US 20060075070A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2006/0075070 A1

(43) Pub. Date: Apr. 6, 2006

Merissert-Coffinieres et al.

(54) DEVELOPMENT AND DEPLOYMENT OF MOBILE AND DESKTOP APPLICATIONS WITHIN A FLEXIBLE MARKUP-BASED DISTRIBUTED ARCHITECTURE

(76) Inventors: **Patrick Merissert-Coffinieres**, Issy Les Moulineaux (FR); **Alain Dinkespiler**, Paris (FR); **Alejandro Daniel Rosenblatt**, Toulouse (FR); **Alexandra-Madalina Stroe**, Cupertino, CA (US)

Correspondence Address:
MORRISON & FOERSTER LLP
755 PAGE MILL RD
PALO ALTO, CA 94304-1018 (US)

(21) Appl. No.: **11/295,028**

(22) Filed: **Dec. 5, 2005**

Related U.S. Application Data

(63) Continuation of application No. 10/116,310, filed on Apr. 2, 2002.

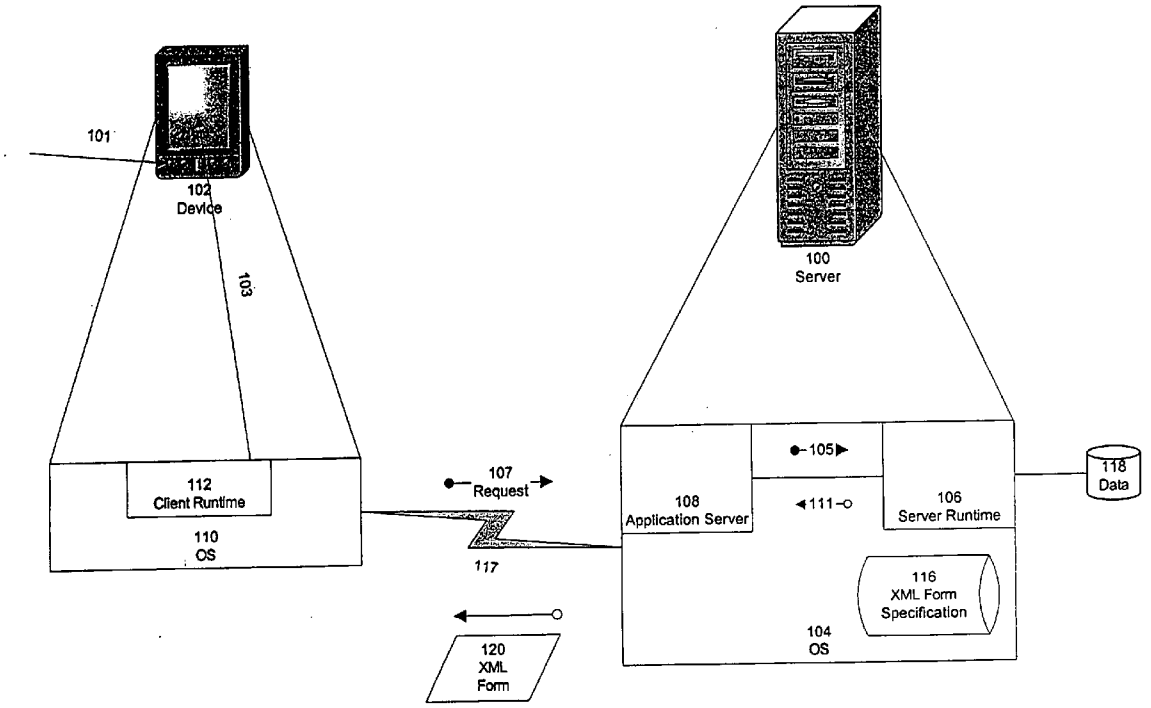
Publication Classification

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.** **709/218**

(57) **ABSTRACT**

A computer program product embodied in a computer-readable medium is configurable to accomplish execution of an application that is specified and encoded in a markup-based descriptor language. The product includes client runtime computer code configured to cause a client computer device to process the markup-based descriptor language to deploy an application to accomplish execution of the application. The client runtime computer code is further configured to process the markup-based descriptor language to selectively configure the client computer device to deploy the application so as to accomplish execution of the application by the client computer device stand-alone or by the client computer device in cooperation with a server to which the device is connectable via a network connection.



Instacart, Ex. 1057

FIGURE 1

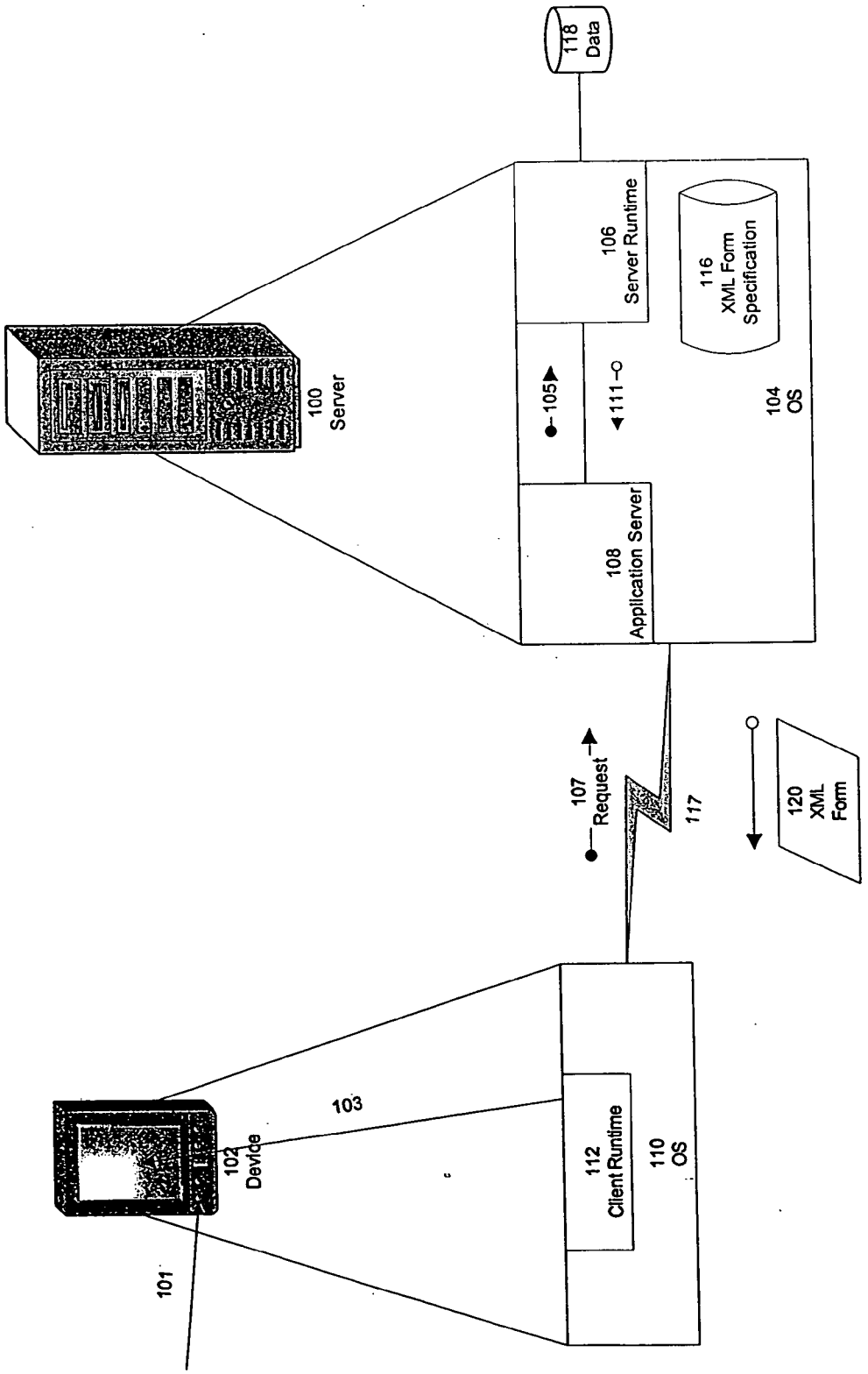

118
Data
100
Server
106
Server Runtime
116
XML Form
Specification
105
111
104
OS
108
Application Server
107
Request
117
120
XML
Form
101
102
Device
103
112
Client Runtime
110
OS

FIGURE 2

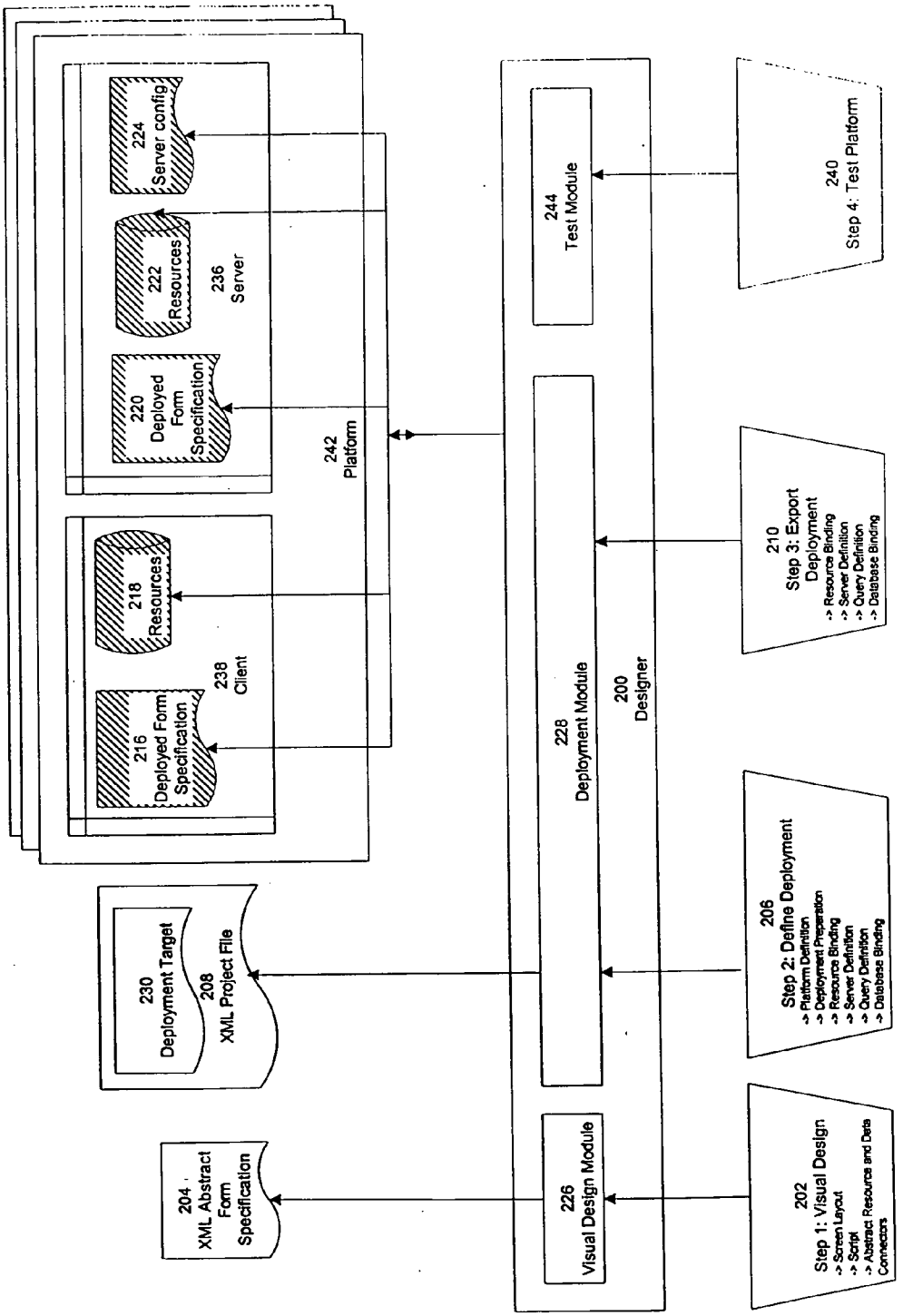
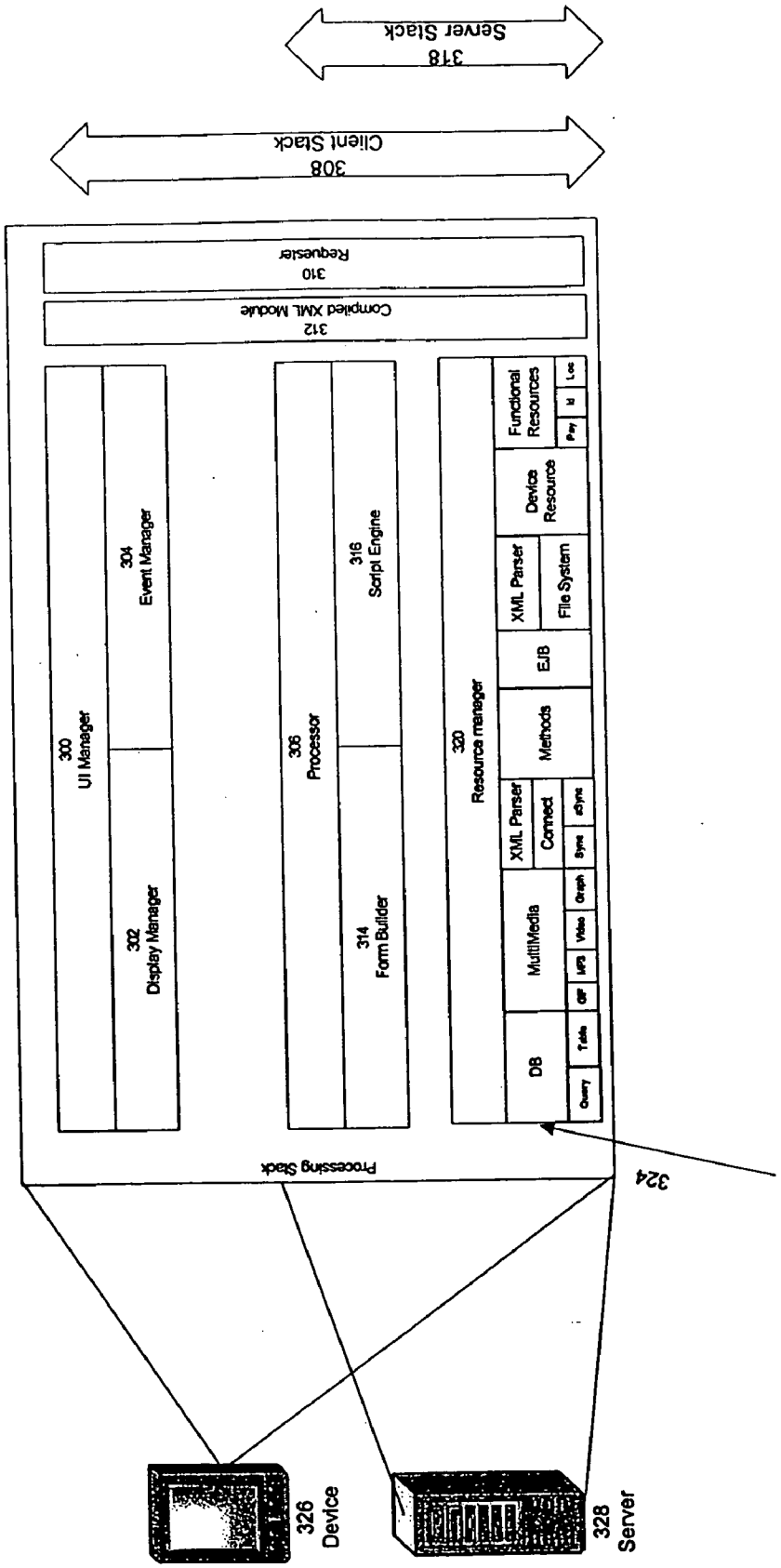

224
Server config
222
Resources
236
Server
220
Deployed
Form
Specification
242
Platform
218
Resources
238
Client
216
Deployed Form
Specification
230
Deployment Target
208
XML Project File
204
XML Abstract
Form
Specification
244
Test Module
228
Deployment Module
200
Designer
226
Visual Design Module
240
Step 4: Test Platform
210
Step 3: Export
Deployment
-> Resource Binding
-> Server Definition
-> Query Definition
-> Database Binding
206
Step 2: Define Deployment
-> Platform Definition
-> Deployment Preparation
-> Resource Binding
-> Server Definition
-> Query Definition
-> Database Binding
202
Step 1: Visual Design
-> Screen Layout
-> Script
-> Abstract Resource and Data
Connectors

FIGURE 3


318
Server Stack
308
Client Stack
310
Requester
312
Compiled XML Module
300
UI Manager
302
Display Manager
304
Event Manager
306
Processor
314
Form Builder
316
Script Engine
320
Resource manager
DB
Query
Table
MultiMedia
GIF
MPG
Video
Graph
XML Parser
Connect
Sync
eSync
Methods
EJB
XML Parser
File System
Device Resource
Functional Resources
Pay
Id
Loc
Processing Stack
324
326
Device
328
Server

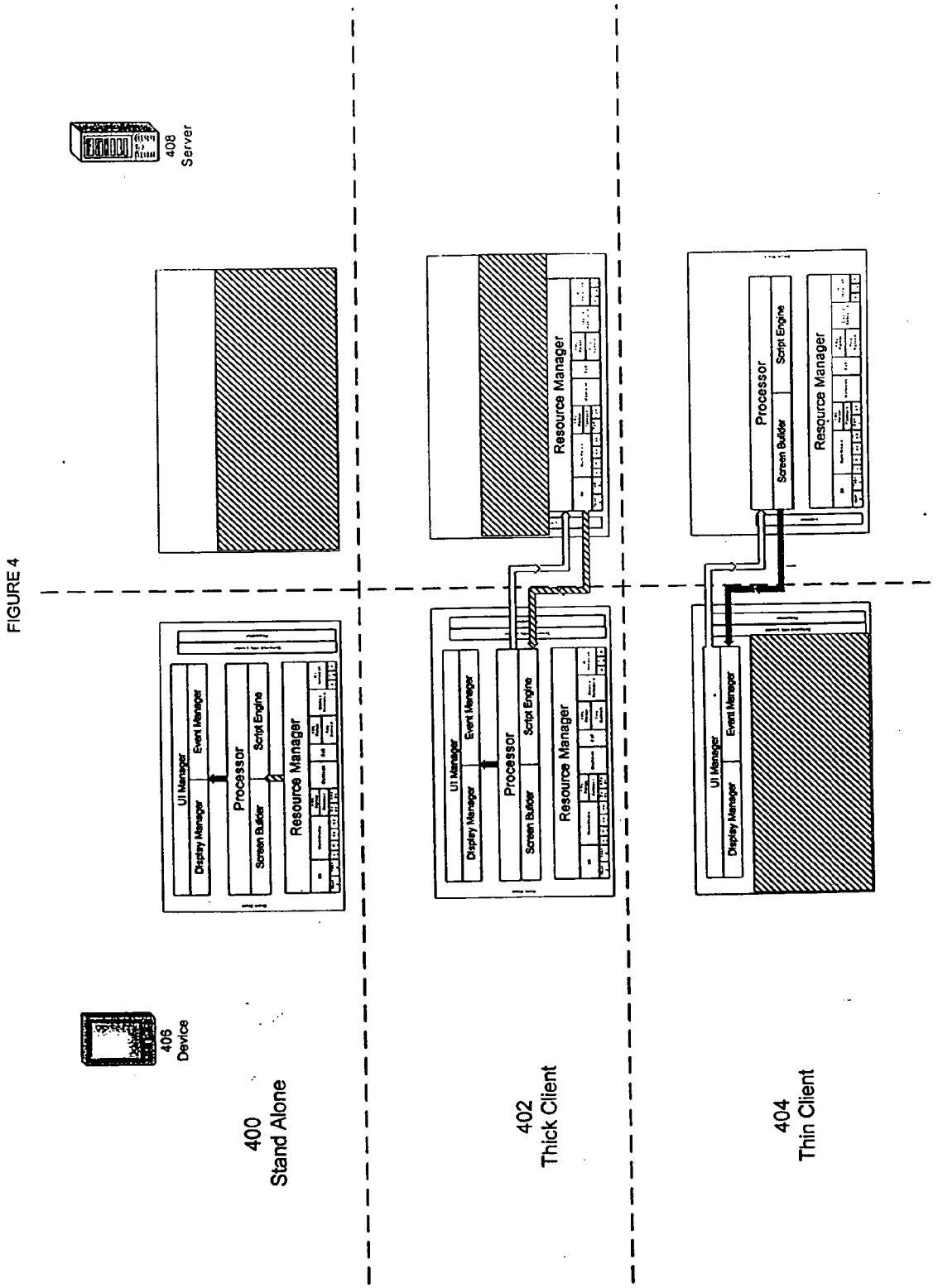

408
Server
406
Device
UI Manager
Event Manager
Display Manager
Processor
Script Engine
Screen Builder
Resource Manager
400
Stand Alone
402
Thick Client
404
Thin Client

FIGURE 4

FIGURE 5

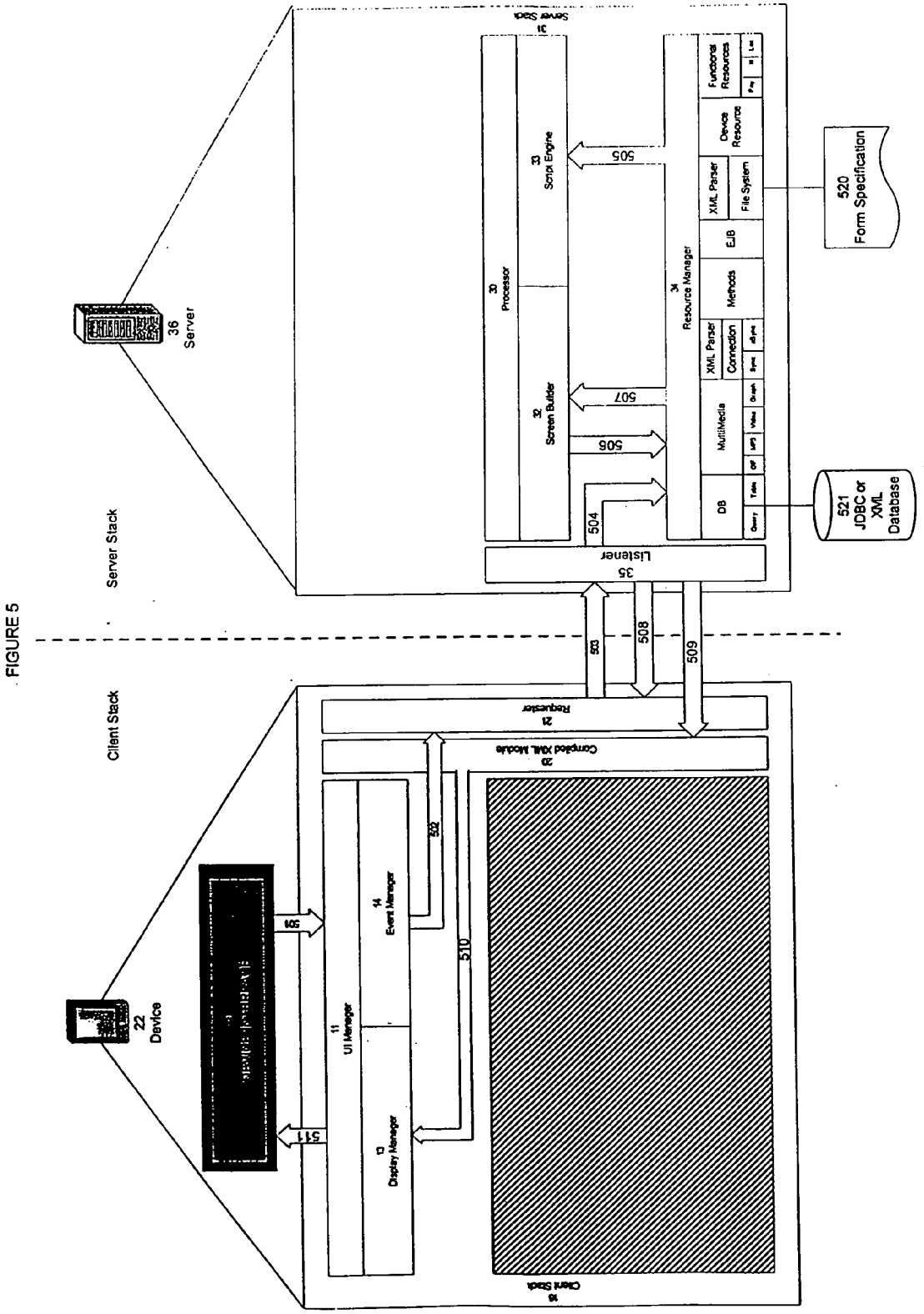
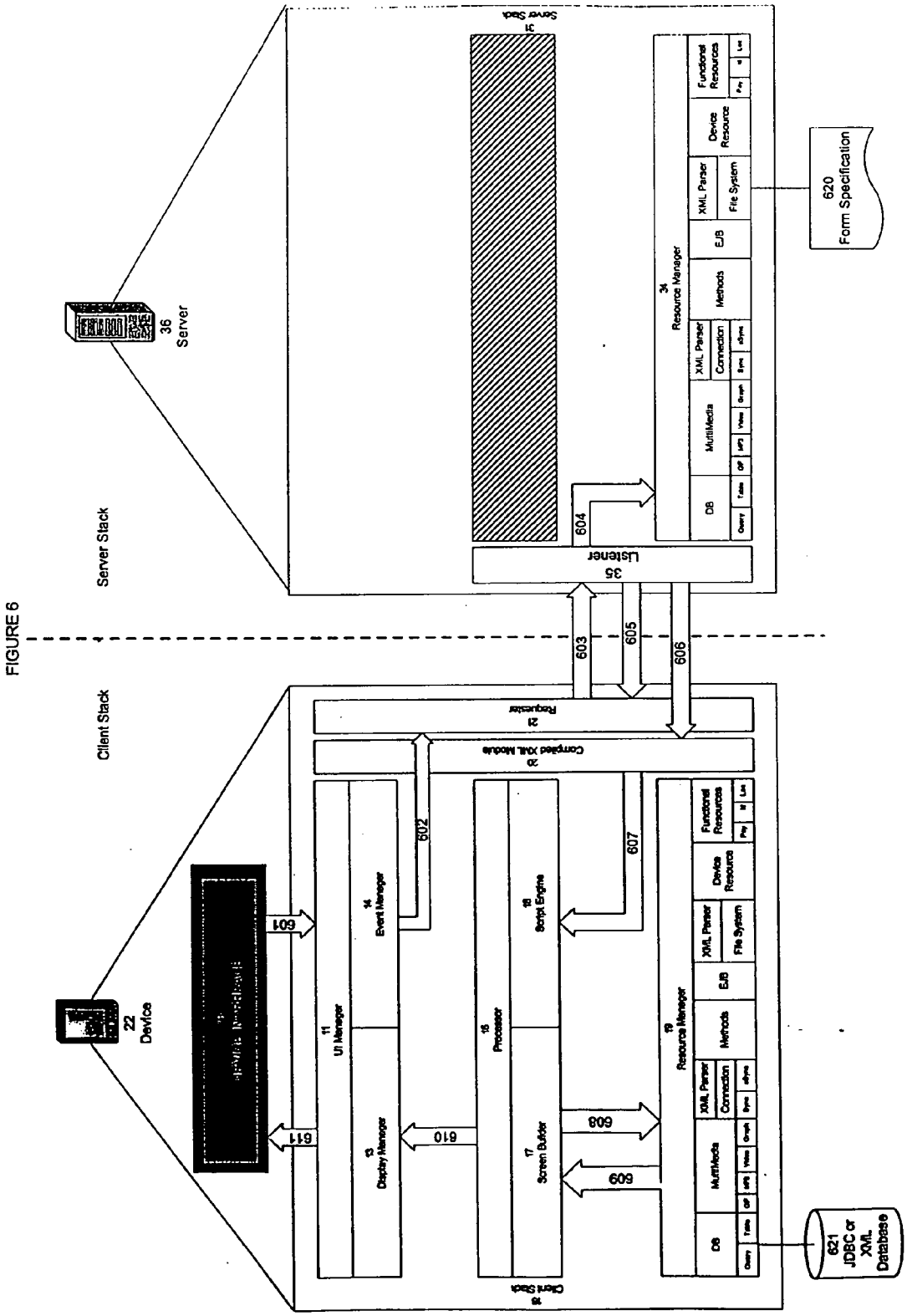

Server Stack
Client Stack
36 Server
22 Device
31 Server Stack
30 Processor
33 Script Engine
32 Screen Builder
34 Resource Manager
DB
MultiMedia
XML Parser
Connection
Methods
EJB
XML Parser
File System
Device Resource
Functional Resources
520 Form Specification
521 JDBC or XML Database
35 Listener
504
505
506
507
508
509
503
21 Requester
20 Compiled XML Module
11 UI Manager
14 Event Manager
13 Display Manager
502
510
511
18 Client Stack

FIGURE 6


Server Stack
Client Stack
36
Server
22
Device
31
Server Stack
35
Listener
34
Resource Manager
DB
MultiMedia
XML Parser
Connection
Methods
EJB
XML Parser
File System
Device Resource
Functional Resources
620
Form Specification
21
Requester
20
Compiled XML Module
11
UI Manager
14
Event Manager
13
Display Manager
15
Processor
18
Script Engine
17
Screen Builder
19
Resource Manager
601
602
603
604
605
606
607
608
609
610
611
621
JDBC or
XML
Database

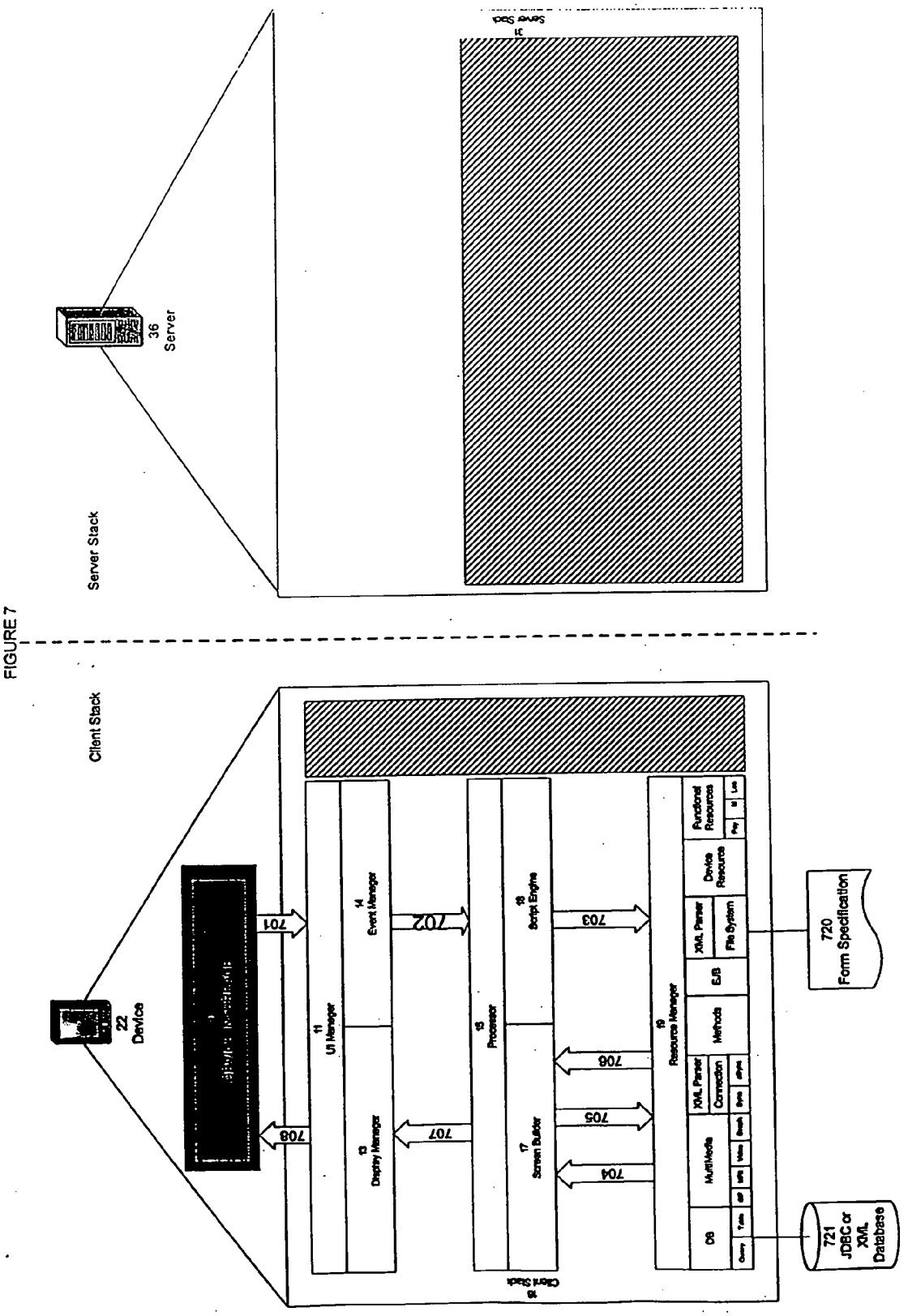
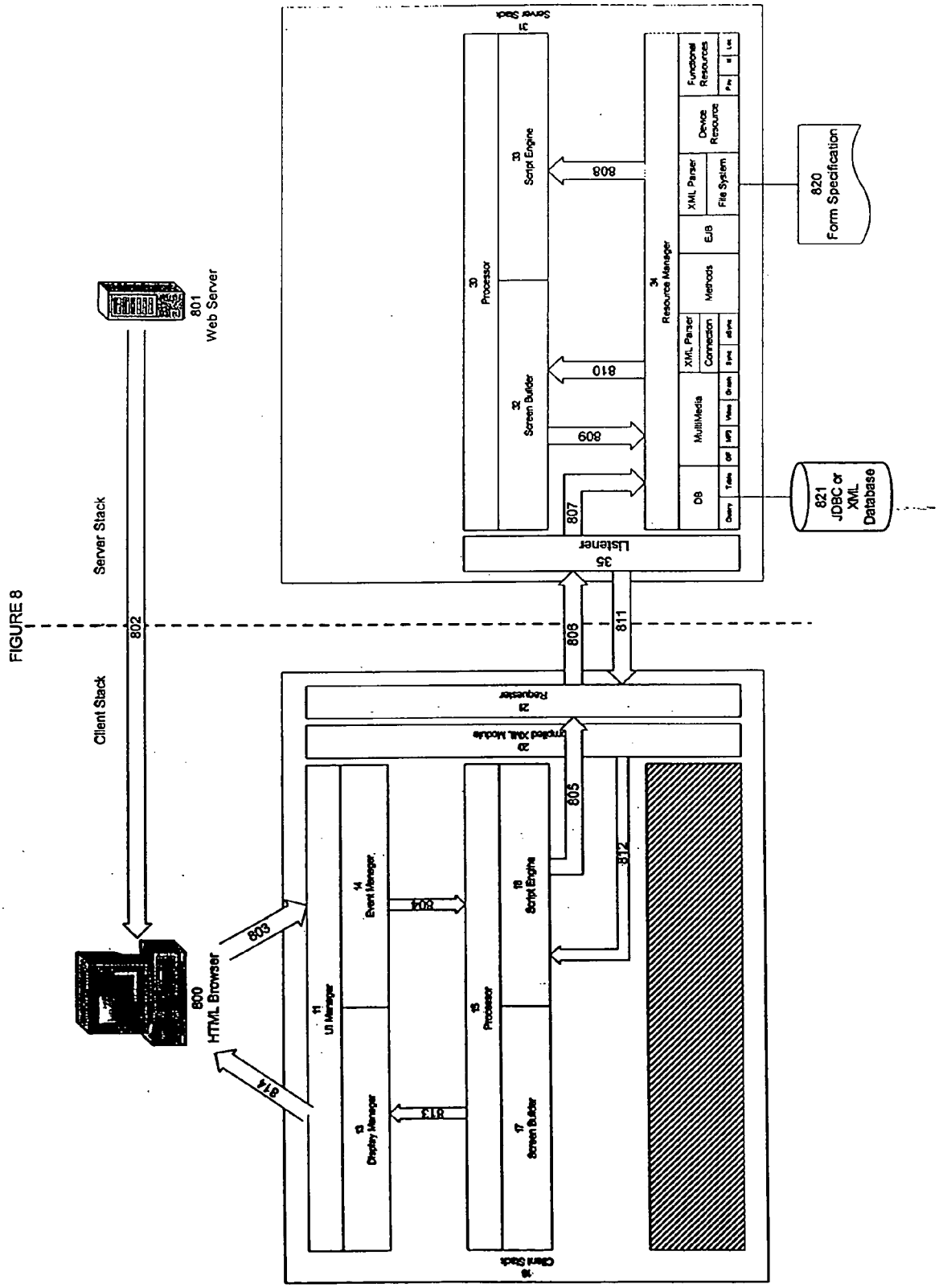

Server Stack
36
Server
Client Stack
22
Device
11
UI Manager
14
Event Manager
13
Display Manager
15
Processor
18
Script Engine
17
Screen Builder
19
Resource Manager
DB
MultiMedia
XML Parser
Connection
Methods
EJB
XML Parser
File System
Device Resource
Functional Resources
701
702
703
704
705
706
707
708
720
Form Specification
721
JDBC or
XML
Database

FIGURE 7

FIGURE 8


Server Stack
Client Stack
801
Web Server
802
800
HTML Browser
803
804
805
806
807
808
809
810
811
812
813
814
Server Stack
31
30
Processor
32
Screen Builder
33
Script Engine
34
Resource Manager
DB
MultiMedia
XML Parser
Connection
Methods
EJB
XML Parser
File System
Device Resource
Functional Resources
35
Listener
820
Form Specification
821
JDBC or
XML
Database
Client Stack
11
UI Manager
13
Display Manager
14
Event Manager
15
Processor
17
Screen Builder
18
Script Engine
20
Compiled XML Module
21
Requester

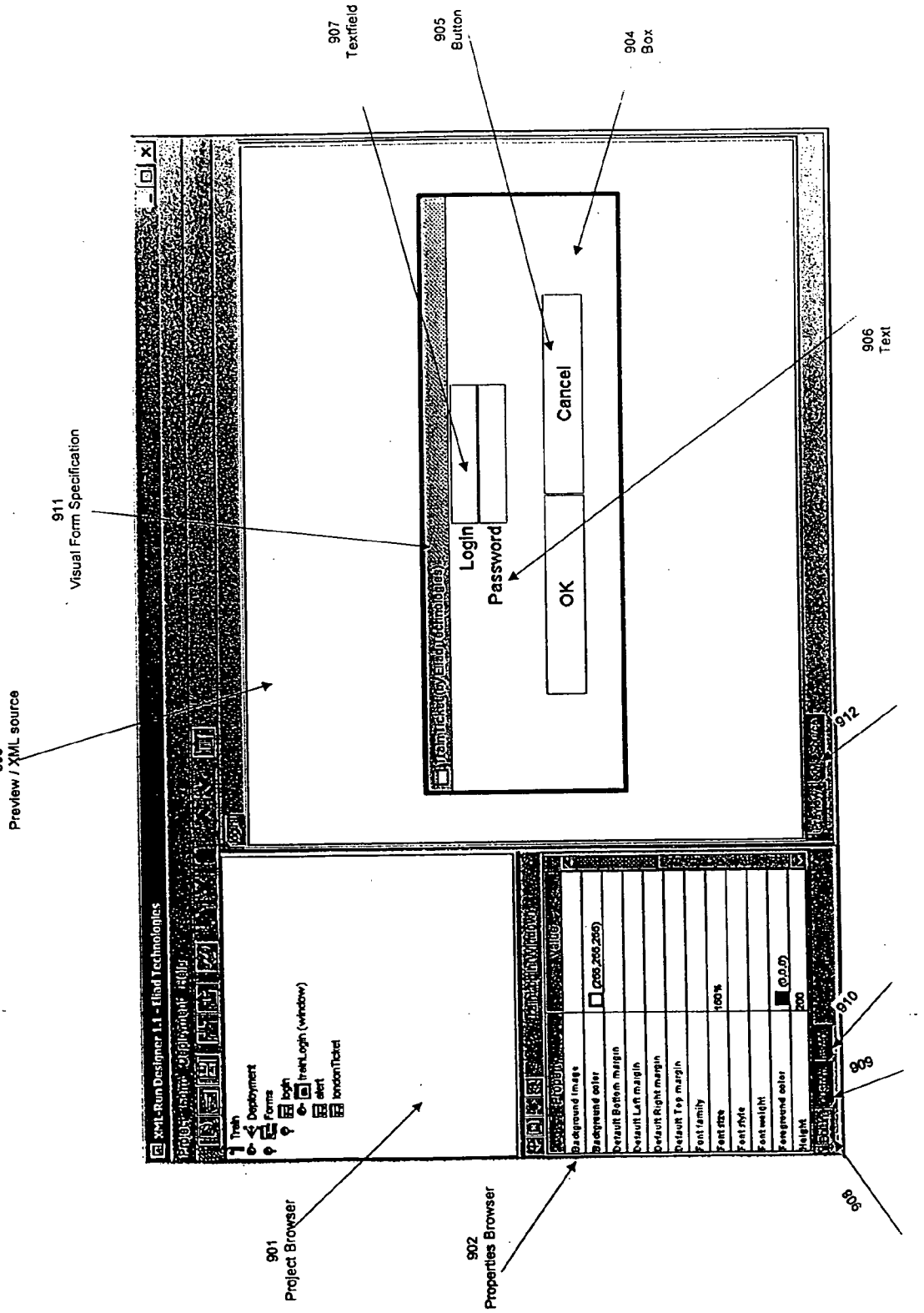
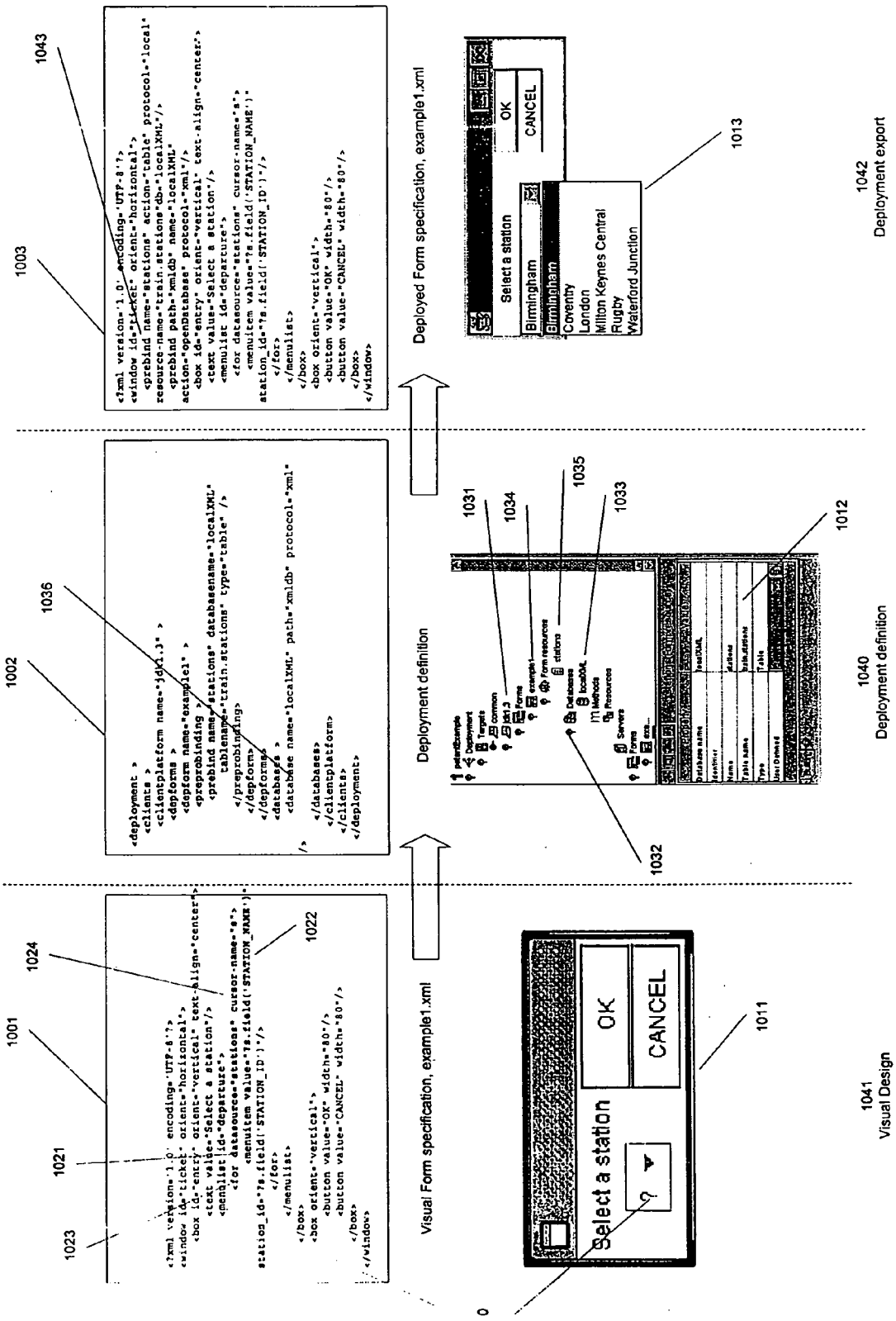

907
Textfield
905
Button
904
Box
906
Text
911
Visual Form Specification
903
Preview / XML source
901
Project Browser
902
Properties Browser
912
910
909
908
Login
Password
Cancel
OK

FIGURE 9

FIGURE 10


1001
1002
1003
1011
1012
1013
1021
1022
1023
1024
1031
1032
1033
1034
1035
1036
1040
1041
1042
1043
Visual Form specification, example1.xml
Deployment definition
Deployed Form specification, example1.xml
Select a station
OK
CANCEL
Birmingham
Coventry
London
Milton Keynes Central
Rugby
Waterford Junction
Deployment definition
Deployment export
Visual Design

DEVELOPMENT AND DEPLOYMENT OF MOBILE AND DESKTOP APPLICATIONS WITHIN A FLEXIBLE MARKUP-BASED DISTRIBUTED ARCHITECTURE

TECHNICAL FIELD OF THE INVENTION

[0001] The present invention is in the field of developing and deploying applications and services for a full range of devices, scaling from mobile phones to computer desktops, within a flexible markup based distributed architecture.

BACKGROUND OF THE INVENTION

[0002] As wireless computing devices become more ubiquitous, the demand for value-added mobile applications and solutions to deliver these applications increases. Enterprises expect new mobile applications to seamlessly integrate with the existing corporate information system, thus granting access to existing resources and services to highly mobile workers. Examples include accessing legacy applications and connecting to enterprise databases through wireless devices.

[0003] Mobile applications may have challenges such as:

[0004] Accessing a mobile application from different kinds of devices, for example from a PDA or a desktop computer, depending on the location of the user. PDAs and desktops differ in several ways, including form factors, operating systems and connectivity.

[0005] Running a mobile application on a mobile device continuing to work offline, in the case of a broken connection. An offline behavior utilizes access to local resources.

[0006] A mobile application accessing services published by some Web Services provider implementing Web Services standards, including SOAP, WSDL and UDDI.

[0007] Deploying mobile applications in such heterogeneous environments greatly increases the complexity of the solutions, and presents many unresolved challenges. These challenges in turn affect the development of applications, as well as ongoing maintenance. Some issues to address are:

[0008] Open standards compatibility. Proprietary solutions are unlikely to be practical in such heterogeneous environments. The solution should fit the emerging Web Services standards, including SOAP, WSDL and UDDI.

[0009] Multi-platform deployment. The same service may be accessed from different kinds of devices and different communication protocols may be used.

[0010] Ease of modification. Since the markets change quickly, especially emerging markets like the wireless applications market, it is desirable that smart applications are easy and quick to modify.

[0011] Adaptable deployment. No single deployment model fits all contexts. The suitable deployment model depends on several factors, such as device resources, security requirements and application characteristics. Deployment models typically range from a thin client connected to a server, to a stand-alone running application.

[0012] Access to local resources. Mobile devices cannot rely exclusively upon server side resources, since the connection with a server cannot always be guaranteed. Furthermore, it can be costly to continually maintain a connection to a server when it is only occasionally required.

[0013] Enhanced user interface. Required to improve the user experience.

[0014] Solutions like HyperText Markup Language (HTML) combined with HyperText Transfer Protocol (HTTP), Wireless Application Protocol (WAP), pure Java programming, Application Servers, and proprietary Software Development Kits (SDKs), each address a subset of these challenges, but more comprehensive solutions are desired.

[0015] HTML and WAP are open standards and may be deployed on multiple platforms. Services developed with these standards may be easily modified, since they become immediately available to the client devices when deployed on the server. Unfortunately, the deployment model of HTML and WAP is a rather rigid thin client model and a connection to a server must remain typically available during the execution of the application. This may be costly and even impractical, should the connection to the server be interrupted unexpectedly. Another drawback is that these solutions allow little access, if any, to local resources. Finally, the user interface is rather modest, at best.

[0016] Another solution mentioned above is pure Java programming. Although Java is quite portable, and is considered to some extent as an "open standard", there are in fact multiple Java standards, including J2ME MIDP, Personal Java, and Java 2 Standard Edition. Thus, deploying a Java application on different platforms may require rewriting it several times, which implies involving highly-skilled developers. The same problem arises when modifying an application. Java provides good means of implementing a deployment model targeted to a specific architecture, accessing local resources and providing effective User Interfaces. Unfortunately, each deployment model generally requires a specific Java program, thus making it unfeasible to dynamically adapt a given application to new architectural requirements.

[0017] Proprietary SDK's provided by mobile device manufacturers are quite comparable in capabilities and drawbacks to the Java programming approach, but have the additional drawback of not being open standards. The use of such proprietary solutions entails a commitment to one particular (inflexible) solution, which in turn restricts the ability to later port an application to other mobile devices. Should the chosen solution at some point no longer meet the requirements of the user, it could be extremely costly to implement a completely new solution.

[0018] Application servers, and especially Java application servers based on the Java 2 Enterprise Edition (J2EE) standard, promote several architecturally significant separations of concerns. One architectural feature is a variation on the HTML approach, namely Java Server Pages (JSP). JSPs are targeted to improve the separation between interaction logic and business logic on the server side, but have the same limitations as traditional HTML architectures on the client side. Another architectural feature is the distinction between

the development phase and the deployment phase for an application, thus allowing some flexibility for adapting to the underlying technical architecture. However, the deployment process is typically only concerned about server-side deployment characteristics, including transactions, security and database access. The distribution of processing and resources between the client and the server are not addressed by this solution.

SUMMARY OF THE INVENTION

[0019] A computer program product embodied in a computer-readable medium is configurable to accomplish execution of an application that is specified and encoded in a markup-based descriptor language. The product includes client runtime computer code configured to cause a client computer device to process the markup-based descriptor language to deploy an application to accomplish execution of the application. The client runtime computer code is further configured to process the markup-based descriptor language to selectively configure the client computer device to deploy the application so as to accomplish execution of the application by the client computer device stand-alone or by the client computer device in cooperation with a server to which the device is connectable via a network connection.

BRIEF DESCRIPTION OF THE FIGURES

[0020] **FIG. 1** illustrates an outline of the runtime architecture, in accordance with an embodiment of the present invention.

[0021] **FIG. 2** illustrates an outline of the development and deployment process, in accordance with an embodiment of the present invention.

[0022] **FIG. 3** illustrates the processing runtime client and server stacks, in accordance with an embodiment of the present invention.

[0023] **FIG. 4** illustrates a description of three major deployment profiles supported by preferred embodiments of the present invention.

[0024] **FIG. 5** illustrates the flow of information within the client and server stacks of an embodiment of the present invention, for the thin connected profile.

[0025] **FIG. 6** illustrates the flow of information within the client and server stacks of an embodiment of the present invention, for the thick connected profile.

[0026] **FIG. 7** illustrates the flow of information within the client stack of an embodiment of the present invention, for the stand-alone profile.

[0027] **FIG. 8** illustrates the flow of information within the client and server stacks of an embodiment of the present invention, for the applet profile.

[0028] **FIG. 9** illustrates an embodiment of a visual design module of the present invention

[0029] **FIG. 10** illustrates embodiments of the markup based descriptors and corresponding screens displayed by embodiments of the present invention

DETAILED DESCRIPTION

[0030] A comprehensive flexible solution, based on a markup language like XML, for developing smart mobile applications and services, and for deploying them in heterogeneous environments. Smart mobile applications can be used in a connected or disconnected mode and they can access and process resources locally or on a server. According to one aspect, a flexible client-server architecture is targeted to run smart mobile applications. The applications and the deployment architecture are specified within markup-based descriptors. The supported architecture specifications range from a standalone client running an entire unconnected application, to a thin client managing the User Interface with the business logic of the application running on the server. In between, many combinations of client and server side processing may be specified. The characteristics of the architecture may be dynamically modified, thus leading to increased flexibility in the deployment. Mostly, the markup-based descriptors are forms specifying the User Interface of the application, the behavior of the application in response to the interactive events, the business logic of the application, and the location where the resources are to be found and processed. The behavior of the application is defined in a scripting language, which is given access to the resources of the device, including access to methods written in a language like Java, either on the client or on the server.

[0031] For example, a client runtime system displays interactive screens, or forms, interacting with specific server components running within an application server. The client runtime retrieves a form specification, either locally or from the server. When the form is accessed from the server, it may be preprocessed therein. Otherwise, it is preprocessed on the client. The preprocessing of a form specification is a construction process, where the actual form to be displayed is constructed as specified in the form specification, possibly using available resources like databases to populate the actual form. The client runtime parses and processes the actual form to generate the User Interface on the device, and then handles the interactive events. The client runtime is typically installed on the computing client device, which may be a smart phone, a PDA, or a desktop computer, but it can also be downloaded on demand as an applet into a browser. The server software is typically installed on a server running a Java application server. The entry point in the server software is a servlet using an XML configuration file generated by the Designer. When a client device sends a request, the servlet analyzes the request, prepares the XML formatted response, and sends it back to the client. The preparation process may include preprocessing of a form specification, and retrieval of deployed forms and/or data from backend systems.

[0032] According to another aspect, a development and a deployment process is provided for smart applications and services. For example, the development and deployment process may be performed in three steps. The first step is the user interface and interaction design step, and produces the interactive specification of the forms. Within this step, databases and other resources are managed and accessed as abstract references, without any indication about their physical location or other implementation-dependent characteristics. The second step is the deployment definition step, that is, defining where and how the form specifications, resources and data are to be deployed on the client and the servers. The third step corresponds to the actual deployment on the target platforms. This separation provides means for deploying the same application across multiple kinds of distributed environments. The environments may differ by

the operating system of the client device, by the resources available on the client or on the server, and by the characteristics of the connection. The result of the development and the deployment process is a set of markup documents describing the architecture and the behavior of the resulting application.

[0033] As an example of this development and deployment process, an easy-to-use Rapid Application Development (RAD) designer tool may be provided for application developers. The designer tool running on the developer's computing device provides the developer with tools for each step of the development and deployment process, from the visual definition of screens to the testing of full-fledged client-server applications. With the designer tool, the developer can define and edit the screen visually. The designer tool creates the XML files required by the client and by the server.

[0034] The same application may thus be deployed on different deployment models and platforms, providing increased flexibility (useful, for example, for wireless implementations). For example, if the application is designed to support the sale force of a large corporation equipped with both wireless PDAs and desktop computers, different types of deployments may be required. In the office, the user may expect to access the application through a thick connected model, using the full processing power of a desktop computer and the high bandwidth of the local network. While outside the office, the user may access the application, for instance to check the status of an order, from either a PDA using a thin connected model or from a browser through the Internet by using an applet deployment mode. When a connection is not available, the application may operate in a stand-alone mode, thus allowing the user to enter an uncommitted order or estimate a total price.

[0035] **FIG. 1** illustrates, at a broad level, the runtime architecture of one embodiment in accordance with the present invention. The runtime architecture is implemented on a computing client device **102** and a server **100**, connected through a connection link **117**. The illustrated embodiment corresponds to a "thin client" runtime configuration.

[0036] The computing client device **102** may include two or more layers. The operating system **110** interfaces to the hardware and handles the low level management of the resources on the device, such as memory, storage, and user input. A Java Virtual Machine is considered an extension of the operating system, abstracting the device's resources into the Java standard, and providing a standard programming platform across devices. A third layer is the client runtime **112**, providing part of the client side functionality. The server **100** may comprise a Java J2EE compatible application server **108**, which manages the services available on the server. The server runtime **106** based on a Java servlet provides part of the server side functionality in accordance to the present invention. The communication link **117** may be an Internet connection based on the HTTP protocol.

[0037] When, through an interaction **101** with the computing client device **102**, a user requests access to a form from a service located on the server **100**, the client runtime **112** sends a request **107** to the server **100** using the available communication link and protocol **117**. On the server **100**, the application server **108** analyzes the request and forwards it to the server runtime **106**. The server runtime processes the request, recognizes it as a form request, retrieves the requested form specification stored in an XML file **116** and additional data from external resources **118** as appropriate, and combines them to construct the preprocessed XML form **120**, which is passed back to the client device **102**. The client runtime **112** then processes the XML form and generates **103** the corresponding User Interface on the device's display.

[0038] This general architecture described with reference to **FIG. 1** illustrates a deployment with a thin client profile where the construction of the form occurs on the server and no local storage is used on the client. However, other deployment profiles may be employed. The following deployment profiles will be later discussed in more detail:

[0039] Stand-alone: the form specifications are stored, and the displayable interactive forms constructed and processed on the client, and no connection to a server is utilized.

[0040] Thick connected client: the form specifications are retrieved from a server, but the displayable interactive forms are constructed and processed on the client. Local resources, such as local databases may be involved.

[0041] Thin Client: the form specifications are stored on the server. The displayable interactive forms are constructed on the server and the preprocessed forms are then passed to the client. The client handles the processing of the events once the form is on the device, and no access to client resources is utilized.

[0042] Applet deployment: the runtime is not installed on the client; it is downloaded on request from the server and runs inside a browser.

[0043] The named deployment profiles are not fixed during the execution of a given application. For example, the frequently used forms of an application may run in a stand-alone mode, and the less frequently used ones may be deployed in a thin or thick client mode. Furthermore, the present invention provides means for dynamically changing the deployment profile during the execution of an application, thus leading to highly flexible architectures.

[0044] **FIG. 2** illustrates, at a broad level, an embodiment of the development and deployment process in accordance with the present invention. A Rapid Application Development (RAD) designer tool **200** running on the developer's computing device is provided. The designer tool **200** comprises three modules: the visual design module **226**, the deployment module **228**, and the test module **244**. The process is carried out in four major steps: the visual design step **202**, the deployment definition step **206**, the export deployment step **210**, and the test platform step **240**.

[0045] The visual design step **202** includes defining the abstract form specifications **204** of the application. An abstract form specification is stored as an XML document. The developer designs the visual aspect of the application's forms, by interacting with the visual design module **226**, typically by positioning visual components on the screen. The developer may specify the interactive behavior of each visual component through script code, as required. Capability is provided for script code to access external resources as abstract references, delaying the binding to the actual

resources until the deployment definition step **206**. As a result, the visual layout and behavior of the applications are specified independently of the target platforms. External resources the script code can manipulate include Java methods, images and databases. While presenting the visual aspect of a form, the designer tool **200** generates an XML representation **204** containing the full specification of the abstract form, including the interactive behavior of the form. The developer may choose to perform part or all of the tasks of the visual design step **202** by directly editing this file, using the text-editing feature of the visual design module **226**.

[**0046**] During the deployment definition step **206**, the developer defines one or more deployment targets **230** for the current application using the deployment module **228**. The characteristics of the deployments for an application are stored in an XML document, namely the project file **208**. Each deployment target **230** corresponds to a specific platform **242**, and comprises the description of the client profile and the accessed servers. For each abstract resource referenced during the visual design step **202**, the developer specifies its actual location and settings.

[**0047**] During the export deployment step **210**, the developer actually deploys the required files on the client device **238** and the server **236**, using the deployment module **228** of the designer tool **200**. When the developer requests the deployment module **228** to export the deployment, the designer tool **200** creates all the files required by the deployment and copies the files and the resources either to the client or to the server according to the deployment definition.

[**0048**] During the test platform step **240**, the developer tests the application in the context of each of the platforms **242** defined during the previous development steps, using the test module **244**.

[**0049**] **FIG. 3** illustrates the client and server processing runtime stacks of various architecture embodiments. The processing stack comprises three principal layers: the resource manager **320**, the processor **306**, and the User Interface (UI) Manager **300**. Two types of stacks are defined: the client stack **308**, which includes the three layers named above, and the server stack **318** which excludes the UI Manager **300**. By providing similar processing stacks on the client device **326** and on the server **328**, improved flexibility in the deployment of applications, is provided since most of the applications' logic may be run uniformly on either the client or on the server.

[**0050**] Within the processor **306**, the form builder **314** handles the transformation of form specifications into displayable interactive forms. The form builder **314** analyzes the form specification to check if additional resources are required, including data, image or methods. The requests for the resources are passed to the resources manager **320**, as needed. For example, the form specification may specify that a visual grid be filled with data extracted from a database table. The form builder **314** is in charge of performing this task. The developer may use script code within the form specification to define how the form may be constructed, making the construction process more flexible. If the form contains script code, the script engine **316** compiles it.

[**0051**] The UI Manager **300** interacts with the input and output capabilities of the client device **326**, such as screen, keyboard, touch screen, sound card or mouse. These elements are specific to the device and are managed by the operating system of the device. The display manager **302** handles the creation of the visual layout of the forms, using the visual controls available on the client device, such as buttons, text fields or grids, as expressed in the displayable interactive form created by the form builder **314**. When interest for an event has been registered within a form, the event manager **304** is in charge of notifying the corresponding component of the form whenever the event occurs, and if script code has been associated with the event on the component, the script engine **316** is in charge of running the corresponding code.

[**0052**] The resource manager **320** receives the requests for external resources, either from the processor **306** or from the event manager **304**. The resource manager **320** includes connectors **324** to access the different types of resources. Available resources include, for example:

[**0053**] Databases

[**0054**] Table: a table from a database, such as JDBC and XML databases

[**0055**] Query: an SQL query on a JDBC database or an XML query to an XML file

[**0056**] Multimedia

[**0057**] Image file: a file containing an image

[**0058**] Multimedia stream: a link to a video or sound streaming server or to a file containing video or sound data

[**0059**] Graphics

[**0060**] Network Files

[**0061**] XML file: an XML document stored as a file, which is parsed into a Document Object Model (DOM) tree

[**0062**] Methods

[**0063**] A Java class: a user defined class implementing some complex business logic

[**0064**] EJB: Entreprise Java Beans

[**0065**] File System: access to the local storage

[**0066**] Other Device Resources

[**0067**] Device specific resource (IrDA, bar code)

[**0068**] Functional Resources

[**0069**] Payment: secure payment

[**0070**] Identification: secure identification

[**0071**] Location service: a service providing the geographical location of the device

[**0072**] When the returned resource is an XML document, the resource manager **320** parses the XML file and returns the DOM representation of the XML document.

[**0073**] The requester **310** handles the communications between the client and the server. The exchange of data is structured essentially in the form of XML messages, either in a text format or a compiled format. The compiled format is managed by the compiled XML module **312**, and is more

efficient, as the size of the transferred data is reduced. Additionally, it saves the trouble of translating objects into XML on the server and the reverse on the client.

[0074] **FIG. 4** shows a broad description of the three primary deployment profiles. Each of these profiles will be later described in a more detailed fashion, in **FIGS. 5, 6, 7**. An additional specific profile, namely the applet profile, represents an example of a mix of the three primary profiles, and is illustrated in **FIG. 8**:

[0075] Thin Connected Client **404**: On the server **408**, the resource manager retrieves the deployed form from the server storage. The processor on the server **408** retrieves additional resources, if necessary, and processes the form. The processor then sends the processed form to the UI manager on the client device **406**, which generates the interface on the device's display.

[0076] Thick connected client **402**: On the server **408**, the resource manager retrieves the form specification from the server storage and sends the form specification to the processor on the client device **406**. On the client device **406**, the processor retrieves additional resources if necessary, processes the form specification and passes the processed form to the UI manager, which generates the interface on the device's display.

[0077] Stand-alone **400**: On the client device **406**, the resource manager retrieves the form specification from the local storage resource and passes the form specification to the processor. The processor retrieves additional resources if necessary, processes the form specification and passes the processed form to the UI manager, which generates the interface on the device's display.

[0078] In each of these modes, including the stand-alone mode, external data can be incorporated through a connector to an external resource. For example, the client device **406** may query a network database or an XML data server on the web.

[0079] An additional mode, the applet deployment mode, can be thought of as a combination of the thin and the thick connected client profiles. However, in this case, the client runtime is not an application but an applet running inside a web browser supporting Java. With this model, the user can access the same application without installing specific runtime software on the client. The use of an applet constrains the deployment, because standard restrictions for applets apply, such as restricted access to local resources and to servers.

[0080] The process of presenting a form on the client device is now described within an embodiment of an architecture. The process will be illustrated for each of the main deployment profiles described above in **FIG. 4**. In the following figures, thick arrows represent the flow of information between stack layers. Notice that, in some embodiments, the grayed layers in the figures represent layers that are actually available on the computing system, although they are not used in the represented deployment profile. Alternatively, in some embodiments, the grayed layers may be absent.

[0081] **FIG. 5** illustrates the flow of information within the client and server stacks, for the thin connected profile.

[0082] **501** The process starts when the user raises an event on the device interface **10**, associated with an action provoking the presentation of a new form.

[0083] **502** The event manager **14** intercepts the event and sends the request to the requester **21**.

[0084] **503** The requester **21** transmits the request as an XML message to the listener **35** on the server **36**.

[0085] **504** The listener **35** analyzes the request and sends a command to the resource manager **34** to retrieve the XML file **520** containing the form specification. The resource manager **34** then parses the XML document into a DOM tree.

[0086] **505** The parsed DOM tree of the form specification **520** is passed to the processor **30**. The screen builder **32** uses this DOM tree to construct a new DOM tree representing the displayable interactive form. During this processes, the script engine **33** may be invoked in order to execute the construction of script code if any.

[0087] **506** As part of the construction process, the processor **30** may invoke the resource manager **34**, requesting some data from a database **521** to populate the displayable interactive form.

[0088] **507** The processor integrates the data in the form as specified in the form specification **520**.

[0089] **508** The listener **35** gets the DOM tree representing the visual form constructed by the processor **30**, generates the XML representation of the visual form and sends it back to the requester **21**. The requester then uses an XML parser to construct a DOM tree representing the displayable interactive form.

[0090] **509** Alternatively, the listener **35** may return a compiled version of the visual form to the compiled XML module **20** of the client device **22**, which in turn constructs a DOM tree representing the displayable interactive form.

[0091] **510** The DOM tree is passed to the UI Manager **11**, which generates the corresponding User Interface controls.

[0092] **511** The form is displayed on the device's interface.

[0093] **FIG. 6** illustrates the flow of information within the client and server stacks, for the thick connected profile.

[0094] **601** The process starts when the user raises an event on the device interface **10**, associated with an action provoking the presentation of a new form.

[0095] **602** The event manager **14** intercepts the event and sends the request to the requester **21**.

[0096] **603** The requester **21** transmits the request to the listener **35** on the server **36**

[0097] **604** The listener **35** analyzes the request and sends a command to the resource manager **34** to retrieve the XML file **620** containing the form specification.

[0098] **605** The listener **35** gets the XML document representing the form specification **620** and sends it back to requester **21**. The requester then uses an XML parser to construct a DOM tree representing the form specification.

[0099] **606** Alternatively, the listener **35** may return a compiled version of the form specification to the compiled

XML module **20** of the client device **22**, which in turn constructs a DOM tree representing the form specification.

[0100] **607** The parsed DOM tree of the form specification is passed to the processor **15**. The screen builder **17** uses this DOM tree to construct a new DOM tree representing the displayable interactive form. During this processes, the script engine **18** may be invoked in order to execute the construction script code if any.

[0101] **608** As part of the construction process, the processor **15** may invoke the resource manager **19**, requesting some data from a database **621** to populate the displayable interactive form.

[0102] **609** The processor then integrates the data in the form as specified in the form specification **620**.

[0103] **610** The DOM tree is passed to the UI Manager **11**, which generates the corresponding User Interface controls.

[0104] **611** The form is displayed on the device's interface.

[0105] **FIG. 7** illustrates the flow of information within the client stack, for the stand-alone profile.

[0106] **701** The process starts when the user raises an event on the device interface **10**, associated with an action starting the execution of script code.

[0107] **702** The event manager **14** intercepts the event, gets the script code associated with the event, and passes it to the processor **15**. The script engine **18** executes the script code.

[0108] **703** Within the execution of the script code, a new form is utilized. A command is sent to the local resource manager **19** requesting to retrieve the local XML file **720** containing the form specification, which is parsed into a DOM tree.

[0109] **704** The parsed DOM tree of the form specification is passed to the processor **15**. The screen builder **17** uses this DOM tree to construct a new DOM tree representing the displayable interactive form. During this processes, the script engine **18** may be invoked in order to execute the construction script code, if any.

[0110] **705** As part of the construction process, the processor **15** may invoke the resource manager **19**, requesting some data from a database **721** to populate the displayable interactive form.

[0111] **706** The processor integrates the data in the form as specified in the form specification **720**.

[0112] **707** The DOM tree is passed to the UI Manager, which generates the corresponding User Interface controls.

[0113] **708** The form is displayed on the device's interface.

[0114] **FIG. 8** illustrates the flow of information within the client and server stacks, for the applet profile. The client runtime runs within a Java compatible HTML browser **800** and the server **801** is a web server.

[0115] **802** The browser loads the applet client runtime code from the web server **801**. The client runtime then runs within the browser's JVM.

[0116] **803** The user raises an event on the HTML browser **800**, associated with an action starting the execution of script code.

[0117] **804** The event manager **14** intercepts the event, gets the script code associated with the event, and passes it to the processor **15**. The engine **18** executes the script code.

[0118] **805** Within the execution of the script code, a new form is required. A command is sent to the requester **21** requesting to retrieve a form from the server.

[0119] **806** The request is transmitted to the listener **35** within an HTTP request to the web server **801**.

[0120] **807** The listener **35** analyzes the request and sends a command to the resource manager **34** to retrieve the XML file **820** containing the form specification. The resource manager **34** then parses the XML document into a DOM tree.

[0121] **808** The parsed DOM tree of the form specification is passed to the processor **30**. The screen builder **32** uses this DOM tree to construct a new DOM tree representing the displayable interactive form. During this processes, the script engine **33** may be invoked in order to execute the construction script code if any.

[0122] **809** As part of the construction process, the processor **15** may invoke the resource manager **34**, requesting some data from a database **821** to populate the displayable interactive form.

[0123] **810** The processor then integrates the data in the form as specified in the form specification **820**.

[0124] **811** The listener **35** gets the DOM tree representing the visual form constructed by the processor **30**, generates the XML representation of the visual form and sends it back to the requester **21**. The requester then uses an XML parser to construct a DOM tree representing the displayable interactive form.

[0125] **812** The script that started the loading process of the form may perform some additional initializations, further manipulating the DOM tree.

[0126] **813** The DOM tree is passed to the UI Manager **11**, which generates the corresponding User Interface controls.

[0127] **814** The form is displayed on the device's interface.

[0128] It should be noted that the applet profile is just one example of mixing the main profiles illustrated in **FIGS. 5, 6** and **7**. Capability is provided for mixing these profiles in many other ways. That is, a given application running on a given client device may combine the behavior of the three profiles. For example, an application on a PDA device may be developed to run in a thin client mode when connected to the corporate server within an intranet, thus taking advantage of real-time data, as illustrated in **FIG. 5**. Alternatively, when the same application recognizes the unavailability of a valid network connection, it may automatically revert to a stand-alone mode, using a local database that has been automatically synchronized during the connected mode. The same application may then run anywhere, such as at a customer's site.

[0129] By reference to **FIG. 2**, an embodiment of the present invention comprises a visual design module **226**, used by the developer during the visual design step **202**. **FIG. 9** illustrates with more details of one example of a visual design module, in which the visual design module

may comprise a project browser panel **901**, a properties browser panel **902**, and a preview and XML source panel **903**.

[**0130**] The developer uses the building elements available from the visual design module to create the forms. For example, the following building elements may be provided:

[**0131**] Box **904**: A container for other objects.

[**0132**] Bulletinboard: A container for other objects, which may be placed at arbitrary locations.

[**0133**] Button **905**: An action initiator.

[**0134**] Checkbox: A Boolean indicator.

[**0135**] Grid: A container for tabular data. Sub-objects include columns and rows.

[**0136**] Image: A container for a picture.

[**0137**] Menulist: A drop-down selection list of menu items

[**0138**] Spring: A flexible spacing element to be used between other objects.

[**0139**] Text **906**: characters and labels.

[**0140**] Textfield **907**: An entry field.

[**0141**] For: A loop element to integrate data from a data source

[**0142**] A building element may be given specific attributes, such as background color and font size through the properties browser **902**. Such built-in attributes are available for each building element and are presented within the built-in attributes tab **908**. Custom attributes may also be added to an element within the custom tab **909**.

[**0143**] To define the behavior of the form, scripts may be added to the events presented within the event tab **910** of the properties browser. A building element has an associated list of built-in events that may be linked to script code. The script code may define the interaction code, the business logic and the computations required to process the event.

[**0144**] External data may be used to customize the screen during the construction phase of a form; a for building element may be used for this purpose. Examples include populating a list of values in a menulist element or a grid element, with data extracted from a database table. Alternatively, the external data may be accessed by some script code executed during the processing of an event. One example includes checking a username and password against a credentials database table when a user validates a login form.

[**0145**] When writing script code, the developer accesses the data sources and other resources like images as abstract resources, without specifying the physical location and connectivity properties of the resources. The mapping to actual resources is done in a later step. There is no means, during this phase, to state whether the resource will be located on the client device or on the server. This is a feature that allows a given application to be deployed on several different deployment targets.

[**0146**] The visual design module translates the form specification **911** into an XML document that may be viewed and edited within the XML Source tab **912**. The hierarchical structure of XML is very well suited to represent visual components of a user interface. The developer may use the visual features of the visual design module; alternatively, the developer may directly edit the XML document representing the current form.

[**0147**] By reference to **FIG. 2**, an embodiment comprises a deployment module **228**. When the form specifications have been created during the visual design step **202**, the developer proceeds to the define deployment step **206**, interacting with the deployment module **228**. The developer starts defining the platforms on which the application will run. Each platform may be characterized by:

[**0148**] The client device environment (e.g., the Java version, the device's profile).

[**0149**] A list of the form specifications deployed on the platform. Especially, some forms may not be deployed in all the platforms.

[**0150**] The involved servers.

[**0151**] The accessed databases and resources.

[**0152**] Platforms can also be added or modified at any time. When a new platform is added, the designer tool **200** updates the relevant data.

[**0153**] When the platforms have been defined, the developer requests the deployment module **228** to prepare the deployment of the platforms. The deployment module **228** analyzes the form specifications listed in the platforms, especially the scripts contained by the forms, and determines the involved abstract resources. A resource is either:

[**0154**] A form resource: a resource used during the construction process of the form. Since the construction process may be uniformly performed on the server or on the client device, the developer can consider performance issues when deciding where to locate the form resources.

[**0155**] Event resource: a resource used during the processing of an event of the form. Event processing is performed on the client device.

[**0156**] For each abstract resource determined during the preparation process, the developer defines a mapping to an actual resource. Therefore, the developer sets the type of the resource, its location and its properties. For example, if the resource is a database table, the settings may include a reference to a database resource and the name of the table. Some resource definitions may be shared by different elements in one form or across different forms. External resources may be defined only once and used anywhere in the application.

[**0157**] Once the deployment has been defined in the define deployment step **206**, the developer proceeds to the export deployment step **210**, using the deployment module **228** of the designer tool **200**. When the developer requests the deployment module **228** to export the deployment, the designer tool **200** creates all the files required by the deployment and copies the files and the resources either to the client or to the server according to the deployment definition. Alternatively, the developer may export the required files to an intermediary storage, such as a local disk, and later copy them to the final target. On the client **238**, the files to deploy include ready-to-deploy form specifications

216 and resources **218**. On the server **236**, the files to deploy include ready-to-deploy form specifications **220**, resources **222** and the server configuration file **224**. Several files and resources may be deployed, including images, queries or other resource files used by the application.

[**0158**] By reference to **FIG. 2**, an embodiment comprises a test module **244**. After the deployment has been exported, the developer proceeds to test **240** the application. In accordance with one aspect, a process is provided to test applications on different platforms. Testing applications on small devices is a good way for the developer to get the real feeling of the user interface and to check the actual behavior of the application. The defined process promotes an iterative approach for testing. As discussed previously, the design step **202** can be performed only once for all the targeted platforms. To simplify the first testing iterations, the developer can define a test platform **242** on which to deploy the application, with the developer's desktop used as a client **238**. The deployment may include a server **236** accessible from the desktop in order to simulate the real-world environment. The developer then tests the application on this platform **242**. Once the functionalities are tested on the desktop test platform, the developer may proceed to test the application for each real targeted platform **242**. In one embodiment, the designer tool **200** interacts with device emulators to proceed to test the application against specific devices. If the actual client device is connected to the developer's desktop, the designer tool **200** may deploy the required files directly on the device when the deployment function is run.

[**0159**] Embodiments of XML descriptors are now described by reference to **FIG. 10**. The XML syntax used in the different steps of the development process is described. The example illustrating the syntax is a simple window with a menulist containing a list of train stations extracted from a database.

[**0160**] During the visual design phase **1041**, the developer visually defines the form **1011** and the designer tool generates the corresponding XML representation of the visual form specification **1001**. Conversely, the developer may edit the XML representation **1001**, and the designer tool will generate the corresponding form on the screen. In the example presented in **FIG. 10**, the visual components visible on the screen have corresponding elements in the shown descriptor example1.xml, for instance the menulist **1030**. The XML tag <for> **1023** is used to fill the menulist **1030** with items extracted from a database. The tag represents a loop on each item retrieved from the data source, and the value of the attribute datasource="stations"**1021** names the source of the data to be used by this loop. This name is the name of the abstract resource representing the data source, and does not necessarily correspond to a physical database name. A mapping with an actual resource will be defined later. In the <for> declaration, the attribute cursor-name="s"**1024** defines the cursor object used within the loop to access the items retrieved from the database. Within the loop definition, the access to the item's content is performed through the syntax s.field('STATION_NAME') **1022**, giving access to the STATION_NAME field of the current item.

[**0161**] To deploy the form, the developer first defines a deployment platform **1031**, named jdk1.3 in the example, and adds the form **1034** example1 to this platform. The developer: then launches the prepare deployment process on the designer tool. The designer analyzes the forms and finds in the form **1011**, as a consequence of the value of the attribute **1021**, an unknown abstract resource named stations **1035**. This resource is listed as a Form Resource because it is used within a <for> declaration. At runtime, <for> declarations are used during the construction of the form, and not during the processing of an event. The properties of the deployment represented on the properties browser **1012** correspond to a deployment definition contained in the XML project file **1002**. This file contains the list of the form specifications included in the project and the deployment definitions **1040** for the client and the server, and indicates all the resources used in the project.

[**0162**] The developer defines a database by adding an element **1033** under the databases **1032** element in the project browser. The developer defines properties of the added database by providing suitable values for the database component properties on the properties browser **1012**, including the database's type and the settings for accessing the physical database. The corresponding syntax **1036** in the project file **1002** indicates that a database named localXML is located in a directory xmldb and corresponds to an XML file.

[**0163**] In order to bind the stations data-source to the localML database, the developer sets the properties **1012** attached to the stations resource **1035** to the following values:

[**0164**] Type: Table. The resource is a database table

[**0165**] Database Name: localXML. Corresponds to the logical name of the database containing the table

[**0166**] Table Name: train.stations. Corresponds to the physical name of the table in the database.

[**0167**] These settings are represented in the XML project file **1002** as: <prebind name="stations" databasename="localXML" tablename="train.stations" type="table"/>.

[**0168**] The next step is the export deployment step. During this step, the designer tool generates the ready-to-deploy form specification **1003**. The designer tool combines the visual form specification **1001** with the contents of the project file **1002**, thus binding the abstract resources contained in the form specification with the physical resources as specified during the deployment definition **1040**. The binding syntax **1043** for the table is:

<prebind name="stations" action="table" protocol="local" resource-name="train.stations"db="localXML"/>

[**0169**] The prebind tag is used to define a binding for a resource used during the construction of the form and corresponds to a form resource. Conversely, a bind tag is used to define a binding for a resource used during the processing of an event and corresponds to an event resource. The associated properties are:

[**0170**] Name: the logical name of the resource

[**0171**] Action: designs the type of action to be performed by the runtime when processing the bind or prebind elements. In the example, the table value indicates that data must be retrieved from a database table. The possible values for this attribute and the corresponding actions are:

[0172] query: retrieve data by querying a database

[0173] table: retrieve data from a database table

[0174] method: access a java method

[0175] file: retrieve a text file or an XML file

[0176] form: open a new form

[0177] openDatabase: connect to a database

[0178] resource-name: the name of the resource, database table in the example

[0179] db: the logical name of the database

[0180] One major issue affecting the user experience with mobile devices is the latency when moving from form to form within an application. This is due to the connection and transfer time for the forms, and the poor performance of the devices' processor. Embodiments provide additional capabilities, which may dramatically improve the perceptible performance of a mobile application.

[0181] Caching: If a form has been downloaded and processed by the device, the form remains in the device's memory as a processed form, and may be quickly and frequently accessed as required. The client runtime manages the list of loaded forms it can keep in memory, releasing cached forms as needed.

[0182] Pre-fetching: designated forms may be downloaded, processed by the device, and stored into the cache, during the idle time of the device's processor, even before the form has been explicitly requested by the user interaction within the application. Idle time typically occurs during user interaction. The client runtime may later access the form from the cache as required, without apparently incurring any download or processing time.

What is claimed is:

1. A computer program product embodied in a computer-readable medium configurable to accomplish execution of an application that is specified and encoded in a markup-based descriptor language, the product comprising:

client runtime computer code configured to cause a client computer device to process the markup-based descriptor language to deploy an application to accomplish execution of the application,

wherein the client runtime computer code is further configured to process the markup-based descriptor language to selectively configure the client computer device to deploy the application so as to accomplish execution of the application by the client computer device stand-alone or by the client computer device in cooperation with a server to which the device is connectable via a network connection.

2. The product of claim 1, wherein the client runtime computer code includes:

process layer computer code, configured to cause the client device to process the markup-based language to respond to events; and

resource manager layer computer code, configured to cause the client device to access particular resources of the client device.

3. The product of claim 2, wherein the client runtime computer code further includes:

user interface manager layer computer code, configured to cause the client computer device to interact with events communicated to the client computer device via input facilities of the client device and to communicate indications of the events to the process layer computer code.

4. The product of claim 3, wherein the client runtime computer code is configured to selectively configure the client computer device to deploy the application so as to accomplish execution of the application by the client computer device in cooperation with a server to which the device is connectable via a network connection by configuring the client computer device to selectively access at least one of a process layer and a resource manager layer executing on the server.

5. The product of claim 1, wherein the client runtime computer code configured to cause a client computer device to process the markup-based descriptor language includes code to cause the client computer device to process the markup language configuration descriptors to determine the selective deployment.

6. The product of claim 1, wherein the client runtime computer code includes computer code configured to generate visual forms by processing visual form specifications that describe the dynamic and visual behavior of the application, wherein the client computer code configured to accomplish execution of the application accomplishes execution of the application at least in part based on the generated visual forms.

7. The product of claim 5, wherein the code configured to cause the client computer device to generate the visual forms includes code configured to cause the client computer device to process visual form specifications, of the markup-based descriptor language, that include references to at least one of resources, of the client device and resource of the server, employed to generate the visual form.

8. The product of claim 7, wherein the resources include database tables, images, and external executable code.

9. The product of claim 7, wherein the code configured to cause the client computer device to generate the visual forms includes code configured to cause the client computer device to display visual components of the visual forms in a particular manner on a display device of the client computer device;

code configured to cause the client computer device to execute particular actions while generating the visual forms; and

code configured to cause the client computer to execute particular actions in response to events.

10. The product of claim 9, wherein the particular actions in response to events includes changing visual characteristics of a displayed visual form, displaying another visual form, and accessing external resources.

11. A method to define a dynamic form, having associated therewith a visual presentation populated with display data and action triggered by events, for display on a client computing device in a particular network deployment configuration, the particular network deployment configuration including particular data resources, the method comprising:

developing the form in a deployment-independent manner, including abstract data resource connectors, to generate a deployment-independent developed form; and

processing the deployment-independent developed form to generate a deployed form in which the abstract data resource connectors are bound to the particular data resources, the deployed form being such that, when processed in the particular network deployment configuration, the particular data resources of the particular network deployment configuration are accessed to display the visual representation of the form populated with display data.

12. The method of claim 11, wherein:

the step of generating the deployment-independent developed form includes

defining the visual presentation associated with the form; and

defining how the visual presentation of the form is to be populated with the display data.

13. The method of claim 12, wherein:

in the step of generating the deployment-independent developed form, the step of defining the visual presentation associated with the form further includes defining a logical behavior of the form.

14. The method of claim 11, and further comprising:

defining a deployment profile, corresponding to the particular deployment, that identifies a correspondence between the abstract resource connectors and the particular data resources of the particular deployment; and

in the step of processing the deployment-independent developed form, processing the deployment-independent developed form in view of the deployment profile.

15. The method of claim 14, wherein the step of processing the deployment-independent form in view of the deployment profile includes:

generating code configured to cause the client device to display the visual presentation of the form on a display device directly associated with the client;

generating code configured to cause the display data to be accessed from the particular data resources to populate the visual representation of the form.

16. The method of claim 15, wherein the step of processing the deployment-independent form in view of the deployment profile further includes:

generating code configured to cause the client device to handle events according to the defined logical behavior.

17. The method of claim 16, wherein the step of generating code configured to cause the display data to be accessed from the particular data resources for populating the form includes generating code configured to format the accessed display data in a manner suitable to populate the visual representation of the form.

18. The method of claim 16, wherein:

the code configured to cause the client device to display the visual presentation of the form is XML code; and

the code configured to cause the display data to be accessed is XML code to configure the Java servlet code.

19. The method of claim 17, wherein:

the code configured to format the accessed display data is Java code, and the resulting formatted accessed display data is XML code-formatted.

20. The method of claim 12, wherein:

the method is employed to define a plurality of related forms;

the step of defining the visual presentation associated with the form includes defining the visual presentation of each of the related forms; and

the method further comprises defining a relationship among the related forms

the method further comprise defining resource shared by a group of forms.

21. The method of claim 20, wherein:

the plurality of related forms includes a first form and a second form;

the step of setting the relationship among the related forms includes generating code configured to cause the client device to display the visual presentation of a first form on a display device directly associated with the client; and

generating code configured to cause the display data to be accessed from the particular data resources to populate the visual representation of a second form while the client device is displaying the visual representation of the first form.

* * * * *