a Text Field may include dialog boxes to define the text field string, define the object style, set the font name, size and effects, set an object name, frame style, frame width, text attributes (text field, password field, numeric field, phone number, SMS number, URL request).

As an example of Setting Tab **309a**, FIG. 3B shows various selections including, but not limited to, setting **309a1** for the web page name, setting **309a2** for the page size, including selections for specific devices **130**, setting **309a3** indicating the width and height of the object, and setting **309a4** to select whether background audio is present and to select an audio file.

FIG. 3C illustrates an embodiment of the Events Tab **309b**, which includes all end user interactions and time based operations. The embodiment of Events Tab **309b** in FIG. 3C includes, for example and without limitation, an Events and Services **309b1**, Advanced Interactive Settings **309b2**, Mouse State **309b3**, Object Selected Audio Setting **309b4**, and Work with Child Objects and Mouse Overs button **309b5**.

Events and Services **309b1** lists events and services that may be applied to the selected objects. These include, but are not limited to, going to external web pages or other Applications pages, either as a new page or by launching a new window, executing an Application or JavaScript method, pausing or exiting, placing a phone call or SMS message, with or without single or multiple Player download, show launch strip, or go back to previous page. Examples of events and services include, but are not limited to those listed in Table II

TABLE II

Events and Services	
Goto External Web Page replacing Current Frame	ChoiceObject: Remove Icon from Launch Strip
Goto External Web Page Launched in a New Window	Goto a specific Internal Web Page with Alert. "Backend Synchronization"
Goto a specific Internal Web Page	Goto Widget Object
Goto the next Internal Web Page	Generate Alert. "With a Fire Event"
Goto External Web Page replacing the Top Frame	Send SMS Message from Linked Text Field
Execute JavaScript Method	Toggle Alert. "Display OnFocus, Hide OffFocus"
Pause/Resume Page Timeout	Execute an Application with Alert. "With a Fire Event"
Execute an Application	Goto Logical First Page
Goto a specific Internal Web Page with setting starting slide	Generate Alert with Backend Synchronization
Exit Application	Send SMS Message with Share (Player Download)
Exit Player	Place PhoneCall from linked Text Field with Share (Player Download)
Place PhoneCall from linked Text Field	Send IM Alert from linked Text Field or Text Area
Text Field/Area: Send String on FIRE	Set and Goto Starting Page
ChoiceObject: Add Icon to Launch Strip	Populate Image
Text Field/Area: Send String on FIRE or Numeric Keys	Preferred Launch Strip

Advanced Interactive Settings **309b2** include Scroll Activation Enabled, Timeline Entry Suppressed, Enable Server Listener, Submit Form, Toggle Children on FIRE, and Hide Non-related Children, Mouse State **309b3** selections are Selected or Fire. When Mouse State Selected is chosen,

Object Selected Audio Setting **309b4** of Inactive, Play Once, Loop, and other responses are presented. When Mouse State Fire is chosen, Object Selected Audio Setting **309b4** is replaced with FIRE Audi Setting, with appropriate choices presented.

When Work with Child Objects and Mouse Overs button **309b5** is selected, a Child Object Mode box pops up, allowing a user to create a child object with shortcut to Menu bar **301** actions that may be used define child objects.

FIG. 3D illustrates one embodiment of an Animation Tab **309c**, which includes all animations and timelines. The Color Tab includes all the possible color attributes, which may vary significantly by object type.

Animation Tab **309c** includes settings involved in animation and timelines that may be associated with objects. One embodiment of Animation Tab **309c** is shown, without limitation, in FIG. 3D, and is described, in Rempell ("Rempell").

A Color Tab **309d** includes a color palette for selecting object colors.

Bindings Tab **309e** is where web component operations are defined and dynamic binding settings are assigned. Thus, for example, a UI object is selected from canvas **305**, and a web component may be selected and configured from the bindings tab. When the user's work is saved, binding information is associated with the UI object that will appear on screen **137**.

FIG. 3E illustrates one embodiment of Bindings Tab and includes, without limitation, the following portions: Web Component and Web Services Operations **309e1**, Attributes Exposed list **309e2**, panel **309e3** which includes dynamic binding of server-side data base values to attributes for the selected object, Default Attribute Value **309e4**, Database Name **309e5**, Table Name **309e6**, Field Name **309e7**, Channel Name **309e8**, Channel Feed **309e9**, Operation **309e10**, Select Link **309e11**, and Link Set checkbox **309e12**.

Web Component and Web Services Operations **309e1** includes web components that may be added, edited or removed from a selected object. Since multiple web components can be added to the same object, any combination of mash-ups of 3rd party web services is possible. When the "Add" button of Web Component and Web Services Operations **309e1** is selected, a pop-up menu **319**, as shown in FIG. 3F, appears on publisher interface **300**. Pop-up menu **319** includes, but is not limited to, the options of: Select a Web Component **319a**; Select Results Page **319b**; Activation Options **319c**; Generate UI Objects **319d**; and Share Web Component **319e**.

The Select a Web Component **319a** portion presents a list of web components. As discussed herein, the web components are registered and are obtained from web component registry **220**.

Select Results Page **319b** is used to have the input and output on different pages—that is, when the Results page is different from Input page. The default selected results page is either the current page, or, if there are both inputs and outputs, it will be set provisionally to the next page in the current page order, if one exists.

Activation Options **319c** include, if there are no Input UI Objects, a choice to either "Preload" the web component, similar to how dynamic binding, or have the web component executed when the "Results" page is viewed by the consumer.

Generate UI Objects **319d**, if selected, will automatically generate the UI objects. If not selected, then the author will bind the Web Component Inputs and Results to previously created UI Objects.

23

Share Web Component **319e** is available and will become selected under the following conditions: 1) Web Component is Selected which already has been used by the current Application; or 2) the current Input page is also a "Result" page for that Web component. This permits the user of device **130**, after viewing the results, to extend the Web Component allowing the user to make additional queries against the same Web Component. Examples of this include, but are not limited to, interactive panning and zooming for a Mapping Application, or additional and/or refined searches for a Search Application.

Dynamic Binding permits the binding of real time data, that could either reside in a 3rd party server-side data base, or in the database maintained by Feed Collector **1010** for aggregating live RSS feeds, as described subsequently with reference to FIG. **10**.

Referring again to FIG. **3E**, Attributes Exposed list **309e2** are the attributes available for the selected object that can be defined in real time through dynamic binding.

Panel **309e3** exposes all the fields and tables associated with registered server-side data bases. In one embodiment, the user would select an attribute from the "Attributes Exposed List" and then select a data base, table and field to define the real time binding process. The final step is to define the record. If the Feed Collector data base is selected, for example, then the RSS "Channel Name" and the "Channel Feed" drop down menus will be available for symbolically selected the record. For other data bases the RSS "Channel Name" and the "Channel Feed" drop down menus are replaced by a "Record ID" text field.

Default Attribute Value **309e4** indicates the currently defined value for the selected attribute. It will be overridden in real time based on the dynamic linkage setting.

Database Name **309e5** indicates which server side data base is currently selected.

Table Name **309e6** indicates which table of the server side data base is currently selected.

Field Name **309e7**, indicates which field form the selected table of the server side data base is currently selected.

Channel Name **309e8** indicates a list of all the RSS feeds currently supported by the Feed Collector. This may be replaced by "Record ID" if a data base other than the Feed Collector **1010** is selected.

Channel Feed **309e9** indicates the particular RSS feed for the selected RSS Channel. Feed Collector **1010** may maintain multiple feeds for each RSS channel.

Operation **309e10**, as a default operation, replaces the default attribute value with the real time value. In other embodiments this operation could be append, add, subtract, multiply or divide.

Select Link **309e11** a button that, when pressed, creates the dynamic binding. Touching the "Select Link" will cause the current data base selections to begin the blink in some manner, and the "Select Link" will change to "Create Link". The user could still change the data base and attribute choices. Touching the "Create Link" will set the "Link Set" checkbox and the "Create Link" will be replaced by "Delete Link" if the user wishes to subsequently remove the link. When the application is saved, the current active links are used to create the SPDL.

Link Set checkbox **309e12** indicates that a link is currently active.

An example of the design of a display is shown in FIGS. **4A** and **4B** according to the system **100**, where FIG. **4A** shows publisher interface **300** having a layout **410** on canvas **305**, and FIG. **4B** shows a device **130** having the resulting layout **420** on screen **137**. Thus, for example, authoring platform

24

110 is used to design layout **410**. Authoring platform **110** then generates an Application and a Player specific to device **130** of FIG. **4B**. The Application and Player are thus used by device **130** to produce layout **420** on screen **137**.

As illustrated in FIG. **4A**, a user has placed the following on canvas **305** to generate layout **410**: text and background designs **411**, a first text input box **413**, a second text input box **415**, and a button **417**. As an example which is not meant to limit the scope of the present invention, layout **410** is screen prompts a user to enter a user name in box **413** and a password in box **415**, and enter the information by clicking on button **417**.

In one embodiment, all UI objects are initially rendered as Java objects on canvas **305**. When the Application is saved, the UI objects are transformed into the PDL, as described subsequently.

Thus, for example, layout **410** may be produced by the user of authoring platform **110** selecting and placing a first Text Field as box **413** then using the Resource Inspector **309** portion of interface **300** to define its attributes.

Device User Experience

Systems **100** and **200** provide the ability for a very large number of different types of user experiences. Some of these are a direct result of the ability of authoring platform **110** to bind UI objects to components of web services. The following description is illustrative of some of the many types of experiences of using a device **130** as part of system **100** or **200**.

Device **130** may have a one or more of a very powerful and broad set of extensible navigation objects, as well as object- and pointer-navigation options to make it easy to provide a small mobile device screen **137** with content and to navigate easily among page views, between Applications, or within objects in a single page view of an Application.

Navigation objects include various types of launch strips, various intelligent and user-friendly text fields and scrolling text boxes, powerful graphical complex lists, as well as Desktop-level business forms. In fact, every type of object can be used for navigation by assigning a navigation event to it. The authoring tool offers a list of navigation object templates, which then can be modified in numerous ways. Launch Strips and Graphical List Templates Launch Strips

Launch strips may be designed by the user of authoring platform **110** with almost no restrictions. They can be stationary or appear on command from any edge of the device, their size, style, audio feedback, and animations can be freely defined to create highly compelling experiences.

FIG. **5** shows a display **500** of launch strips which may be on displayed canvas **305** or on screen **137** of device **130** having the proper Player and Application. Display **501** includes a portal-type Launch Strip **501** and a channel-type Launch Strip **502**, either one of which may be included for navigating the Application.

Launch Strip **501** includes UI objects **501a**, **501b**, **501c**, **501d**, and **501e** that that becomes visible from the left edge of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector **309** with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. When the Applications Page, having been saved by authoring platform **110** and transferred to display **130**, is executed on device **130**, a user of the device may easily navigate the Application.

Launch Strip **502** includes UI objects **502b**, **502c**, **502d**, and **503e** that that becomes visible from the bottom of the display, when requested. UI objects **501a**, **501b**, **501c**, **501d**, and **501e** are each associated, through resource inspector

25

309 with navigational instructions, including but not limited to navigating to a different Applications Page, or requesting web content. Launch Strip 502 also includes UI objects 502a and 503g, which include the graphic of arrows, and which provide access to additional navigation objects (not shown) when selected by a user of device 130. Launch strip 502 may also include sound effects for each channel when being selected, as well as popup bubble help.

Additional navigational features are illustrated in FIG. 6A as a display of a Channel Selection List 601a, in FIG. 6B as a display of a Widget Selection List 601b, and in FIG. 6C as a display of a Phone List 601c. Lists 601a, 601b, and 601c may be displayed on canvas 305 or on screen 137 of device 130 having the proper Player and Application. As illustrated, graphical lists 601a, 601b, and 601c may contain items with many possible text and image elements. Each element can be defined at authoring time and/or populated dynamically through one or more Web Service 250 or API. Assignable Navigation Events. All objects, and/or all elements within an object, can be assigned navigation events that can be extended to registered web services or APIs. For example, a Rolodex-type of navigation event can dynamically set the starting slide of the targeted page view (or the starting view of a targeted Application).

In the embodiment of FIGS. 6A, 6B, and 6C, each list 601a, 601b, and 601c has several individual entries that are each linked to specific actions. Thus Channel Selection List 601a shows three objects, each dynamically linked to a web service (ESPN, SF 49ers, and Netflix) each providing a link to purchase or obtain items from the Internet. Widget Selection List 601b includes several objects presenting different widgets for selecting. Phone List 601c includes a list phone number objects of names that, when selected by a user of device 130 cause the number to be dialed Entries in Phone List 601c may be generated automatically from either the user's contact list that is resident on the device, or through a dynamic link to any of user's chosen server-side facilities such as Microsoft Outlook, Google Mail, etc. In one embodiment, Phone List 601c may be generated automatically using a web component assigned to the Application, which would automatically perform those functions.

In another embodiment, authoring platform 110 allows a navigation selection of objects with a Joy Stick and/or Cursor Keys in all 4 directions. When within a complex object the navigation system automatically adopts to the navigation needs for that object. For coordinate sensitive objects such as geographical information services (GIS) and location-based services (LBS) or virtual tours a soft cursor appears. For Lists, scrolling text areas and chats, Launch strips, and slide shows the navigation process permits intuitive selection of elements within the object. Scroll bars and elevators are optionally available for feedback. If the device has a pointing mechanism then scroll bars are active and simulate the desktop experience.

Personalization and Temporal Adoption

System 100 and 200 permit for the personalization of device 130 by a variety of means. Specifically, what is displayed on screen 137 may depend on either adoption or customization. Adoption refers to the selection of choices, navigation options, etc. are based on user usage patterns. Temporal adoption permits the skins, choices, layouts, content, widgets, etc. to be further influenced by location (for example home, work or traveling) and time of day (including season and day of week). Customization refers to user selectable skins, choices, layouts, dynamic content, widgets, etc. that are available either through a customization on the

26

phone or one that is on the desktop but dynamically linked to the user's other internet connected devices.

To support many personalization functions there must be a convenient method for maintaining, both within a user's session, and between sessions, memory about various user choices and events. Both utilizing a persistent storage mechanism on the device, or a database for user profiles on a server, may be employed.

FIG. 7 shows a display 700 of a mash-up which may be on displayed canvas 305 or on screen 137 of device 130 having the proper Player and Application. Display 700 includes several object 701 that have been dynamically bound, including an indication of time 701a, an indication of unread text messages 701b, an RSS news feed 701c (including 2 "ESPN Top Stories" 701c1 and 701c2), components 701d from two Web Services—a weather report ("The Weather Channel"), and a traffic report 701e ("TRAFFIC.COM").

In assembling the information of display 700, device 130 is aware of the time and location of the device—in this example the display is for a workday when a user wakes. Device 130 has been customized so that on a work day morning the user wishes to receive the displayed information. Thus in the morning, any messages received overnight would be flagged, the user's favorite RSS sports feeds would be visible, today's weather forecast would be available, and the current traffic conditions between the user's home and office would be graphically depicted. User personalization settings may be maintained as persistent storage on device 130 when appropriate, or in a user profile which is maintained and updated in real-time in a server-side data base. Push Capable Systems

In another embodiment system 100 or 200 is a push-capable system. As an example, of such systems, short codes may be applied to cereal boxes and beverage containers, and SMS text fields can be applied to promotional websites. In either case, a user of device 130 can text the short code or text field to an SMS server, which then serves the appropriate Application link back to device 130.

FIG. 8 is a schematic of an embodiment of a push enabled system 800. System 800 is generally similar to system 100 or 200. Device 130 is shown as part of a schematic of a push capable system 800 in FIG. 8. System 800 includes a website system 801 hosting a website 801, a server 803 and a content server 805. System 801 is connected to servers 803 and/or 805 through the Internet. Server 803 is generally similar to server 120, servers 805 is generally similar to server 140.

In one embodiment, a user sets up a weekly SMS update from website system 801. System 801 provides user information to server 803, which is an SMS server, when an update is ready for delivery. Server 803 provides device 130 with an SMS indication that the subscribed information is available and queries the user to see if they wish to receive the update. Website 801 also provides content server 805 with the content of the update. When a user of device 130 responds to the SMS query, the response is provided to content server 805, which provides device 130 with updates including the subscribed content.

In an alternative embodiment of system 800, server 803 broadcasts alerts to one or more devices 130, such as a logical group of devices. The user is notified in real-time of the pending alert, and can view and interact with the message without interrupting the current Application.

FIG. 9 is a schematic of an alternative embodiment of a push enabled system 900. System 900 is generally similar to system 100, 200, or 800. In system 900 a user requests information using an SMS code, which is delivered to device

27

130. System 900 includes a promotional code 901, a third-party server 903, and content server 805. Server 803 is connected to servers 803 and/or 805 through the Internet, and is generally similar to server 120.

A promotional code 901 is provided to a user of device 130, for example and without limitation, on print media, such as on a cereal box. The use of device 130 sends the code server 903. Server 903 then notifies server 805 to provide certain information to device 130. Server 805 then provides device 130 with the requested information.

Device Routines

Device routines 114 may include, but are not limited to: an authoring tool SDK for custom code development including full set of Java APIs to make it easy to add extensions and functionality to mobile Applications and tie Applications to back-end databases through the content server 140; an expanding set of web services 250 available through the authoring tool SDK; a web services interface to SOAP/XML enabled web services; and an RSS/Atom and RDF feed collector 1010 and content gateway 1130.

Authoring Tool SDK for Custom Code Development Including Full Set of Java APIs

In one embodiment, authoring platform 110 SDK is compatible for working with various integrated development environments (IDE) and popular plug ins such as J2ME Polish. In one embodiment the SDK would be another plug in to these IDEs. A large and powerful set of APIs and interfaces are thus available through the SDK to permit the seamless extension of any Application to back end business logic, web services, etc. These interfaces and APIs may also support listeners and player-side object operations.

There is a large set of listeners that expose both player-side events and dynamically linked server side data base events. Some examples of player side events are: player-side time based event, a page entry event, player-side user interactions and player-side object status. Examples of server-side data base events are when a particular set of linked data base field values change, or some filed value exceeds a certain limit, etc.

A superset of all authoring tool functionality is available through APIs for layer-side object operations. These include, but are not limited to: page view level APIs for inserting, replacing, and or modifying any page object; Object Level APIs for modifying any attribute of existing objects, adding definitions to attributes, and adding, hiding or replacing any object.

Authoring Tool SDK Available Web Services

The APIs permit, without limit, respond, with or without relying on back-end business logic, that is, logic that what an enterprise has developed for their business, to any player-side event or server-side dynamically linked data-base, incorporating any open 3rd party web service(s) into the response.

RSS/ATOM and RDF Feed Conversion Web Service

FIG. 10 is a schematic of one embodiment a system 1000 having a feed collector 1010. System 1000 is generally similar to system 100, 200, 800, or 900. Feed collector 1010 is a server side component of system 100 that collects RSS, ATOM and RDF format feeds from various sources 1001 and aggregates them into a database 1022 for use by the Applications built using authoring platform 110.

Feed collector 1010 is a standard XML DOM data extraction process, and includes Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, and Custom Populator Rule 1016, DOM XML Parsers 1011, 1015, and 1017, Feed Processed Data Writer 1018, Custom Rule

28

Based Field Extraction 1019, Rule-based Field Extraction 1020, Channel Data Controller 1021, and Database 1022.

The feed collector is primarily driven by two sets of parameters: one is the database schema (written as SQL DDL) which defines the tables in the database, as well as parameters for each of the feeds to be examined. The other is the feed collection rules, written in XML, which can be used to customize the information that is extracted from the feeds. Each of the feeds is collected at intervals specified by the feed parameter set in the SQL DDL.

Feed collector 1010 accepts information from ATOM, RDF or RSS feed sources 1001. Using a rules-based populator, any of these feeds can be logically parsed, with any type of data extraction methodology, either by using supplied rules, or by the author defining their own custom extraction rule. The rules are used by the parser to parse from the feed sources, and the custom rule base field extraction replaces the default rules and assembles the parsed information into the database

In particular, Atom Populator Rule 1012, RSS Populator Rule 1013, RDF Populator Rule 1014, Custom Populator Rule 1016, and DOM XML Parsers 1011, 1015, and 1017 are parse information from the feeds 1001, and Feed Processed Data Writer 1018, Custom Rule Based Field Extraction 1019, Rule-based Field Extraction 1020, and Channel Data Controller 1021, supply the content of the feeds in Database 1022, which is accessible through content server 140.

FIG. 11 is a schematic of an embodiment of a system 1100 having a Mobile Content Gateway 1130. System 1100 is generally similar to system 100, 200, 800, 900, or 1000. System 1100 includes an SDK 1131, feed collector 1010, database listener 1133, transaction server 1134, custom code 1135 generated from the SDK, Java APIs, Web Services 1137, and PDL snippets compacted objects 1139. System 1100 accepts input from Back End Java Code Developer 1120 and SOAP XML from Web Services 1110, and provides dynamic content to server 140 and Players to devices 130.

In one embodiment authoring platform 110 produces a Server-side PDL (SPDL) at authoring time. The SPDL resides in server 120 and provides a logical link between the Application's UI attributes and dynamic content in database 1022. When a user of device 130 requests dynamic information, server 120 uses the SPDL to determine the link required to access the requested content.

In another embodiment Web Services 1137 interface directly with 3rd party Web Services 1110, using SOAP, REST, JAVA, JavaScript, or any other interface for dynamically updating the attributes of the Application's UI objects. XSP Web Pages as a Web Service

In one embodiment, a PDL for a page is embedded within an HTML shell, forming one XSP page. The process of forming XSP includes compressing the description of the page and then embedding the page within an HTML shell.

In another embodiment, a PDL, which contains many individual page definitions, is split into separate library objects on the server, so that each page can to presented as a PDL as part of a Web Service.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java), and The code and data have been reduced by 4 to 10 times.

Compression has two distinct phases. The first takes advantage of how the primitive representations had been assembled, while the second utilizes standard LZ encoding.

The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

One embodiment for compacting data that may be used is described in Rempell. In that patent the compressed data is described as being a database. The terminology used here is a PDL, that is the "internal database" of Rempell is equivalent to the PDL of the present Application.

In Rempell, a process for compacting a "database" (that is, generating a compact PDL) is described, wherein data objects, including but not limited to, multi media objects such as colors, fonts, images, sound clips, URLs, threads, and video, including multi level animation, transformation, and time line are compacted. As an extension to Rempell in all cases these objects are reduced and transformed to Boolean, integer and string arrays.

The compression technique involves storing data in the smallest arrays necessary to compactly store web page information. The technique also includes an advanced form of delta compression that reduces integers so that they can be stored in a single byte, as high water marks.

Thus, for example, the high water mark for different types of data comprising specific web site settings are stored in a header record as Boolean and integer variables and URL and color objects. Data that defines web page, paragraph, text button, and image style and text button, image and paragraph high watermark settings can be stored in one-dimensional arrays as Boolean, integer and string variables and URL, font, image or thread objects at. The URL, color, font, image and thread objects can also be created as required.

Data that defines text button, image, paragraph, or other parent objects and paragraph line high watermark settings can be stored in two-dimensional arrays (by web page and by object number) as Boolean, integer, string, floating point variables and URLs. Again, the URL, color, font, image, audio clip, video clip, text area and thread objects can also be created as required. Data that defines a paragraph line and paragraph line segment high watermarks can be stored in three-dimensional arrays (by web page, by paragraph number, and by line number) as Boolean, integer or string variables. Again, the URL, color or font objects can be created as required. Data that defines a paragraph line segment can be stored into four-dimensional arrays (by web page, by paragraph number, by line number and by line number segment) as Boolean, integer or string variables or URL, color and font objects.

As a data field is added, changed or deleted, a determination is made at on whether a value for a given high watermark needs to be changed. If so, it is updated. As a specific method in the build engine is called, a determination is made on whether a feature flag needs to be set. For example, if a particular JAVA method is called, which requires an instance of a certain JAVA Class to be executed by the run time engine, then that JAVA Class is flagged, as well as any supporting methods, variables and/or object definitions.

In one implementation, the header record, the style record, the web page record, and the object records, are carefully defined in a specific order, written in that order, and explicitly cast by object type when read by the run time engine. Exception handling can be implemented to recover from any errors. This helps assure that data integrity is maintained throughout the build and run time processes.

Also described in Rempell is the "run generation process." This is equivalent generating a Player in the present application. This process starts when the build process detects that the user is finished defining the web site (user has saved the web site and invokes the run generation

process), and concludes with the actual uploading of all the necessary web site run time files to the user's server.

In one embodiment, the PDL includes a first record, a "Header" record, which contains can include the following information:

1: A file format version number, used for upgrading database in future releases.

2: The default screen resolution, in virtual pixels, for both the screen width and height. This is usually set to the web designer's screen resolution, unless overwritten by the user.

3: Whether the Application is a web site.

4: Virtual web page size settings. A calculation is performed by the build engine method, in order to calculate what the maximum web page length is, after reformatting all paragraphs on all internal web pages, based on the default screen resolution.

5: Web page and styles high watermarks.

6: The Websitename.

As new web pages or new objects are created by the user, or as text is added to or deleted from a paragraph, or as new styles are created or deleted, appropriate high watermarks are set, in order to show the current number of each of these entities. Thus, the values for the number of active web pages and the number of text button, image, paragraph or other styles are written as high watermarks in the header. The high watermarks for the number of text button, image, paragraph or other objects that exist for each web page, the number of lines for each paragraph object, and the number of line segments for each paragraph line are written within the body of the PDL, and used as settings for each of the loops in the four-dimensional data structure. Because no structural limits are set on the number of web pages, objects per web page, styles, or paragraph size, these high watermarks greatly reduce the external database file size, and the time it takes for the run time engine to process the data stored in its database.

The settings for all paragraph, text button and image styles are then written as a style record based on their high watermark. This data includes Boolean and integer variables, and font and color objects, written as a one-dimensional array, based on the high watermark values for the number of styles that exist.

The body of the PDL is then written. All Boolean values are written inside a four-dimensional loop. The outside loop contains the Boolean values used to define web pages (i.e. a one-dimensional array definition) as well as the high watermarks for the number of text button, image, paragraph or other objects per web page, with the loop set at the high watermark which defines the number of existing web pages for this web site structure. The second level consists of three or more two dimensional loops with the loops set to the high watermarks defining the actual number of text button, image, and paragraph or other objects that appear on any given web page and contains the values used to define web page objects (i.e. a two-dimensional array definition; web page number by object number). Included within the loop for paragraph objects are the high watermarks for the number of lines for each paragraph object. The third loop is set by the high watermark defining the actual number of paragraph lines that for all paragraphs on any web page and contains the values used to define paragraph lines (i.e. a three-dimensional array definition; web page number by object number by paragraph line.) Included within the loop for paragraph lines are the high watermarks for the number of line segments for each paragraph line. The inner most loop is set by the high watermarks defining the number of line segments per paragraph line and contains the values

31

used to define paragraph line segments (i.e. a four-dimensional array definition; web page number by object number by paragraph line by paragraph line segment).

All integer values are written inside a four-dimensional loop. Their four loops are controlled by the same high watermark settings as used for the Boolean records, and they describe the same logical entities.

Multimedia objects are written inside a two-dimensional loop. They include URL, color, and font objects, and can include other types of objects. A URL object is the encoded form of a URL Address, used by a web browser or a JAVA method to access files and web addresses. All multimedia objects must be serialized before they can be written. This means that the objects are converted into a common external definition format that can be understood by the appropriate deserialization technique when they are read back in and cast into their original object structure. The outside loop contains web page related objects, and the inner loop contains image, text button, paragraph, etc. related URL, color, and font objects. The outer loop is defined by the web page high watermark and the inner loops by the high watermarks for the actual number of text button, image, paragraph or other objects on a web page.

String records are written inside a four-dimensional loop. The outer loop may be empty. The second loop can include the string values for text button objects, audio and video filenames, and audio and video channel names. The third loop contains values for paragraph line related data, and the innermost loop contains the values for paragraph line segment definitions. The string records are controlled by the same high watermarks as those used for Boolean and integer records. String records are stored utilizing an appropriate field delimiter technology. In one implementation, a UTF encoding technology that is supported by JAVA is utilized.

Single and double floating-point, and long integer records are written inside a two-dimensional loop. The outer loop may be empty. The inner loop contains mathematical values required for certain animations and image processing algorithms. The single and double floating-point, and long integer records are controlled by the same high watermarks as those used for Boolean and integer records.

In one embodiment, a versionizing program analyzes the feature flags, and only those variable definitions, defined in the "Main" object class, relating to the object classes and methods that will be executed at run time, are extracted. All references to object classes that will be called at run time are extracted, creating the source code for the run engine "Main" object class that is ready for compilation.

All external image, video and audio files are resolved. The external references can be copied to designated directories, either on the user's local disk or file server. The file Pathnames can be changed to reflect these new locations. During the installation of the build tools, the necessary class libraries are either installed on the local system or made available on the server where the build tools can be optionally located. The necessary environmental variables are set to permit normal access to the required class libraries.

The customized run engine and a library of the referenced run time classes are compiled and converted into byte code. Finally, the run time engine for the web site is created. The required set of class objects required at run time is flagged for inclusion into the CAB/JAR file.

Next, an HTML Shell File (HSF) is constructed. The first step of this process is to determine whether the dynamic web page and object resizing is desired by testing the Application setting. If the Application was a web page, and thus requiring dynamic web page and object resizing, virtual screen

32

resolution settings are placed in an appropriate HTML compliant string. If the Application is a banner or other customized Application, the absolute values for the run time object (applet size) height and width are placed in an appropriate HTML compliant string as absolute width and height values.

An analysis is made for the background definition for the first internal web page. If a background pattern is defined, an appropriate HTML compliant string for setting the HTML "background" to the same background image is generated. If the first web page definition is a color instead, then the RGB values from those colors are converted to hexadecimal and an appropriate HTML compliant String is generated setting the "bgcolor" to the required hexadecimal value. This process synchronizes the web page background with the background that will be drawn by the web browser when it first interprets the HSF.

Thereafter, a JAVA method generates HTML and JavaScript compliant strings, that when executed by a web browser, generate additional sets of HTML and JavaScript compliant strings that are again executed by the web browser. More specifically, if the Application required dynamic web page and object resizing then JavaScript and HTML compliant strings are generated so that, when interpreted by the web browser at the time the HTML Shell File is initialized, the screen resolution sensing JAVA applet (SRS) will be executed. JavaScript code is generated in order to enable JavaScript to SRS applet communication. In one implementation, the code is generated by performing the following functions:

- 1: Determine the current web browser type.
- 2: Load the SRS from either a JAR or CAB File, based on web browser type.
- 3: Enter a timing loop, interrogating when the SRS is loaded.
- 4: When the SRS returns an "available" status, interrogate the SRS, which will return the current screen and window's actual height and width.
- 5: Convert the virtual screen resolution settings into appropriate absolute screen width and height values.

Strings defining additional JavaScript code are generated that perform the following steps at the time the HSF is initialized by the web browser:

- 1: Generate HTML compliant strings that set the run time engine's applet size to the appropriate values.
- 2: Generate an HTML complaint string that contains a "param" definition for linking the run time engine to the PDL.
- 3: Generate an HTML complaint string, dependent upon the type of web browser, which causes the current web browser to load either the JAR or the CAB File(s).
- 4: Generate JavaScript Code compliant strings that create and dynamically write the applet size defining HTML strings utilizing the JavaScript "document.write" function. This dynamically created code causes the web browser to execute the run time engine, in the correctly sized window, from the correct JAR or CAB file, and linked to the external database.

The writing out the above-generated HTML and JavaScript compliant strings creates the HSF. The necessary security policy permissions are asserted, and a "Website-name".html file is created.

In one embodiment, the processes for creating the CAB and JAR Files is as follows. The image objects, if any, which were defined on the first internal web page are analyzed. If they are set to draw immediately upon the loading of the first web page, then they are flagged for compression and inclu-

33

sion in the CAB and JAR Files. The feature flags are analyzed to determine which JAVA classes have been compiled. These class files are flagged for compression and inclusion in the library CAB and JAR Files. Strings that are BAT compliant definitions are created that will, when executed in DOS, create compressed CAB and JAR Files. These CAB and JAR Files contain the compressed versions of all necessary JAVA class files, image files, the "Website-name".class, customized run time engine file, and the "Website-name".dta database file. In one implementation of the invention, two BAT files are created. The first, when executed, will create a CAB/JAR file with the "Website-name".dta database file and the customized "main" run time engine, excluding all the image and button object animation, transformation, and image processing code. The second BAT file, when executed, will create a CAB/JAR file with all the library of all the referenced image and button object animation, transformation, and image processing code.

The necessary security policy permissions for file creation are then asserted, and "Website-name".bat and "Website-namelib".bat files are written. The "Website-name".bat and "Website-name".bat files are then executed under DOS, creating compressed "Website-name".cab and "Website-namelib".cab files and compressed "Website-name" jar and "Website-namelib" jar files. The HTML Shell File and the JAR and CAB files are then, either as an automatic process, or manually, uploaded to the user's web site. This completes the production of an XSP page that may be accessed through a web browser.

Displaying Content on a Device Decompression Management

Authoring platform 110 uses compaction to transform the code and data in an intelligent way while preserving all of the original classes, methods and attributes. This requires both an intelligent server engine and client (handset) Player, both of which fully understand what the data means and how it will be used.

The compaction technology described above includes transformation algorithms that deconstruct the logic and data into their most primitive representations, and then reassembles them in a way that can be optimally digested by further compression processing. This reassembled set of primitive representations defines the PDL of authoring platform 110.

Prior to compression the code has already been transformed so that there are no dependencies on the original programming language (Java). The data is then compressed by first taking advantage of how the primitive representations had been assembled, and then by utilizing standard LZ encoding. The final result is an overall reduction of 40 to 100 times the original size as represented by Java serialized objects.

The Player, when preparing a page view for execution, decompresses and then regenerate the original objects, but this time in compliance with the programming APIs of device 130. Specifically, device 130 operates on compacted image pages, one at a time. The cache manager retrieves, decompresses, and reassembles the compacted page images into device objects, which are then interpreted by device 130 for display on screen 137.

Response Director

In one embodiment, system 100 includes a Response Director, which determines a user's handset, fetches the correct Application from different databases, and delivers a respective highly compressed Application in a PDL format over the air (OTA).

34

In one embodiment, the Response Director operates on a network connected computer to provide the correct Player to a given device based on the information the device sent to it. As an example, this may occur when a device user enters their phone number into some call-to-action web page. The response director is called and sends an SMS message to the device, which responds, beginning the recognition process.

FIG. 12 illustrates one embodiment of a system 1200 that includes a response director 210, a user agent database 1201, an IP address database 1203, and a file database 1205. System 1200 is generally similar to system 100, 200, 800, 900, 1000, or 1100.

Databases 1201, 1203, and 1205 may reside on server 120, 210, or any computer system in communication with response director 210. System 1200, any mobile device can be serviced, and the most appropriate Application for the device will be delivered to the device, based on the characteristics of the device.

User agent database 1201 includes user agent information regarding individual devices 130 that are used to identify the operating system on the device. IP address database 1203 identifies the carrier/operator of each device 130. File database 1205 includes data files that may operate on each device 130.

The following is an illustrative example of the operation of response director 210. First, a device 1300 generates an SMS message, which automatically sends an http:// stream that includes handset information and its phone number to response director 210. Response director 210 then looks at a field in the http header (which includes the user agent and IP address) that identifies the web browser (i.e., the "User Agent"). The User Agent prompts a database lookup in user agent database 1201 which returns data including, but not limited to, make, model, attributes, MIDP 1.0 MIDP 2.0, WAP and distinguishes the same models from different countries. A lookup of the IP address in IP address 1203 identifies the carrier/operator.

File database 1205 contains data types, which may include as jad1, jad2, html, wml/wap2, or other data types, appropriate for each device 130. A list of available Applications are returned to a decision tree, which then returns, to device 130, the Application that is appropriate for the respective device. For each file type, there is an attributes list (e.g., streaming video, embedded video, streaming audio, etc.) to provide enough information to determine what to send to the handset.

Response director 210 generates or updates an html or jad file populating this text file with the necessary device and network dependent parameters, including the Application dependent parameters, and then generate, for example, a CAB or JAD file which contains the necessary Player for that device. For example, the jad file could contain the operator or device type or extended device-specific functions that the player would then become aware of

If there is an Application that has a data type that device 130 cannot support, for example, video, response director 210 sends an alternative Application to the handset, for example one that has a slide show instead. If the device cannot support a slide show, an Application might have text and images and display a message that indicates it does not support video.

Another powerful feature of response director 210 is its exposed API from the decision tree that permits the overriding of the default output of the decision tree by solution providers. These solution providers are often licensees who want to further refine the fulfillment of Applications and Players to specific devices beyond what the default algo-

35

rithms provide. Solution providers may be given a choice of Applications and then can decide to use the defaults or force other Applications.

Authoring platform **110** automatically scales Applications at publishing time to various form factors to reduce the amount of fragmentation among devices, and the Response Director serves the appropriately scaled version to the device. For example, a QVGA Application will automatically scale to the QCIF form factor. This is important because one of the most visible forms of fragmentation resides in the various form factors of wireless, and particularly mobile, devices, which range from 128×128, 176×208, 240×260, 220×220, and many other customized sizes in between.

FIG. **13** is a schematic of an embodiment of a system **1300**. System **1300** is generally similar to system **1200**. System **1300** is an overview of the entire Player fulfillment process, starting with the generation of players during the player build process.

System **1300** includes response director **210**, a device characteristics operator and local database **1301**, a player profile database **1303** and a player build process **1305**, which may be authoring platform **110**.

As an example of system **1300**, when response director **210** receives an SMS message from device **130**, the response director identifies the device characteristics operator and locale from database **1301** and a Player URL from database **1303** and provides the appropriate Player to the device.

In another embodiment, Player P extend the power of response director **210** by adapting the Application to the resources and limitations of any particular device. Some of these areas of adaptation include the speed of the device's microprocessor, the presence of device resources such as cameras and touch screens. Another area of adaptation is directed to heap, record store and file system memory constraints. In one embodiment, the Player will automatically throttle down an animation to the frame rate that the device can handle so that the best possible user experience is preserved. Other extensions include device specific facilities such as location awareness, advanced touch screen interactions, push extensions, access to advanced phone facilities, and many others

Memory Management

In one embodiment, Player P includes a logical page virtual memory manager. This architecture requires no supporting hardware and works efficiently with constrained devices. All page view images, which could span multiple Applications, are placed in a table as highly compacted and compressed code. A typical page view will range from 500 bytes up to about 1,500 bytes. (See, for example, the Rempell patent) When rolled into the heap and instantiated this code increases to the more typical 50,000 up to 250,000 bytes. Additional alert pages may also be rolled into the heap and superimposed on the current page view. Any changes to any page currently downloaded are placed in a highly compact change vector for each page, and rolled out when the page is discarded. Note that whenever an Application is visited that had previously been placed in virtual memory the Server is interrogated to see if a more current version is available, and, if so, downloads it. This means that Application logic can be changed in real-time and the results immediately available to mobile devices.

To operate efficiently with the bandwidth constraints of mobile devices, authoring platform **110** may also utilize anticipatory streaming and multi-level caching. Anticipatory streaming includes multiple asynchronous threads and IO request queues. In this process, the current Application is

36

scanned to determine if there is content that is likely to be required in as-yet untouched page views. Anticipatory streaming also looks for mapping Applications, where the user may zoom or pan next so that map content is retrieved prior to the user requesting it. For mapping applications, anticipatory streaming downloads a map whose size is greater than the map portal size on the device and centered within the portal. Any pan operation will anticipatory stream a section of the map to extend the view in the direction of the pan while, as a lower priority, bring down the next and prior zoom levels for this new geography. Zooming will always anticipatory stream the next zoom level up and down.

Multi-level caching determines the handset's heap through an API, and also looks at the record store to see how much memory is resident. This content is placed in record store and/or the file system, and may, if there is available heap, also place the content there as well. Multi-level caching permits the management of memory such that mobile systems best use limited memory resources. Multi-level caching is a memory management system with results similar to embedding, without the overhead of instantiating the content. In other words, with multi-level caching, handset users get an "embedded" performance without the embedded download. Note that when content is flagged as cacheable and is placed in persistent storage, a digital rights management (DRM) solution will be used.

One embodiment of each of the methods described herein is in the form of a computer program that executes on a processing system. Thus, as will be appreciated by those skilled in the art, embodiments of the present invention may be embodied as a method, an apparatus such as a special purpose apparatus, an apparatus such as a data processing system, or a carrier medium, e.g., a computer program product. The carrier medium carries one or more computer readable code segments for controlling a processing system to implement a method. Accordingly, aspects of the present invention may take the form of a method, an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of carrier medium (e.g., a computer program product on a computer-readable storage medium) carrying computer-readable program code segments embodied in the medium. Any suitable computer readable medium may be used including a magnetic storage device such as a diskette or a hard disk, or an optical storage device such as a CD-ROM.

It will be understood that the steps of methods discussed are performed in one embodiment by an appropriate processor (or processors) of a processing (i.e., computer) system executing instructions (code segments) stored in storage. It will also be understood that the invention is not limited to any particular implementation or programming technique and that the invention may be implemented using any appropriate techniques for implementing the functionality described herein. The invention is not limited to any particular programming language or operating system. It should thus be appreciated that although the coding for programming devices has not been discussed in detail, the invention is not limited to a specific coding method. Furthermore, the invention is not limited to any one type of network architecture and method of encapsulation, and thus may be utilized in conjunction with one or a combination of other network architectures/protocols.

Reference throughout this specification to "one embodiment," "an embodiment," or "certain embodiments" means that a particular feature, structure or characteristic described

37

in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases “in one embodiment,” “in an embodiment,” or “in certain embodiments” in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner, as would be apparent to one of ordinary skill in the art from this disclosure, in one or more embodiments.

Throughout this specification, the term “comprising” shall be synonymous with “including,” “containing,” or “characterized by,” is inclusive or open-ended and does not exclude additional, unrecited elements or method steps. “Comprising” is a term of art which means that the named elements are essential, but other elements may be added and still form a construct within the scope of the statement. “Comprising” leaves open for the inclusion of unspecified ingredients even in major amounts.

Similarly, it should be appreciated that in the above description of exemplary embodiments, various features of the invention are sometimes grouped together in a single embodiment, figure, or description thereof for the purpose of streamlining the disclosure and aiding in the understanding of one or more of the various inventive aspects. This method of disclosure, however, is not to be interpreted as reflecting an intention that the claimed invention requires more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive aspects lie in less than all features of a single foregoing disclosed embodiment, and the invention may include any of the different combinations embodied herein. Thus, the following claims are hereby expressly incorporated into this Mode(s) for Carrying Out the Invention, with each claim standing on its own as a separate embodiment of this invention.

Thus, while there has been described what is believed to be the preferred embodiments of the invention, those skilled in the art will recognize that other and further modifications may be made thereto without departing from the spirit of the invention, and it is intended to claim all such changes and modifications as fall within the scope of the invention. For example, any formulas given above are merely representative of procedures that may be used. Functionality may be added or deleted from the block diagrams and operations may be interchanged among functional blocks. Steps may be added or deleted to methods described within the scope of the present invention.

We claim:

1. A system for generating code to provide content on a display of a device, said system comprising:
computer memory storing:

- a) symbolic names required for evoking one or more web components each related to a set of inputs and outputs of a web service obtainable over a network, where the symbolic names are character strings that do not contain either a persistent address or pointer to an output value accessible to the web service, where each symbolic name has an associated data format class type corresponding to a subclass of User Interface (UI) objects that support the data format type of the symbolic name, and where each symbolic name has a preferred UI object, and

- b) an address of the web service;

an authoring tool configured to:

define a UI object for presentation on the display,

where said defined UI object corresponds to a web component included in said computer memory selected from a group consisting of an input of the

38

web service and an output of the web service, where each defined UI object is either:

- 1) selected by a user of the authoring tool; or
- 2) automatically selected by the system as the preferred UI object corresponding to the symbolic name of the web component selected by the user of the authoring tool,

access said computer memory to select the symbolic name corresponding to the web component of the defined UI object,

associate the selected symbolic name with the defined UI object, where the selected symbolic name is only available to UI objects that support the defined data format associated with that symbolic name,

store information representative of said defined UI object and related settings in a database;

retrieve said information representative of said one or more said UI object settings stored in said database; and

build an application consisting of one or more web page views from at least a portion of said database utilizing at least one player, where said player utilizes information stored in said database to generate for the display of at least a portion of said one or more web pages,

wherein when the application and player are provided to the device and executed on the device, and

when the user of the device provides one or more input values associated with an input symbolic name to an input of the defined UI object, the device provides the user provided one or more input values and corresponding input symbolic name to the web service, the web service utilizes the input symbolic name and the user provided one or more input values for generating one or more output values having an associated output symbolic name,

and the player receives the output symbolic name and corresponding one or more output values and provides instructions for the display of the device to present an output value in the defined UI object.

2. The system of claim 1, where said system stores information in a registry, and wherein the registry includes definitions of input and output related to said web service.

3. The system of claim 1, where said web component is a text chat, a video chat, an image, a slideshow, a video, or an RSS feed.

4. The system of claim 1, where said UI object is an input field for a chat.

5. The system of claim 1, where said UI object is an input field for a web service.

6. The system of claim 1, where said UI object is an input field usable to obtain said web component, where said input field includes a text field, a scrolling text box, a check box, a drop down-menu, a list menu, or a submit button.

7. The system of claim 1, where said web component is an output of a web service, is the text provided by one or more simultaneous chat sessions, is the video of a video chat session, is a video, an image, a slideshow, an RSS display, or an advertisement.

8. The system of claim 1, where said authoring tool is further configured to:

define a phone field or list; and

generate code that, when executed on the device, allows a user to supply a phone number to said phone field or list.

9. The system of claim 1, where said authoring tool is further configured to:

39

define a SMS field or list; and
generate code that, when executed on the device, allows
a user to supply an SMS address to said SMS field or
list.

10. The system of claim 1,
where said code includes three or more codes, where one
of said three or more codes is device specific, and
where two of said three or more codes is device
independent.

11. The system of claim 1, where said code is provided
over said network.

12. The system of claim 1, wherein said defined UI object
corresponds to a widget.

13. The system of claim 1, where said player is activated
and runs in a web browser.

14. The system of claim 1, where said player is a native
program.

15. A method of displaying content on a display of a
device having a player and non-volatile computer memory
storing symbolic names required for evoking one or more
web components each related to a set of inputs and outputs
of a web service obtainable over a network, where the
symbolic names are character strings that do not contain
either a persistent address or pointer to an output value
accessible to the web service, where each symbolic name
has an associated data format class type corresponding to a
subclass of User Interface (UI) objects that support the data
format type of the symbolic name, and where each symbolic
name has a preferred UI object, and an address of the web
service, said method comprising:

defining a UI object for presentation on the display, where
said UI object corresponds to a web component
included in the computer memory, where said web
component is selected from a group consisting of an
input of a web service and an output of the web service,
where each defined UI object is either: 1) selected by a
user of the authoring tool; or 2) automatically selected
by the system as the preferred UI object corresponding
to a symbolic name of the web component selected by
the user of the authoring tool;

selecting the symbolic name corresponding to the web
component of the defined UI object;

associating the selected symbolic name with the defined
UI object, where the selected symbolic name is only
available to UI objects that support the defined data
format associated with that symbolic name;

storing information representative of said defined UI
object and related settings in a database;

retrieving said information representative of said one or
more said UI object settings stored in said database;
and

building an application consisting of one or more web
page views from at least a portion of said database
utilizing the player, where said player utilizes informa-
tion stored in said database to generate for the display
of at least a portion of said one or more web pages,

40

wherein, when the application and player are provided to
the device and executed on the device, and when the
user of the device provides one or more input values
associated with an input symbolic name to an input of
the defined UI object, 1) the device provides the user
provided one or more input values and corresponding
input symbolic name to the web service, 2) the web
service utilizes the input symbolic name and the user
provided one or more input values for generating one or
more output values having an associated output sym-
bolic name, and 3) the player receives the output
symbolic name and corresponding one or more output
values and provides instructions for the display of the
device to present an output value in the defined UI
object.

16. The method of claim 15, where said method stores
information in a registry, and wherein the registry includes
definitions of input and/or output related to said web service.

17. The method of claim 15, where said web component
is a text chat, a video chat, an image, a slideshow, a video,
or an RSS feed.

18. The method of claim 15, where said UI object is an
input field for a chat.

19. The method of claim 15, where said UI object is an
input field for a web service.

20. The method of claim 15, where said UI object is an
input field usable to obtain said web component, where said
input field includes a text field, a scrolling text box, a check
box, a drop down-menu, a list menu, or a submit button.

21. The method of claim 15, where said web component
is an output of a web service, is the text provided by one or
more simultaneous chat sessions, is the video of a video chat
session, is a video, an image, a slideshow, an RSS display,
or an advertisement.

22. The method of claim 15, further comprising:

defining a phone field or list; and
generating code that, when executed on the device, allows
a user to supply a phone number to said phone field or
list.

23. The method of claim 15, further comprising:

defining a SMS field or list; and
generating code that, when executed on the device, allows
a user to supply an SMS address to said SMS field or
list.

24. The method of claim 15, and such that said player
interprets dynamically received, device independent values
of the web component defined in the application.

25. The method of claim 15, further comprising:

providing said application and player over said network.

26. The method of claim 15, wherein said UI object
corresponds to a widget.

27. The method of claim 15, where said player is activated
and runs in a web browser.

28. The method of claim 15, where said player is a native
program.

* * * * *

EXHIBIT F

**IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

Express Mobile, Inc,

Plaintiff,

v.

KTree Computer Solutions Inc.,

Defendant.

§
§
§
§
§
§
§
§
§
§
§

Case No. 2:17-cv-128-JRG-RSP

REPORT AND RECOMMENDATION

Before the Court is Defendant’s Motion for Judgment on the Pleadings (Dkt. No. 9) (the “Motion”). The Court recommends the Motion be **DENIED WITHOUT PREJUDICE**.

First, the claims appear to address a problem particular to the internet: dynamically generating websites and displaying web pages based on stored user-selected settings. *See DDR Holdings, LLC v. Hotels.com, L.P.*, 773 F.3d 1245, 1257 (Fed. Cir. 2014). On the face of the asserted patents, the claims appear to be necessarily rooted in computer technology and directed to specific problems of and improvements to then-existing web publishing applications. *See, e.g.*, Dkt. No. 22-1 at 1:6–8, 1:22–24 (“As such it is virtually impossible to write a web publishing application in HTML and JavaScript. All conventional implementations must, and do, utilize a full-featured programming language, such as C++ or Visual Basic.”); 1:36–39 (“As such, a conventional web publishing application written in one of these languages suffers from the severe performance problems inherent in these languages.”); 2:60–64 (“Because of the implementation of a variety of performance and file reduction techniques, the entire run time environment can range from as low as 12K, and no larger than

5OK.”); 3:4–6 (“The present invention provides a real time, dynamic linkage between JAVA and HTML including two-way communications, in real time, between JAVA and JavaScript.”).

Second, the asserted claims do not bear all of the hallmarks of claims that have been invalidated on the pleadings by other courts in the past. For example, the claims are not merely do-it-on-a-computer claims. *See Alice Corp. Pty. v. CLS Bank Int'l*, 134 S. Ct. 2347, 2358, 189 L. Ed. 2d 296 (2014). The Court has also reviewed *Internet Patents Corp. v. Active Network, Inc.*, 790 F.3d 1343 (Fed. Cir. 2015), cited by Defendant, and is not persuaded that the case is directly on point. For one, it is unclear from this record whether a run time file or run time engine as claimed in the asserted patents is analogous to the “template file” recited in the *Internet Patents* claims. The Court also does not have a firm conviction at this early stage that these claims contain no inventive concept.

Thus, the Court is not persuaded that claim construction and a fuller factual record would be unhelpful to the patent eligibility determination. *See Autumn Cloud LLC v. TripAdvisor, Inc.*, Dkt. No. 66 at 2–4, Case No. 2:16-cv-853-JRG-RSP (E.D. Tex. April 3, 2017).

Accordingly, the Court recommends the Motion be **DENIED WITHOUT PREJUDICE**.¹

SIGNED this 4th day of May, 2017.


 ROY S. PAYNE
 UNITED STATES MAGISTRATE JUDGE

¹ A party’s failure to file written objections to the findings, conclusions, and recommendations contained in this report within fourteen days after being served with a copy shall bar that party from de novo review by the district judge of those findings, conclusions, and recommendations and, except on grounds of plain error, from appellate review of unobjected-to-factual findings and legal conclusions accepted and adopted by the district court. Fed. R. Civ. P. 72(b)(2); *see Douglass v. United Servs. Auto. Ass’n*, 79 F.3d 1415, 1430 (5th Cir. 1996) (en banc).

EXHIBIT G

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA

EXPRESS MOBILE, INC.,

Plaintiff,

v.

CODE AND THEORY LLC,

Defendant.

Case No. [18-cv-04679-RS](#)

EXPRESS MOBILE, INC.,

Plaintiff,

v.

PANTHEON SYSTEMS INC.,
Defendant.

Case No. [18-cv-04688-RS](#)

**ORDER DENYING MOTIONS TO
DISMISS**

I. INTRODUCTION

These are two in a number of related patent cases involving allegations that the various defendants infringe U.S. Patent Nos. 6,546,397 (the '397 patent) and 7,594,168 (the '168 patent), which share a specification. Certain terms of the patents have been the subject of claim construction by this court in *X.Commerce, Inc. v. Express Mobile, Inc.*, Case No. 17-cv-02605-RS, a declaratory relief action where the patent holder, plaintiff Express Mobile in these cases, appears as the defendant. Claim construction has also taken place in at least one other district in other litigation brought by Express Mobile on the patents.

Defendants in these two actions move to dismiss contending the patent claims are drawn only to abstract ideas, ineligible for protection under Section 101 of the Patent Act, as elucidated in *Alice Corp. Pty. Ltd. v. CLS Bank Int'l*, 134 S. Ct. 2347 (2014) and its progeny. Because the patents purport to describe a novel technological approach to creating websites on the internet, defendants' characterization of the patents as claiming only an abstract idea fails, and the motions to dismiss must be denied.

II. BACKGROUND

As described in the patents' shared specification and explained in the briefing, Express Mobile contends the patents "bring together a number of disparate ideas and concepts, to create a new paradigm for creating, storing, and building web pages." According to Express Mobile, prior to the invention of the patents, web pages were created, stored and rendered in a fundamentally different manner. Individual web pages were typically created by either programming directly in HTML or JavaScript code, or by using a visual editor that output HTML formatted files. These approaches allegedly were cumbersome and inflexible, in various respects.

The inventive methodology purportedly described in the patents involves building a web page by defining it as a set of user-selected "objects" and/or "settings." The result is not a markup language code file for the web page, but instead a collection of user selected objects and object attributes. These can be saved in a database, for ease of access and efficient storage. Express Mobile explains that because complete code files for each page do not need to be stored, the page structure—the full HTML code itself—is created on the fly each time the page is loaded in a user's Web browser. This is achieved in part through a browser-appropriate "run time engine" and related files.

Defendants contend claim 1 of the '397 patent is representative of both patents for purposes of *Alice* analysis.¹ It provides:

¹ Express Mobile asserts claim 1 is not representative, and discusses at least one other claim (Claim 1 of the '168) patent, but it has not shown the result would be any different for any of the

1 A method to allow users to produce Internet websites on and for
2 computers having a browser and a virtual machine capable of
3 generating displays, said method comprising:

4 (a) presenting a viewable menu having a user selectable panel of
5 settings describing elements on a website, said panel of setting being
6 presented through a browser on a computer adapted to accept one or
7 more of said selectable settings in said panel as inputs therefrom,
8 and where at least one of said user selectable settings in said panel
9 corresponds to commands in said virtual machine;

10 (b) generating a display in accordance with one or more user
11 selected settings substantially contemporaneously with the selection
12 thereof;

13 (c) storing information representative of said one or more user
14 selected settings in a database;

15 (d) generating a website at least in part by retrieving said
16 information representative of said one or more user selected settings
17 stored in said database; and

18 (e) building one or more webpages to generate said website from at
19 least a portion of said database and at least one run time file, where
20 said at least one run time file utilizes information stored in said
21 database to generate virtual machine commands for the display of at
22 least a portion of said one or more web pages.

23 III. DISCUSSION

24 As explained in *Alice*, the Supreme Court has “interpreted § 101 and its predecessors ... for
25 more than 150 years” to “‘contain[] an important implicit exception: Laws of nature, natural
26 phenomena, and abstract ideas are not patentable.’ ” The *Alice* court applied a two-step framework
27 for determining patent eligibility, previously articulated in *Mayo Collaborative Servs. v.*
28 *Prometheus Labs., Inc.*, 132 S.Ct. 1289 (2012):

First, we determine whether the claims at issue are directed to one of
those patent-ineligible concepts. If so, we then ask, “[w]hat else is

other asserted claims.

there in the claims before us?” To answer that question, we consider the elements of each claim both individually and “as an ordered combination” to determine whether the additional elements “transform the nature of the claim” into a patent-eligible application. We have described step two of this analysis as a search for an “inventive concept”—i.e., an element or combination of elements that is sufficient to ensure that the patent in practice amounts to significantly more than a patent upon the [ineligible concept] itself.

Alice, 134 S.Ct. at 2355.

Alice also explained, “The ‘abstract ideas’ category embodies “the longstanding rule that ‘[a]n idea of itself is not patentable.’ ” *Id.* at 2355; *see also Le Roy v. Tatham*, 14 How. 156, 175, 14 L.Ed. 367 (1853). (“A principle, in the abstract, is a fundamental truth; an original cause; a motive; these cannot be patented, as no one can claim in either of them an exclusive right”).

Alice repeated the caution given in *Mayo*, however, that the exclusion for “abstract ideas” must not be applied too broadly, “we tread carefully in construing this exclusionary principle lest it swallow all of patent law.” 134 S.Ct. at 2354 (*citing Mayo*, 132 S.Ct. at 1293–1294.) At some level, “all inventions . . . embody, use, reflect, rest upon, or apply laws of nature, natural phenomena, or abstract ideas.” *Mayo*, 132 S.Ct. at 1293.

On the facts before it, the *Alice* court also expressly declined to “labor to delimit the precise contours of the ‘abstract ideas’ category.” 134 S.Ct. at 2357. Instead, it merely found that the concept of providing an “intermediated settlement” was not meaningfully distinguishable from the idea of “risk hedging” at issue in *Bilski v. Kappos*, 561 U.S. 593 (2010). In both instances, the idea involved was “a fundamental economic practice long prevalent in our system of commerce.” *Id.* at 2356.

Here, defendants rely primarily on *Intellectual Ventures I LLC v. Capital One Bank (USA)*, 792 F.3d 1363 (Fed. Cir. 2015). They argue the patents here are not meaningfully distinguishable from one held invalid in that case, U.S. Patent No. 7,603,382, entitled “Advanced Internet Interface Providing User Display Access of Customized Webpages.” Defendants’ insistence to the contrary notwithstanding, the patents are simply not directly comparable. The ’382 patent in *Intellectual Ventures* “generally relate[d] to customizing web page content as a function of

navigation history and information known about the user.” 792 F.3d at 1369. The representative claim described “[a] system for providing web pages accessed from a web site in a manner which presents the web pages tailored to an individual user.” The *Intellectual Ventures* court had little trouble concluding that merely tailoring the information presented to a website user based on information about that user or when the website was being viewed represented patent-ineligible abstract ideas.

This sort of information tailoring is “a fundamental . . . practice long prevalent in our system. . . .” . . . There is no dispute that newspaper inserts had often been tailored based on information known about the customer—for example, a newspaper might advertise based on the customer’s location. Providing this minimal tailoring—e.g., providing different newspaper inserts based upon the location of the individual—is an abstract idea

Tailoring information based on the time of day of viewing is also an abstract, overly broad concept long-practiced in our society. There can be no doubt that television commercials for decades tailored advertisements based on the time of day during which the advertisement was viewed.

Id.

The patents here are not comparable merely because they also involve webpages that reflect information provided by a “user.” Indeed, the patents do not even involve the same category of “user”—here the “user” is the person who is trying to *create* webpages, in *Intellectual Ventures* the user is a person viewing the webpage to whom customized content will be delivered. The patents here are directed at a purportedly revolutionary technological solution to a technological problem—how to create webpages for the internet in a manner that permits “what you see is what you get” editing, and a number of other alleged improvements over the then-existing methodologies.

A more apt comparison is *Enfish, LLC v. Microsoft Corp.*, 822 F.3d 1327 (Fed. Cir. 2016), which reversed a district court’s finding of ineligibility under *Alice*. At issue in *Enfish* was “an innovative logical model for a computer database.” *Enfish* supports the notion that a dividing line can be drawn between patents which merely describe using a computer and/or the internet to carry

out pre-existing and well-known tasks and techniques, and those that relate to the functioning of computers themselves. The former will virtually always fail under *Alice* unless some “inventive concept” can be found in the second step of the analysis; the latter are substantially less easily characterized as merely abstract ideas.

Enfish explains:

The first step in the *Alice* inquiry in this case asks whether the focus of the claims is on the specific asserted improvement in computer capabilities (*i.e.*, the self-referential table for a computer database) or, instead, on a process that qualifies as an “abstract idea” for which computers are invoked merely as a tool. As noted *infra*, in *Bilski* and *Alice* and virtually all of the computer-related § 101 cases we have issued in light of those Supreme Court decisions, it was clear that the claims were of the latter type—requiring that the analysis proceed to the second step of the *Alice* inquiry, which asks if nevertheless there is some inventive concept in the application of the abstract idea. See *Alice*, 134 S.Ct. at 2355, 2357–59. In this case, however, the plain focus of the claims is on an improvement to computer functionality itself, not on economic or other tasks for which a computer is used in its ordinary capacity.

822 F.3d at 1335–36.

Enfish drew a line between “improvement[s] to computer functionality itself,” and “economic or other tasks for which a computer is used in its ordinary capacity.” The court concluded:

we find that the claims at issue in this appeal are not directed to an abstract idea within the meaning of *Alice*. Rather, they are directed to a specific improvement to the way computers operate”

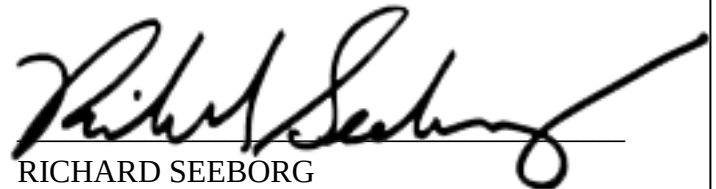
Id. at 1336.

So too here.

Finally, to the extent that defendants are arguing that any potentially patent-eligible technological improvements set out in the specification are not reflected in the actual claims, dismissal under *Alice* is not appropriate, at least at this juncture. Although some claim construction has taken place, it simply cannot be said on the present record that the claims are drawn so broadly as to be divorced from the potentially patent-eligible purported technological improvements described in the specification. Accordingly, the motions to dismiss are denied.

IT IS SO ORDERED.

Dated: January 29, 2019



RICHARD SEEBORG
United States District Judge

United States District Court
Northern District of California

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA

EXPRESS MOBILE, INC.,
Plaintiff,

v.

CODE AND THEORY LLC,
Defendant.

Case No. [18-cv-04679-RS](#)

EXPRESS MOBILE, INC.,
Plaintiff,

v.

PANTHEON SYSTEMS INC.,
Defendant.

Case No. [18-cv-04688-RS](#)

**ORDER DENYING MOTIONS TO
DISMISS**

I. INTRODUCTION

These are two in a number of related patent cases involving allegations that the various defendants infringe U.S. Patent Nos. 6,546,397 (the '397 patent) and 7,594,168 (the '168 patent), which share a specification. Certain terms of the patents have been the subject of claim construction by this court in *X.Commerce, Inc. v. Express Mobile, Inc.*, Case No. 17-cv-02605-RS, a declaratory relief action where the patent holder, plaintiff Express Mobile in these cases, appears as the defendant. Claim construction has also taken place in at least one other district in other litigation brought by Express Mobile on the patents.

Defendants in these two actions move to dismiss contending the patent claims are drawn only to abstract ideas, ineligible for protection under Section 101 of the Patent Act, as elucidated in *Alice Corp. Pty. Ltd. v. CLS Bank Int'l*, 134 S. Ct. 2347 (2014) and its progeny. Because the patents purport to describe a novel technological approach to creating websites on the internet, defendants' characterization of the patents as claiming only an abstract idea fails, and the motions to dismiss must be denied.

II. BACKGROUND

As described in the patents' shared specification and explained in the briefing, Express Mobile contends the patents "bring together a number of disparate ideas and concepts, to create a new paradigm for creating, storing, and building web pages." According to Express Mobile, prior to the invention of the patents, web pages were created, stored and rendered in a fundamentally different manner. Individual web pages were typically created by either programming directly in HTML or JavaScript code, or by using a visual editor that output HTML formatted files. These approaches allegedly were cumbersome and inflexible, in various respects.

The inventive methodology purportedly described in the patents involves building a web page by defining it as a set of user-selected "objects" and/or "settings." The result is not a markup language code file for the web page, but instead a collection of user selected objects and object attributes. These can be saved in a database, for ease of access and efficient storage. Express Mobile explains that because complete code files for each page do not need to be stored, the page structure—the full HTML code itself—is created on the fly each time the page is loaded in a user's Web browser. This is achieved in part through a browser-appropriate "run time engine" and related files.

Defendants contend claim 1 of the '397 patent is representative of both patents for purposes of *Alice* analysis.¹ It provides:

¹ Express Mobile asserts claim 1 is not representative, and discusses at least one other claim (Claim 1 of the '168) patent, but it has not shown the result would be any different for any of the

1 A method to allow users to produce Internet websites on and for
2 computers having a browser and a virtual machine capable of
3 generating displays, said method comprising:

4 (a) presenting a viewable menu having a user selectable panel of
5 settings describing elements on a website, said panel of setting being
6 presented through a browser on a computer adapted to accept one or
7 more of said selectable settings in said panel as inputs therefrom,
8 and where at least one of said user selectable settings in said panel
9 corresponds to commands in said virtual machine;

10 (b) generating a display in accordance with one or more user
11 selected settings substantially contemporaneously with the selection
12 thereof;

13 (c) storing information representative of said one or more user
14 selected settings in a database;

15 (d) generating a website at least in part by retrieving said
16 information representative of said one or more user selected settings
17 stored in said database; and

18 (e) building one or more webpages to generate said website from at
19 least a portion of said database and at least one run time file, where
20 said at least one run time file utilizes information stored in said
21 database to generate virtual machine commands for the display of at
22 least a portion of said one or more web pages.

23 III. DISCUSSION

24 As explained in *Alice*, the Supreme Court has “interpreted § 101 and its predecessors ... for
25 more than 150 years” to “‘contain[] an important implicit exception: Laws of nature, natural
26 phenomena, and abstract ideas are not patentable.’ ” The *Alice* court applied a two-step framework
27 for determining patent eligibility, previously articulated in *Mayo Collaborative Servs. v.*
28 *Prometheus Labs., Inc.*, 132 S.Ct. 1289 (2012):

First, we determine whether the claims at issue are directed to one of
those patent-ineligible concepts. If so, we then ask, “[w]hat else is

other asserted claims.

there in the claims before us?” To answer that question, we consider the elements of each claim both individually and “as an ordered combination” to determine whether the additional elements “transform the nature of the claim” into a patent-eligible application. We have described step two of this analysis as a search for an “inventive concept”—i.e., an element or combination of elements that is sufficient to ensure that the patent in practice amounts to significantly more than a patent upon the [ineligible concept] itself.

Alice, 134 S.Ct. at 2355.

Alice also explained, “The ‘abstract ideas’ category embodies “the longstanding rule that ‘[a]n idea of itself is not patentable.’ ” *Id.* at 2355; *see also Le Roy v. Tatham*, 14 How. 156, 175, 14 L.Ed. 367 (1853). (“A principle, in the abstract, is a fundamental truth; an original cause; a motive; these cannot be patented, as no one can claim in either of them an exclusive right”).

Alice repeated the caution given in *Mayo*, however, that the exclusion for “abstract ideas” must not be applied too broadly, “we tread carefully in construing this exclusionary principle lest it swallow all of patent law.” 134 S.Ct. at 2354 (*citing Mayo*, 132 S.Ct. at 1293–1294.) At some level, “all inventions . . . embody, use, reflect, rest upon, or apply laws of nature, natural phenomena, or abstract ideas.” *Mayo*, 132 S.Ct. at 1293.

On the facts before it, the *Alice* court also expressly declined to “labor to delimit the precise contours of the ‘abstract ideas’ category.” 134 S.Ct. at 2357. Instead, it merely found that the concept of providing an “intermediated settlement” was not meaningfully distinguishable from the idea of “risk hedging” at issue in *Bilski v. Kappos*, 561 U.S. 593 (2010). In both instances, the idea involved was “a fundamental economic practice long prevalent in our system of commerce.” *Id.* at 2356.

Here, defendants rely primarily on *Intellectual Ventures I LLC v. Capital One Bank (USA)*, 792 F.3d 1363 (Fed. Cir. 2015). They argue the patents here are not meaningfully distinguishable from one held invalid in that case, U.S. Patent No. 7,603,382, entitled “Advanced Internet Interface Providing User Display Access of Customized Webpages.” Defendants’ insistence to the contrary notwithstanding, the patents are simply not directly comparable. The ’382 patent in *Intellectual Ventures* “generally relate[d] to customizing web page content as a function of

navigation history and information known about the user.” 792 F.3d at 1369. The representative claim described “[a] system for providing web pages accessed from a web site in a manner which presents the web pages tailored to an individual user.” The *Intellectual Ventures* court had little trouble concluding that merely tailoring the information presented to a website user based on information about that user or when the website was being viewed represented patent-ineligible abstract ideas.

This sort of information tailoring is “a fundamental . . . practice long prevalent in our system. . . .” . . . There is no dispute that newspaper inserts had often been tailored based on information known about the customer—for example, a newspaper might advertise based on the customer’s location. Providing this minimal tailoring—e.g., providing different newspaper inserts based upon the location of the individual—is an abstract idea

Tailoring information based on the time of day of viewing is also an abstract, overly broad concept long-practiced in our society. There can be no doubt that television commercials for decades tailored advertisements based on the time of day during which the advertisement was viewed.

Id.

The patents here are not comparable merely because they also involve webpages that reflect information provided by a “user.” Indeed, the patents do not even involve the same category of “user”—here the “user” is the person who is trying to *create* webpages, in *Intellectual Ventures* the user is a person viewing the webpage to whom customized content will be delivered. The patents here are directed at a purportedly revolutionary technological solution to a technological problem—how to create webpages for the internet in a manner that permits “what you see is what you get” editing, and a number of other alleged improvements over the then-existing methodologies.

A more apt comparison is *Enfish, LLC v. Microsoft Corp.*, 822 F.3d 1327 (Fed. Cir. 2016), which reversed a district court’s finding of ineligibility under *Alice*. At issue in *Enfish* was “an innovative logical model for a computer database.” *Enfish* supports the notion that a dividing line can be drawn between patents which merely describe using a computer and/or the internet to carry

out pre-existing and well-known tasks and techniques, and those that relate to the functioning of computers themselves. The former will virtually always fail under *Alice* unless some “inventive concept” can be found in the second step of the analysis; the latter are substantially less easily characterized as merely abstract ideas.

Enfish explains:

The first step in the *Alice* inquiry in this case asks whether the focus of the claims is on the specific asserted improvement in computer capabilities (*i.e.*, the self-referential table for a computer database) or, instead, on a process that qualifies as an “abstract idea” for which computers are invoked merely as a tool. As noted *infra*, in *Bilski* and *Alice* and virtually all of the computer-related § 101 cases we have issued in light of those Supreme Court decisions, it was clear that the claims were of the latter type—requiring that the analysis proceed to the second step of the *Alice* inquiry, which asks if nevertheless there is some inventive concept in the application of the abstract idea. See *Alice*, 134 S.Ct. at 2355, 2357–59. In this case, however, the plain focus of the claims is on an improvement to computer functionality itself, not on economic or other tasks for which a computer is used in its ordinary capacity.

822 F.3d at 1335–36.

Enfish drew a line between “improvement[s] to computer functionality itself,” and “economic or other tasks for which a computer is used in its ordinary capacity.” The court concluded:

we find that the claims at issue in this appeal are not directed to an abstract idea within the meaning of *Alice*. Rather, they are directed to a specific improvement to the way computers operate”

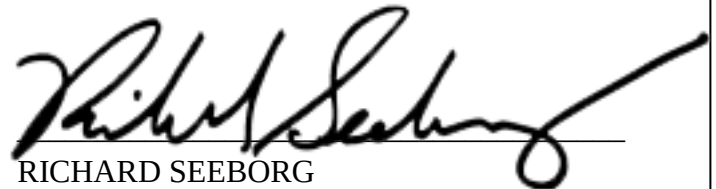
Id. at 1336.

So too here.

Finally, to the extent that defendants are arguing that any potentially patent-eligible technological improvements set out in the specification are not reflected in the actual claims, dismissal under *Alice* is not appropriate, at least at this juncture. Although some claim construction has taken place, it simply cannot be said on the present record that the claims are drawn so broadly as to be divorced from the potentially patent-eligible purported technological improvements described in the specification. Accordingly, the motions to dismiss are denied.

IT IS SO ORDERED.

Dated: January 29, 2019



RICHARD SEEBORG
United States District Judge

United States District Court
Northern District of California

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28