

(43) International Publication Date
2 April 2009 (02.04.2009)

PCT

(10) International Publication Number
WO 2009/043033 A4

- (51) International Patent Classification: 60/976,325 28 September 2007 (28.09.2007) US
G06F 15/16 (2006.01) 60/976,327 28 September 2007 (28.09.2007) US
60/976,339 28 September 2007 (28.09.2007) US
- (21) International Application Number: PCT/US2008/078169
- (22) International Filing Date: 29 September 2008 (29.09.2008)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
- | | | |
|------------|--------------------------------|----|
| 60/976,266 | 28 September 2007 (28.09.2007) | US |
| 60/976,292 | 28 September 2007 (28.09.2007) | US |
| 60/976,246 | 28 September 2007 (28.09.2007) | US |
| 60/976,275 | 28 September 2007 (28.09.2007) | US |
| 60/976,301 | 28 September 2007 (28.09.2007) | US |
| 60/976,283 | 28 September 2007 (28.09.2007) | US |
| 60/976,311 | 28 September 2007 (28.09.2007) | US |
| 60/976,308 | 28 September 2007 (28.09.2007) | US |
| 60/976,222 | 28 September 2007 (28.09.2007) | US |
| 60/976,312 | 28 September 2007 (28.09.2007) | US |
| 60/976,281 | 28 September 2007 (28.09.2007) | US |
| 60/976,304 | 28 September 2007 (28.09.2007) | US |
| 60/976,258 | 28 September 2007 (28.09.2007) | US |
| 60/976,216 | 28 September 2007 (28.09.2007) | US |
| 60/976,252 | 28 September 2007 (28.09.2007) | US |
| 60/976,269 | 28 September 2007 (28.09.2007) | US |
| 60/976,249 | 28 September 2007 (28.09.2007) | US |
| 60/976,320 | 28 September 2007 (28.09.2007) | US |
- (71) Applicant (for all designated States except US): **XCERION AKTIEBOLAG** [SE/US]; 2003 Western Avenue, Suite 500, Seattle, WA 98121 (US).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): **ARTHURSSON, Daniel** [SE/SE]; 2003 Western Avenue, Suite 500, Seattle, WA 98121 (US). **BRISTAV, Markus** [SE/SE]; 2003 Western Avenue, Suite 500, Seattle, WA 98121 (US).
- (74) Agent: **FEEKES, Clint, J.**; Christensen O'connor Johnson Kindness PLLc, 1420 5th Avenue, Suite 2800, Seattle, WA 98101-2347 (US).
- (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH,

[Continued on next page]

(54) Title: NETWORK OPERATING SYSTEM

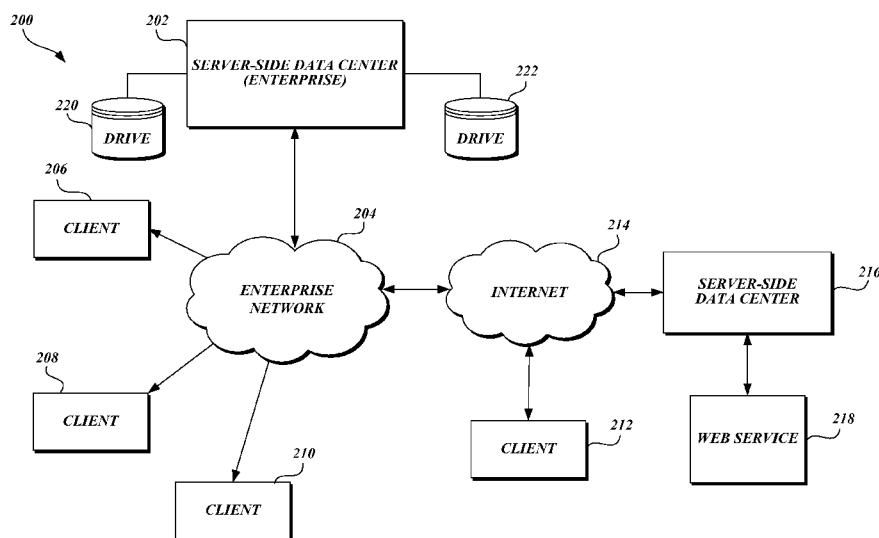


Fig.2.

(57) Abstract: Generally described, the present invention is directed to a network operating system that provides more effective ways of leveraging the connectivity of computer networks. In one embodiment, an XML virtual machine is implemented that accepts high-level application code written in an XML programming language as input. Functionality is provided to interpret or translate the application code written in an XML programming language into code that is suitable for execution across computer platforms. Moreover, the XML virtual machine supports the Model View Controller (MVC) design paradigm that facilitates true data abstraction from applications to a common data model. Multi-instance applications, with each instance potentially supporting multiple views, may be created and executed.



GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(88) Date of publication of the international search report:

23 July 2009

AMENDED CLAIMS

[received by the International Bureau on 29 July 2009 (29.07.2009)]

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. An application platform for executing applications in which functionality is separated into layers in accordance with the MVC design paradigm, the application platform configured to:

provide (1600) a bind operation for binding an object defined in an application's user interface logic to a document in a data model, wherein multiple objects may be bound to the same document;

allowing (2206) the object to utilize rules and interpret content in the document in which the object is bound without coupling the object's functionality to the structure of data in the document; and

maintaining (2100) a rule handler in the data model that defines how each rule-using object bound to the document will interpret their data binding.

2. The application platform as recited in Claim 1, further comprising:

allowing a listening object to register as a listener for data updates performed on the document; and

in response to the document experiencing a data update, notifying and supplying the listening object with data for determining whether the listening object is affected by the data update.

3. The application platform as recited in Claim 1, wherein maintaining a rule handler in the data model that defines how each rule-using object bound to the document will interpret their data binding includes causing the semantic logic of an object's rules to be included in the data model within a namespace that is separate from a namespace associated with a different object.

4. The application platform as recited in Claim 1, wherein binding an object defined in an application's user interface logic to a document in the data model, includes:

identifying a semantic description of a bind operation encountered in application code, the semantic description having an expression referencing a document in the data model that the runtime object will be bound; and

evaluating the expression relative to an application instance.

5. The application platform as recited in Claim 4, wherein binding an object defined in an application's user interface logic includes registering the object as a data update listener on the document that is notified when a data update affecting the object's data binding occurs.

6. The application platform as recited in Claim 1, wherein binding an object defined in an applications user interface logic to a document, includes:

determining whether the object utilizes rules;

if the object utilizes rules:

causing the rule handler to apply the rules of the object relative to the current version of the data model;

merging the rules of the object into the data model, the rules being merged into a namespace that is separate from a different object.

7. The application platform as recited in Claim 1, wherein maintaining a rule handler in the data model, includes:

receiving a data update from the object that modifies the rules maintained in the data model;

notifying one or more existing rule-using objects bound to the document of the data update; and

executing the rule handler relative to the data model available to the one or more existing rule-using objects.

8. A data update bus configured to perform a method that propagates a data update between a data model and one or more objects, comprising:

instantiating (1254) an abstract representation of a document (e.g., "document object");

registering (1612) one or more objects as data update listeners on the document object;

receiving (2002-2004) a call to perform a data update, the call including a fragment describing a relative change to the data model;

for each object registered as a data update listener:

determining (2214) whether a data binding associated with an object is affected by the data update; and

if a data binding is affected by the data update, setting (2008) new data from the data model on the object.

9. The data update bus as recited in Claim 8, wherein registering one or more objects as data update listeners includes creating a translated XBind, the translated XBind referencing data in the data model that the one or more objects are bound.

10. The data update bus as recited in Claim 8, wherein the object is a user interface component and registering the object as a data update listener, includes:

causing a rule handler to apply the rules of the user interface component relative to the current version of the data model; and

merging rules associated with the user interface component into the data model with rules associated with a different object.

11. The data update bus as recited in Claim 8, wherein the fragment encapsulates the data for implementing the data update in the data model serialized in XML.

12. The data update bus as recited in Claim 8, wherein the fragment is propagated to remote listeners utilizing a channel that abstracts communications between a local computing device and a network service.

13. The data update bus as recited in Claim 8, wherein determining whether a data binding associated with the object is affected by the data update includes comparing an XBind associated with the object with an XBind that references a location in the data model affected by the data update.

14. The data update bus as recited in Claim 8, wherein if the object utilizes rules, setting data on the object, includes:

causing a rule handler to execute the rules associated with the document; and
providing the object with current namespace information.

15. A computing device configured to set rules of an object to interpret content in a data model, comprising:

a processor (300);

a memory (304) having computer-executable instructions which, when executed by the processor, causes the computing device to:

add (1616) a rule associated with a new object to a rule handler maintained in a data model;

cause (2108) the rule handler to apply the rules of the new object relative to the current version of the data model; and

merge (2110) the rules of the new object into the data model, the rules being merged into a namespace that is separate from namespaces allocated to the rules associated with one or more existing objects that are bound to a document in the data model.

16. The computing device as recited in Claim 15, further configured to notify the one or more existing objects bound to the document of the data update to the rules maintained in the data model.

17. The computing device as recited in Claim 15, wherein to add a rule associated with a new object to a rule handler includes registering the new object as a data update listener that is notified when the document experience a data update.

18. The computing device as recited in Claim 15, wherein to cause the rule handler to apply the rules of the new object relative to the current version of the data model includes executing the rules of the one or more existing objects that are bound to the document.

19. The computing device as recited in Claim 15, wherein to merge the rules of the new object into the rule handler includes providing a reference to the new object that identifies an XML namespace allocated to the object's rules.

20. The computing device as recited in Claim 15, wherein the new object is a user interface component defined in accordance with an Application Programming

Interface that does not have developers provide scripting logic to define the component's functionality.

21. A method implemented in instructions executed by a computing device for evaluating an expression that references an atomic component, the method comprising:

- determining (1804) whether a semantic expression in an application requests data associated with an atomic component that is not associated with a defined data binding;
- identifying (1806) a runtime object that holds data relating to the current state of the atomic component, the state of the runtime object configured to change during execution of the application;
- requesting and receiving (1808) data referenced in the expression from the runtime object that corresponds to the atomic component; and
- propagating (1810) the data referenced in the expression to a system provided document maintained in a data model.

22. The method as recited in Claim 21, wherein the expression utilizes a name that is global to the application to reference the atomic component and wherein identifying a runtime object that holds data relating to the current state of the atomic component includes identifying an application instance which caused the atomic component to be instantiated.

23. The method as recited in Claim 21, wherein determining whether a semantic expression in an application requests data associated with an atomic component includes configuring each runtime object associated with an application to be queried for an XBind that defines the runtime objects data binding.

24. The method as recited in Claim 23, wherein the XBind is a multi-dimensional data type that includes at least a URL, base path, and optional selection.

25. The method as recited in Claim 21, wherein requesting and receiving data referenced in the expression from the runtime object that corresponds to the atomic component includes serializing data maintained by the runtime object into XML.

26. The method as recited in Claim 21, wherein each expression defined in the application that is evaluated complies with XML syntax, and wherein propagating the data referenced in the expression to a system provided state document includes creating a translated expression that points to a location in the system provided state document where the data referenced in the expression is accessible.

27. A system for hosting a network file system, the system comprising:
an XML network service (2510) for two-way communication with a client computing device;
a file server (2516) for storing raw data files;
an index server (2518) for storing full-text indexes of the raw data files and file metadata; and
a folder database (2520) for storing folder hierarchy information.

28. The system as recited in Claim 27, further comprising a transaction coordinator for facilitating concurrent use of a file maintained in the network file system by multiple client computing devices when the multiple client computing devices each register as a listener to an object cached by the transaction coordinator.

29. The system as recited in Claim 28, further comprising a message server for providing the client computing device a notification concerning updates affecting the object cached by the transaction coordinator.

30. The system as recited in Claim 27, further comprising a communication channel instantiated on the client computing device that provides functions to an application executing on the client computing device in performing file system operations exposed by the network service that include one or more of the group consisting of loading, opening, and saving a file.

31. The system as recited in Claim 27, further comprising a communication channel instantiated on the client computing device configured to communicate an Action URL to the network service, the Action URL identifying the type of processing that the network service will perform.

32. The system as recited in Claim 27, wherein the network service is further configured to:

accept a request from an application executing on the client computing device to perform a search for a file associated with specified metadata;

cause one or more files having attributes that match the metadata to be identified from an index of file metadata created by the index server, the list of files being returned from the index server in the format of a network feed; and

return the list to the application executing on the client computing device.

33. The system as recited in Claim 27, wherein the folder hierarchy information maintained in the folder database further includes permissions associated with folders in the file system, the permissions used to determine whether a user is a group member that has rights to access a group folder.

34. The system as recited in Claim 27, wherein the network service is further configured to construct a document describing a folder tree, comprising:

obtaining folder metadata associated with a root folder from the folder database;

obtaining metadata describing each file in the root folder from the index server; and

inserting the folder and file metadata corresponding to the root folder into the document.

35. The system as recited in Claim 34, further comprising:

for each subfolder having the same permissions with respect a user as the root folder;

obtaining folder metadata associated with the subfolder from the folder database;

obtaining metadata describing each file in the subfolder folder from the index server;

inserting the folder and file metadata corresponding to the subfolder into the document.

36. The system as recited in Claim 35, further comprising:

for each subfolder having different permissions with respect to the user as the root folder;

inserting an XLink corresponding to the subfolder into the document, the XLink allowing a user to access the subfolder once the user's right to access the subfolder are authenticated.

37. A system for providing software as a service to a client computing device, comprising:

a network service (2510) providing access to a network file system in which a users files are available as a virtual drive on the client computing device;

at least one application (2402) that provides application logic in semantic application code written according to the one or more XML schemas;

an XML virtual machine (510) delivered as a service from the network service (2510) configured to interpret semantic application code written according to one or more XML schemas, the semantic application code being translated into platform independent code on the client computing device;

wherein the XML virtual machine (510) is configured to cause the logic of the at least one application to be executed without being installed as an executable on the client computing device.

38. The system as recited in Claim 37, wherein the data and application logic of the at least one application executed by the XML virtual machine is separated into a model, view, and controller.

39. The system as recited in Claim 37, wherein the at least one application that describes application logic according to the one or more XML schemas is configured to provide a common desktop environment in which common applications and user data are accessible across different computing devices.

40. The system as recited in Claim 37, wherein application user interface logic interpreted by the XML virtual machine adheres to a first XML schema and wherein application process logic adheres to a second XML schema.

41. The system as recited in Claim 37, wherein the network service is further configured to facilitate current use of a file by multiple users with an update to the file being propagated to each user in real time.

42. The system as recited in Claim 37, wherein the network service is further configured to:

authenticate that a user is a member of a group that has rights to access a root folder in the network file system;

generate and transmit a document describing the contents of the root folder to the client computing device.

43. The system as recited in Claim 42, wherein to generate and transmit a document describing the contents of the root folder to the client computing device, includes:

(a) obtaining folder metadata associated with the root folder from a folder database;

(b) obtaining metadata describing each file in the root folder from an index server; and

(c) repeating (a) and (b) for each folder having the same permissions with respect to the user as the root folder.

44. A method implemented in computer-executable instructions that, when executed by a computer processor, produces a document for transmission to a client computing device that describes a folder tree, the method comprising:

in response to receiving a request from the client computing device to access a root folder, querying (2814) a folder database for information associated with the root folder including an access control list describing a users access rights with regard to the root folder;

authenticating (3116) that the user has rights to access the root folder;

adding (2816) metadata associated with the root folder to the document that describes the folder tree;

for each subfolder within the hierarchy of the root folder that has the same permissions with respect to the user as the root folder:

obtaining (2818) folder metadata of the subfolder from the folder database;

adding (2816) metadata associated with the subfolder to the document that describes the folder tree.

45. The method as recited in Claim 44, further comprising:
querying an index server for information describing each file in the root folder;
adding metadata associated with each file in the root folder to the document that describes the folder tree; and

wherein at least one file in the root folder is of a file type that initiates execution of an application upon the root folder being mounted as a virtual drive on the client computing device.

46. The method as recited in Claim 44, further comprising:
accepting a request from a client computing device to perform a search for a file associated with specified content;

identifying one or more files from an index having content that matches data provided in the request.

47. A message server configured to provide client computing devices notifications concerning a subscribed object, the message server comprising:

a processor (300); and

a computer-readable media (304) having computer-executable instructions stored thereon that, when executed by the processor, cause the message server to perform a method of providing the client computing devices a notification regarding the subscribed object, the method comprising:

receiving (3144) a notification that the subscribed object has been updated;

locating (3150) a message chain associated with the subscribed object;

traversing (2704) links in the message chain; and

sending (3156) a notification to each client computing device identified in traversing the message chain that registered as a listener on the subscribed object.

48. The message server as recited in Claim 47, wherein the computer-readable media has at least one subscription list associated with a client computing device, the subscription list identifying each subscribed object that the client computing device registered as a listener.

49. The message server as recited in Claim 48, wherein subscribed objects that the client computing device registered as a listener is one or more selected from the group

of the online status of a user, a change to a shared file, and a modification affecting a group folder.

50. The message server as recited in Claim 47, wherein a message chain associated with the subscribed object links entries in the subscription lists associated with different client computing devices.

51. The message server as recited in Claim 47, wherein a transaction coordinator in communication with the message server causes the message server to send a notification to multiple client computing devices indicating that a file cached by the transaction coordinator is actively shared by multiple users.

52. The message server as recited in Claim 47, wherein a subscribed object represents a file that stores communications being exchanged in a chat session, the message server configured to send chat communications when the file storing data exchanged during the chat session experiences a data update.

53. A method implemented in computer-executable instructions that, when executed by a computer processor, initiates a chat session between two or more computing devices, the method comprising:

- creating a file (3000) on a network file system;

- for each user participating in the chat session:

- registering (3034) the user as a listener for data updates to the file at the network file system;

- caching (1116) a version of the file on a local client computing device;

- binding (1600) one or more user interface components to display chat communications to the version of the file cached locally; and

- in response to a chat communication being input on the local client computing device, updating (2008) the binding between the user interface component and the version of the file cached locally to display the chat communication, and propagating (2010) a transaction containing the chat communication to a network service.

54. The method as recited in Claim 53, further comprising:

- receiving the transaction at a network service;

determining that the file is actively shared by multiple client computing devices;
and

transmitting the chat communication to each of the client computing that registered as a listener for data updates to the file.

55. The method as recited in Claim 54, wherein determining that the file is actively shared includes causing a message server to perform a lookup of an object in a subscription list, the object linking multiple client computing devices in a message chain.

56. The method as recited in Claim 53, wherein registering the user as a listener for data updates to the file at the network file system includes maintaining a special folder in the network file system, the special folder being monitored by a server process that abstracts the initiation of a chat session for an application executing on the client computing device.

57. The method as recited in Claim 53, wherein creating a file on a network file system, includes:

retrieving a raw file data from a file server; and

instantiating and caching an object on a transaction coordinator that provides a runtime representation of the file.

58. The method as recited in Claim 53, wherein registering the user as a listener for data updates to the file at the network file system, includes:

instructing a message server to add the object to a subscription list associated with a client computing device; and

if multiple client computing devices are registered as listeners, linking entries of the object between multiple subscription lists.

59. The method as recited in Claim 53, wherein caching a version of the file on a local client computing device includes using a communication channel to load the file in a client-side cache.

60. The method as recited in Claim 53, wherein application code is executed in response to a chat communication being input on the local client computing device configured to:

propagate text input into a user interface component to a file cached locally; and
cause the text to be transmitted to remote devices using a network service.

61. A chat application implemented in instructions executed by a computer processor for performing real-time communications with a remote computing device, comprising:

a data model (2500) for storing data communicated during a chat session;

a text display component (3702) bound to the data model;

a text entry component (3704) for accepting a chat communication input by a user during a chat session; and

a send component (3706) that causes application code to be executed in response to the user generating an activation event to send a chat communication, the application code configured to:

propagate (2508) the chat communication to a file in the data model that the text display component is bound; and

cause (2510) the chat communication to be transmitted to the remote computing devices using a network service.

62. The chat application as recited in Claim 61, further comprising a communication channel, configured to:

receive a chat communication from the network service generated by a remote computing device; and

propagate the received chat communication to the file in the data model that the text display component is bound.

63. The chat application as recited in Claim 61, wherein to bind the text display component to the data model, includes:

instantiating document object; and

registering the text display component as a data update listener on the document object, wherein the document object notifies the text display component when a chat communication is propagated to the data model.

64. The chat application as recited in Claim 63, wherein the text display component provides at least one rule for interpreting how a chat communication in the data model is visually displayed.

65. The chat application as recited in Claim 61, wherein the text entry component is an atomic component that is not associated with a data binding, and wherein to propagate the chat communication to a file in the data model includes accessing a runtime object associated with the text entry component to obtain the text of a chat communication.

66. The chat application as recited in Claim 61, further comprising a communication channel configured to access operations of a network file system to load a file that is actively shared by multiple computing devices.

67. A client computing device configured to boot a network operating system, comprising:

a processor (300);

a computer-readable media (304) having computer-executable instructions stored thereon that, when executed by the processor (300), causes the computing device to perform a method of booting a network operating system, comprising:

initiating (3204) a process virtual machine configured to execute one or more applications on the client computing device;

instantiating (3210) a communication channel for managing communications with a data source and using the communication channel to obtain a user-specific startup sequence; and

implementing (3214) the startup sequence, wherein implementing the startup sequence includes causing at least one application to be obtained from the data source and executed by the process virtual machine.

68. The client computing device as recited in Claim 67, wherein the process virtual machine is an XML virtual machine that is configured to execute applications described semantically in one or more XML programming languages, and wherein initiating the XML virtual machine includes obtaining the XML virtual machine from a server-side data center.

69. The client computing device as recited in Claim 67, wherein the process virtual machine is embedded in a microprocessor within the client computing device.

70. The client computing device as recited in Claim 67, wherein the specific communication channel that is instantiated is identified based on a location URL navigated to using a host Web browser program.

71. The client computing device as recited in Claim 67, wherein the specific communication channel that is instantiated is identified based on one configuration parameter associated with the client computing device selected from the group of a HTTP query parameter, HTTP cookie data, IP address, geographic location, and machine type.

72. The client computing device as recited in Claim 67, wherein the communication channel is configured to manage communications with a data source that is local to the client computing device and the user-specific startup sequence is implemented without using any network services.

73. The client computing device as recited in Claim 67, wherein implementing the startup sequence causes a folder from a first network file system and a folder from a second network file system to each be mounted as a virtual drive on the client computing device using separate communication channels.

74. The client computing device as recited in Claim 67, wherein the startup sequence is described semantically in a configurable XML document that identifies one or more applications to launch after a successful login to a server-side data center.

75. The client computing device as recited in Claim 67, wherein a login application is associated with the communication channel that is launched to obtain user credentials for mounting a group folder.

76. A system for connecting a client computing device to multiple data sources, the system comprising:

- a first data source (2500) associated with a first communication channel (2504) for delivering a process virtual machine to the client computing device;

- a second data source (2420) associated with a second communication channel (2414);

- a client computing device configured to:

- instantiate the first communication channel (2504) to mount a virtual drive representing the first data source on the client computing device; and

- initiate (1000) execution of at least one application on the process virtual machine, the application configured to cause the second communication channel (2414) to be instantiated that mounts a virtual drive representing the second data source (2420) on the client computing device.

77. The system as recited in Claim 76, wherein the first data source is a network file system accessible from the Internet and the second data source is file system provided from a private network.

78. The system as recited in Claim 76, wherein a data source that is mounted as a virtual drive is one selected from the group of a local storage device, a network file system, and a shared group folder.

79. The system as recited in Claim 76, wherein a data source mounted as a virtual drive on the client computing device is a folder tree from a network file system, and wherein mounting the folder tree includes creating a document for transmission to the client computing device, comprising:

- querying a folder database for information associated with the root folder of the folder tree;

- authenticating that a user has rights to access the root folder; and

- adding metadata associated with the root folder to the document.

80. The system as recited in Claim 79, further comprising:

- for each subfolder within the hierarchy of the root folder that has the same permissions with respect to the user as the root folder;

obtaining folder metadata of the subfolder; and
adding metadata associated with the subfolder to the document that describes the folder tree.

81. The system as recited in Claim 76, wherein at least one file in the root folder is of a file type that initiates execution of an application upon the root folder being mounted as a virtual drive on the client computing device.

82. The system as recited in Claim 76, wherein the first and second communication channels are created in accordance with a communicator Application Programming Interface that accepts logic for accessing data from a data source and translating the data into an XML format expected by an executing application.

83. The system as recited in Claim 76, wherein applications executed by the process virtual machine accesses a model document to implement application functionality without interacting directly with a data source that is managed by a communication channel.

84. A method implemented in computer-executable instructions that, when executed by a computer processor, allows a communication channel to be created in accordance with a Application Programming Interface, the method comprising:

accepting one or more API calls directed at creating a communication channel (2504), the one or more API calls including call parameters that define logic for at least obtaining data from a data source (2500) and creating a model document (2502);

instantiating a communication channel (2504) that implements the logic defined in the one or more API calls; and

obtaining data from the data source (2500) and creating a model document (2502), wherein the application accesses the model document (2502) in implementing application functionality without interfacing directly with the data source that is managed by the communication channel (2504).

85. The method as recited in Claim 84, wherein the logic provided in accordance with the APIs defines standardized operations for loading, reading, and editing data obtained from the data source.

86. The method as recited in Claim 84, wherein obtaining data from the data source and creating a model document includes mounting a virtual drive on a client computing device, and wherein the data source is one selected from the group of a local storage device, a network file system, and a shared group folder.

87. A method of using an Action URL to allow a first application to access functionality of a second application, the method comprising:

instantiating (3210) a communication channel for managing message passing between the first and second applications;

causing (2806) an Action URL to be passed from the first application to the communication channel, wherein the format of the Action URL includes a first portion for identifying the communication channel to handle the Action URL and a second portion that uses a document path of the URL standard to provide an action descriptor; and

invoking (3000) functionality of the second application as specified in the action descriptor portion of the Action URL.

88. The method as recited in Claim 87, wherein invoking functionality of the second application, includes:

translating the action descriptor portion of the Action URL into a request expected by a network service; and

transmitting the request to the network service.

89. The method as recited in Claim 87, wherein the action descriptor portion of the Action URL is in a format that includes a function name that identifies the functionality to invoke and one or more parameters that are passed to the second application.

90. The method as recited in Claim 87, wherein the first application is configured to access an XML document in a data model when implementing application functionality, the method further configured to:

obtain data generated by the second application from a network service; and
translate the data into an XML format for access by the first application.

91. The method as recited in Claim 87, wherein invoking functionality of the second application, includes:

identifying a data type of a document specified in a parameter of the Action URL;
and

performing actions that depend on the identified data type.

92. The method as recited in Claim 91, wherein the document provides the process logic of the application and wherein the action performed that depends on the identified data type is to launch the second application.

93. The method as recited in Claim 87, wherein the Action URL accesses communicator-specific operations apart from standardized operations that are defined in communicator APIs.

94. The method as recited in Claim 87, wherein the Action URL invokes application functionality for processing data in a data model on a local computing device that is outside an XML virtual machine environment.

95. The method as recited in Claim 87, wherein an XLink is inserted in an XML document that describes logic of the first application and wherein the Action URL is passed from the first application to the communication channel when the XLink is activated.

96. The method as recited in Claim 95, wherein logic referenced in the XLink is not currently stored on a local computing device and wherein invoking functionality of the second application includes loading a document from a network service using the communication channel.

97. The method as recited in Claim 87, wherein functionality of the second application is invoked on demand when a user activates an XLink embedded in an HTML document.

98. The method as recited in Claim 97, wherein the XLink is associated with an invitation to become a group member and wherein activation of the XLink, includes:
transmitting the Action URL to a network file system;

authorizing a user specified in the Action URL to access a group folder in the network file system; and

mounting the group folder as a virtual drive on a client computing device associated with the user.

99. The method as recited in Claim 87, wherein passing of the Action URL from the first application to the communication channel occurs asynchronously from functionality of the second application being invoked.

100. A system having computer executable instructions, that, when executed by a computer processor, links functionality provided by different resources, the system comprising:

at least one XLink (2826) defining an explicit relationship between resources, wherein resources referenced in the XLink (2826) may be local or reside on separate computing devices;

an action URL having a first portion that identifies a communication protocol (2414-2418) for transmitting data between resources and a second portion that uses a document path of the URL standard to provide an action descriptor; and

wherein activation of the XLink (2826) causes the Action URL to be communicated from a first resource to a second resource using the communication protocol (2414-2418) identified in the Action URL.

101. The system as recited in Claim 100, further comprising a data type recognizer configured to identify a data type of a resource referenced in the Action URL and initiate actions to perform processing that depends on the identified data type.

102. The system as recited in Claim 101, wherein the action performed that depends on the identified data type is one selected from the group of launching an application and rendering at least one user interface component.

103. The system as recited in Claim 100, where the first and second resources are one selected from the group of a file, image, document, application, and query result.

104. The system as recited in Claim 100, further comprising a communication channel for managing communications between the first and second resources, and wherein activation of the XLink initiates the passing of the Action URL from an application being executed by an XML virtual machine to the communication channel.

105. The system as recited in Claim 104, wherein the communication channel is further configured to:

obtain data generated by the second resource from a network service; and

translate content obtained from the network service into an XML format for access by the application being executed by the XML virtual machine.

106. The system as recited in Claim 100, wherein the first portion of the Action URL is associated with a communication channel and wherein the action descriptor includes a function name and one or more parameters.

107. A method implemented in computer-executable instructions that, when executed by a computer processor, models a selection state of a user interface component, the method comprising:

defining (616) a binding between the user interface component and a document in a data model, wherein the binding causes one or more data items in the document to be displayed on the user interface component;

wherein the user interface component is configured to:

allow (2300) the one or more data items displayed on the user interface component to be selected; and

in response to a selection of the one or more data items, generate (1502) a variable that represents a selection state of the user interface component, the variable having at least three dimensions including a first dimension referencing the document in the data model, a second dimension limiting the selection to an ancestor node in the document, and a third dimension identifying one or more descendent nodes from the ancestor node that stores the selected one or more data items.

108. The method as recited in Claim 107, wherein defining a binding between the user interface component and a document in a data model includes propagating one or more rules associated with the user interface component to the data model, the one or

more rules providing logic that allows a generic component to interpret content in the data model.

109. The method as recited in Claim 107, wherein the variable is an XBind having a URL in a first dimension, a base path expression in a second dimension, and a plurality of base path expressions in a third dimension.

110. The method as recited in Claim 107, wherein:

if one data item is selected, a second dimension of the variable references a root node of an XML document and a third dimension provides a path to a descendent node of the root node that stores the selected data item; and

if a plurality of data items are selected, the second dimension of the variable provides a path to a parent node of the one or more descendent nodes and the third dimension includes an XPath expression for identifying particular data items in the descendant nodes.

111. The method as recited in Claim 110, further comprising evaluating an XPath expression into text.

112. The method as recited in Claim 107, wherein the user interface component displays a folder tree from a network file system and wherein the variable identifies a folder in the network file system that was selected by a user.

113. A computer-readable media having computer-executable modules that provide a standard way of representing a state of a user interface component, comprising:

a data model (710) for storing structured data;

at least one user interface component configured to:

cause (2200) one or more data items from the data model to be displayed on a visual rendering of the user interface component;

allow (2300) the one or more data items to be selected; and

generate (1502) a variable that represents a selection state of the user interface component; and

a state manager configured to cause the selection state of the user interface component as represented in the variable to be propagated to the data model (710).

114. The computer-readable medium as recited in Claim 113, wherein an application defines a binding between the user interface component and an XML document in the data model, wherein the one or more data items displayed on the user interface component are identified by the application in an XPath expression.

115. The computer-readable medium as recited in Claim 113, wherein storing structured data in the data model includes providing a rule handler the rules of multiple runtime objects bound to the same document.

116. The computer-readable medium as recited in Claim 113, wherein the variable generated by the user interface component has at least three dimensions including a first dimension referencing the document in the data model that stores the one or more data items, a second dimension that limits a selection to an ancestor node in the document, and a third dimension identifying one or more descendent nodes from the ancestor node that stores the selected one or more data items.

117. The computer-readable medium as recited in Claim 113, wherein the variable generated by the user interface component is an XBind having a URL as the first dimension, a base path expression in the second dimension, and a plurality of base path expressions in the third dimension.

118. The computer-readable medium as recited in Claim 113, wherein the state manager is further configured to facilitate synchronization of the application's state across multiple client computing devices, comprising:

for each user interface component associated with an executing application:

propagating the selection state of the user interface component as represented in a common variable format (e.g., XBind) to the data model; and

communicating the selection state of the user interface component, serialized in XML, over a communication channel that abstracts communications with a network service.

119. The computer-readable medium as recited in Claim 113, wherein the state manager is further configured to facilitate restoration of an applications state between sessions, comprising:

for each user interface component associated with an executing application, propagating the state of the user interface component to a state document in the data model;

in a first session, saving the state document to a network file system; and

in a second session, retrieving the state document from the network file system and causing each user interface component associated with the application to restore their selection state,

120. The computer-readable medium as recited in Claim 113, wherein the user interface component is an atomic component that is not associated with a defined data binding, and wherein to generate a variable that represents the selection state of the user interface component, includes:

accessing text input into the user interface component from a runtime object;

causing the text to be inserted into a state document in the data model; and

creating an XBind that points to a location in the state document where the text is stored.

121. The computer-readable medium as recited in Claim 113, wherein the one or more data items from the data model displayed on a visual rendering of the user interface component describes a folder tree of a network file system.

122. A computer-readable medium configured to restore a state of an application from a previous network session, comprising:

a server side data center (502) for storing user data including data describing the state of an application;

at least one client computing device (300) comprising:

a processor (300);

a memory (304) having computer-executable modules for saving and restoring the state of an application, including:

a data update bus (710) configured to set one or more data items from a data model for display by user interface components associated with the application, wherein the one or more data items displayed by the user interface components may be selected by a user;

a state manager configured to propagate (2202) a state of each user interface component associated with the application to a state document and cause the state document to be saved at the server side data center (502) in a first network session;

wherein, in a second network session, the computing device is further configured to:

cause (1600) the one or more data items from the data model to be bound and displayed by each of the user interface components; and

obtain (1116) the state document from the server side data center (502) and restore the state of the user interface components that existed in the first network session.

123. The system as recited in Claim 122, wherein a selection state of each user interface component associated with the application is described in a common variable format (e.g., XBind).

124. The system as recited in Claim 122, wherein to restore the state of the user interface components includes causing a user interface component to be rendered at the coordinate positions on a screen display that were occupied during the first network session.

125. The system as recited in Claim 122, wherein to cause the state document to be obtained from the server side data center includes using a communication channel that abstracts communication with a particular network service to load the state document in a client-side cache.

126. The system as recited in Claim 122, wherein to propagate a state of each user interface component associated with the application to a state document, includes:

determining whether a user interface component is associated with a data binding; and

if the user interface component is an atomic component and not associated with a data binding, accessing data input into the user interface component from a runtime object and propagating the data input into the user interface component to the state document.

127. A method implemented in computer-executable instructions that, when executed by a computer processor, satisfies a request by a second client computing device to open a file when the file is open by a first client computing device, the method comprising:

- on a server computing device, retrieving (3124) and caching the raw file data in memory;

- transmitting (3128) the raw file data to the second client computing device;

- notifying (3150) the first client computing device that the file is shared;

- obtaining and committing (3152-3154) any previously uncommitted transactions from the first client computing device; and

- notifying (3156) the second client computing device that the file is shared and propagating any committed transactions obtained from the first client computing device to the second client computing device.

128. The method as recited in Claim 127, further comprising interpreting the transactions propagated to the second client computing device and update the version of the file stored locally.

129. The method as recited in Claim 127, wherein the raw file data is retrieved from a file server and cached in memory by a transaction coordinator, the transaction coordinator configured to instruct a message server to add the file to subscription lists associated with both the first and second client computing devices.

130. The method as recited in Claim 127, wherein propagating any previously uncommitted transactions between the first and second client computing devices, includes:

- receiving a transaction at a network service;

- determining that the file is shared by multiple client computing devices; and

- causing a message server to send the transaction to each client computing devices that is registered as a listener on the file.

131. The method as recited in Claim 127, wherein data updates that occur when the file is open by multiple users as represented in transactions are sent to the server-side

and committed to the raw file cached in memory without immediately being committed to the file maintained in storage on a file server.

132. The method as recited in Claim 131, wherein a transaction affecting a file that is open by multiple users is committed to storage on a file server in response to one of a user generating a command to save the file, each client computing device closing the file, and after a predetermined time period elapsing from creation of the transaction.

133. The method as recited in Claim 127, wherein a user is authenticated as having rights to access the file before the raw file is transmitted to a computing device associated with the user.

134. The method as recited in Claim 127, wherein a transaction manager on the second client computing device is configured to determine whether any conflicts exist between transactions committed to the file by the first client computing device and transactions generated locally when the second client computing device was offline.

135. A computer readable medium having computer executable instructions, that, when executed by a computer processor, performs a method to create a file in a group folder, the method comprising:

receiving (3136) a subscription request from a first client computing device to listen for changes to the group folder;

receiving (3010) a file creation request from a second client computing device to create a new file in the group folder;

allocating (3020) space for raw file data in a storage location;

instructing (3022) a folder database to create an entry for the new file and associate the new file with the group folder; and

causing (3030) a message to be transmitted to the first client computing device, the message causing a view of the group folder on the first client computing device to be updated to reflect the creation of the new file by the second client computing device.

136. The computer-readable medium as recited in Claim 135, wherein the subscription request is generated on the first client computing device by a communication

channel that abstracts the generation and transmission of requests to listen for changes to the group folder.

137. The computer-readable medium as recited in Claim 135, wherein a communication channel causes the group folder to be mounted on the first and second client computing devices as a virtual drive.

138. The computer-readable medium as recited in Claim 135, wherein an application executing on a client computing device requests the file using a URL, and wherein a communication channel instantiated by the application abstracts loading of the file from a network file system.

139. The computer-readable medium as recited in Claim 135, wherein creation of the file abstracts the initiation of a real-time communication session between the first and second client computing devices.

140. The computer-readable medium as recited in Claim 135, wherein an application executing on the second client computing device causes the file to be created to initiate a chat session between group members.

141. The computer-readable medium as recited in Claim 135, wherein the file creation is received by a network service that authenticates a users right to create the file in the group folder, comprising:

querying a folder database and obtaining an access control list associated with the group folder; and

comparing the access control list with the permissions associated with the user.

142. The computer-readable medium as recited in Claim 135, wherein the entry for the file in the folder database associates a file identifier with multiple storage locations that each store raw file data.

143. The computer-readable medium as recited in Claim 135, wherein to associate the new file with the group folder includes propagating metadata associated with the file to an index server configured to perform full text indexing of file data.

144. The computer-readable medium as recited in Claim 135, wherein receiving a subscription request from a first client computing device to listen for changes in a group folder includes adding an object that represents the group folder to a subscription list associated with the first client computing device, and wherein causing a message to be transmitted to the first client computing device includes locating a message chain to identify which client computing devices registered as a listener for updates to the group folder.

145. A system allowing multiple users to receive updates affecting a group folder in a network file system, comprising:

- a storage component (2514) configured to manage storage of raw file data and folders in the network file system;

- a message server (2512) operative to:

- receive (3136) subscription requests to listen for modifications to a group folder that is shared by multiple group members;

- manage (2704) subscription lists associated with client computing devices of group members that are listening for modifications to the group folder;

- a transaction coordinator (2522) that facilitates access to the group folder by the client computing devices, the transaction coordinator operative to:

- retrieve (3122) folder information from the storage component (2514); and

- in response to a modification affecting the group folder, causing (3156) the message server (2512) to transmit a message to the client computing devices of group members that describes the modification to the group folder.

146. The system as recited in Claim 145, wherein the modification affecting the group folder is the creation of a new file in the group folder, and wherein the transaction coordinator is further configured to:

- cause the storage component to allocate space for raw file data; and

- instruct a folder database to create an entry for the new file and associate the file with the group folder.

147. A method of opening an application package containing an application's process and UI logic, the method comprising:

obtaining (1116) resources of an application being opened as specified in a corresponding application package on a local client computing device;

instantiating (1128) an instance object configured to manage the runtime state of the application; and

initiating (1130) execution of an application's process and view logic utilizing an application object that is allocated memory space by an application manager.

148. The method as recited in Claim 147, wherein obtaining the resources of an application being opened, includes:

evaluating an XML-based expression having a URL that references at least one resource of the application; and

checking resources currently maintained in a client-side cache to determine whether the at least one application resource is available locally.

149. The method as recited in Claim 147, wherein the number of application instances that are instantiated is limited to a specified number as indicated in the application package.

150. The method as recited in Claim 147, wherein obtaining the resources of an application being opened as specified in a corresponding application package includes instantiating an application object configured to multiple instance objects.

151. The method as recited in Claim 147, wherein obtaining the resources of an application being opened, includes:

determining whether a primary network location that stores the resources is accessible; and

if the resources are not available from the primary network location, utilizing semantic logic in the application package to identify a secondary network location.

152. The method as recited in Claim 147, wherein the resources obtained as specified in the application package includes at least one resource selected from an image, XML document, and binary file.

153. The method as recited in Claim 147, wherein the instance object is configured to track at least one object in the view layer and at least one object in the controller layer associated with an application that adheres to the MVC design paradigm.

154. The method as recited in Claim 147, wherein the instance object holds a local expression engine and causes expressions referencing named objects that are global to the application to be evaluated relative to the appropriate application instance.

155. The method as recited in Claim 147, wherein data is provided with the application package describing a state of an executing application serialized in XML and wherein to initiate execution of an application's process and view logic includes re-creating an application state.

156. The method as recited in Claim 147, wherein to initiate execution of an application's process and view logic includes utilizing a data type recognizer to identify a document type from semantic data in the application package and cause actions to be performed that are specific to the identified document type.

157. A computer readable medium having computer executable instructions, that, when executed, by a computer processor performs a method of obtaining one or more application resources in preparation of transitioning to an offline state, comprising:

determining (3408) whether an executing application is configured to cause a dynamic resource to be loaded on the client computing device at runtime, wherein the dynamic resource is not identified in an application package; and

if a dynamic resource is identified that is not cached on the client computing device, obtaining (3412-3414) the dynamic resource from a network service.

158. The computer-readable medium as recited in Claim 157, further comprising:

identifying a static resource utilized by the application as specified in an application package; and

if the static resource is not cached on the client computing device, obtaining the static resource from the network service.

159. The computer-readable medium as recited in Claim 158, wherein identifying a static resource utilized by the application includes defining an XML schema in which the resources and settings associated with an application are described in accordance with a structured semantic format.

160. The computer-readable medium as recited in Claim 158, wherein the static resources utilized by the application are embedded in the application package.

161. The computer-readable medium as recited in Claim 157, further comprising:

notifying the network service that a user of a first client computing device is offline; and

notifying a second client computing devices that subscribed as a listener to the network status of the user that the user is offline.

162. The computer-readable medium as recited in Claim 157, wherein determining whether an executing application is configured to cause a dynamic resource to be loaded on the client computing device includes identifying a resource associated with at least one open UI or process XML document.

163. The computer-readable medium as recited in Claim 147, wherein obtaining the dynamic resource from the network service includes using a communication channel that abstracts operations of a network file system to load the resource in a client-side cache.

164. A system for obtaining application resources when a client computing device is transitioning to an offline state, comprising:

a server-side data center (502) configured to provide access to application resources as a network service;

at least one client computing device (300), comprising:

a processor (302); and

a network interface (310) for exchanging data with the server-side data center over a network; and

a computer-readable media (304) having computer-executable components for obtaining application resources from the server-side data center, comprising:

an application manager (904) configured to identify a static resource utilized by the application as specified in an application package;

an instance component (1406) operative to determine whether application code that provides an application's logic references a dynamic resource that is scheduled to be loaded when the application executes; and

a communication component (2504) configured to obtain an application's static and dynamic resources not stored locally when the client computing device (300) is scheduled to transition to the offline state.

165. The system as recited in Claim 164, wherein the application manager is further configured to notify the server-side data center that the status of the client computing device is offline and wherein the server-side data is further configured to notify at least one listener regarding the change in a user's network status.

166. The system as recited in Claim 164, wherein to determine whether application code that provides an application's logic references a dynamic resource includes identifying a resource associated with at least one open UI or process XML document.

167. A method implemented in instructions executed by a computer processor of initiating execution of an applications process code, the method comprising:

identifying (1106) a data type of a document associated with an application being launched; and

if a determination is made that the document contains an applications process code:

instantiating (1118) a process object having an execute method for executing a process step in the application;

causing (1000) the process object to be associated with an application instance; and

activating (1130) an execute method in the process object and providing the process object a corresponding application instance and process step in the applications process code to execute.

168. The method as recited in Claim 167, wherein the applications process code is described in a process XML document adhering to a first schema, the process XML document having at least one operation element to open a UI XML document that adheres to a second schema.

169. The method as recited in Claim 167, wherein identifying a data type of a document associated with an application being launched includes using a data type recognizer to apply criteria associated with a plurality of data types to structural data in the document.

170. The method as recited in Claim 169, wherein the data type recognizer is configured to cause a different set of actions to be performed when opening a process XML document depending on whether an application package or standalone application is being launched.

171. The method as recited in Claim 167, wherein instantiating a process object, includes:

identifying and registering a trigger defined in a trigger element of a process XML document; and

in response to a user interface component experiencing a trigger event, executing the process step as specified in the trigger element.

172. The method as recited in Claim 167, wherein the instantiated process object allows a second object to register as a listener that is notified in response to the process logic of an executing application being modified.

173. The method as recited in Claim 167, wherein the applications process code is described in an XML document and wherein activating an execute method of the process object includes evaluating an expression that references the XML document.

174. A method implemented in instructions executed by a computer processor of using a process XML schema to execute an applications process code, the method comprising:

defining a process XML schema (600) in which a valid process XML document includes:

a trigger element (602) having one or more attributes for identifying process code to execute in response to a user interface component experiencing a trigger event;

at least one operation element (612) having one or more attributes identifying an operation handler to execute and an associated expression; and

in response to the user interface component experiencing a trigger event with attributes matching the trigger element, causing (1800) logic represented in the at least one operation element (620) to be executed.

175. The method as recited in Claim 174, wherein the trigger element includes view and component attributes for identifying the application view and component associated with the trigger.

176. The method as recited in Claim 174, wherein the trigger element includes event and step attributes specifying the event type and process step to execute in response to the trigger being activated.

177. The method as recited in Claim 174, wherein the operation element includes a component element having one or more attributes identifying a user interface component and an associated XPath expression.

178. The method as recited in Claim 177, wherein causing logic represented in the at least one operation element to be executed includes creating an XBind that references data in a data model.

179. The method as recited in Claim 174, wherein executing the logic represented in the at least one operation element includes opening a UI XML document identified in an expression and causing an application view represented in the UI XML document to be visually rendered on a computer display.

180. The method as recited in Claim 174, wherein the process XML document is generated using a meta-editor in which constraints of a process XML schema are applied to graphical input provided by a user.

181. A system of providing software as a service to one or more client computing devices, comprising:

a network file system (2500) for storing raw data documents and making a users data documents available as a virtual drive on the client computing device;

at least one application (2402) that provides application logic in semantic application code written according to the one or more XML schemas; and

an XML virtual machine (510) delivered from a server computing device configured to interpret semantic application code written according to one or more XML schemas, the semantic application code being translated into platform independent code by the XML virtual machine (510).

182. The system as recited in Claim 181, wherein the XML virtual machine is configured to cause the logic of the at least one application to be executed without being installed as an executable on the client computing device.

183. The system as recited in Claim 181, wherein the XML virtual machine is configured to interpret user interface logic in a format that adheres to a first XML schema and wherein application process logic interpreted by the XML virtual machine is in a format that adheres to a second XML schema.

184. The system as recited in Claim 181, wherein the at least one application that describes application logic according to the one or more XML schemas is configured to provide a common desktop environment in which applications and user data are accessible across different computing devices.

185. The system as recited in Claim 181, wherein the network file system is further configured to facilitate current use of a file by multiple client computing devices with an update to the file being propagated to each of the client computing devices in real time.

186. The system as recited in Claim 181, wherein the network file system is further configured to:

authenticate that a user is a member of a group that has rights to access a root folder in the network file system; and

generate and transmit a file describing a folder tree associated with the group to the client computing device.

187. A method implemented in computer-executable instructions that, when executed by a computer processor, opens a new application view and associates the view with an application instance, the method comprising:

utilizing (1304) a data type recognizer to identify a data type of a file as containing an applications user interface logic;

creating (1308) a view object that causes the application's user interface components to be rendered; and

utilizing (1012) the data type recognizer to pass the resulting view object to an instance object.

188. The method as recited in Claim 187, wherein identifying a data type of a file includes applying hierarchical criteria to structural data in the file, and wherein the data type recognizer is configured to cause a different actions to be performed depending on the type of file that is being opened.

189. The method as recited in Claim 187, wherein causing an applications user interface components to be rendered, includes:

instantiating a runtime object that provides computational logic of a user interface component;

transforming XML-based user interface logic into an HTML-based format capable of being rendered by a Web Browser; and

using the view object to connect graphical elements rendered in the Web browser with the runtime object.

190. The method as recited in Claim 189, wherein the computational logic provided by the runtime object is defined in a programming language that is different from the XML-based user interface logic.

191. The method as recited in Claim 187, further comprising binding data displayed by the user interface component to a set of data in a data model identified with an XPath expression.

192. The method as recited in Claim 187, wherein an identifier of a particular application instance is supplied to the view object, and wherein a name assigned to the view object in a global view manager includes the identifier of the associated application instance.

193. The method as recited in Claim 192, wherein the global view manager uses the identifier in the name assigned to a view object to differentiate between views that are associated with different application instances.

194. The method as recited in Claim 187, wherein a name assigned to the view object in a view manager that is local to the instance does not include the identifier of the application instance.

195. The method as recited in Claim 187, wherein the applications user interface logic is described in a UI XML document adhering to a first schema, and wherein the applications process logic is described in a process XML document that adheres to a second schema.

196. A method implemented in instructions executed by a computer processor of using a UI XML schema to define an applications graphical layout, the method comprising:

defining (518) a UI XML schema in which a valid UI XML document (650) includes at least a view element in which the name of the application view is provided;

specifying a graphical layout of at least one user interface component in elements and attributes in the UI XML document (650); and

in response to an application being launched, instantiating (1050) a runtime object that represents the application view and causing (2300) the at least one user interface component to be rendered on a graphical display in accordance with the graphical layout defined in the UI XML document (650).

197. The method recited in Claim 196, wherein the user interface component is defined in accordance with a component APIs that accepts logic for setting data from a data model on the user interface component and updating data that is visually rendered by the user interface component when the data model experiences a data update.

198. The method recited in Claim 197, further comprising:
instantiating a runtime object that implements the computational logic of the user interface component according to the component APIs; and
initializing the computational logic provided by the runtime object on the visual rendering of the user interface component.

199. The method recited in Claim 196, wherein an attribute associated with a user interface component in the UI XML document identifies a unique name for referencing the user interface component in expressions.

200. The method recited in Claim 196, wherein the application view is rendered without having the application define any source or script code in the UI XML document.

201. The method recited in Claim 196, wherein instantiating a runtime object representing the application view, includes:

identifying a semantic description of an open operation encountered in an application's process code; and

evaluating an expression associated with the open operation that references the view defined in the UI XML document.

202. The method recited in Claim 196, wherein instantiating a runtime object representing the application view includes utilizing a data type recognizer to identify the data type of the UI XML document.

203. A system for opening an application view, comprising:

an open operation handler (1402) configured to execute an open operation defined in an application's process code;

a data type recognizer (1404) operative to determine the data type of a document when called by the open operation handler (1402); and

a view object (1408) configured to:

cause (2300) the graphical elements represented in the document that defines the visual layout of an application view to be rendered; and

connect (1700) the rendered graphical elements with the application's process code.

204. The system as recited in Claim 203, further comprising an instance object configured to couple operational meaning to the execution of the application's process code, and wherein the data type recognizer is further configured to associate the view object with a corresponding instance object.

205. The system as recited in Claim 204, further comprising an expression engine configured to evaluate expressions referencing the view object relative to the appropriate application instance.

206. The system as recited in Claim 203, further comprising one or more user interface components that are created within a relationship hierarchy of the view object and wherein to connect the rendered graphical elements with the applications process code includes notifying a trigger event bus when the one or more user interface components experience a trigger event.

207. A method implemented in computer-executable instructions that, when executed by a computer processor, sets a rule associated with a user interface component in a data model, the method comprising:

instantiating (1308) a runtime object that implements the computational logic of the user interface component;

if the user interface component is a rule user, allocating (2106) a namespace to the components rules in the data model;

applying (2108) the rules of the user interface component through the execution of a rule handler; and

merging (2110) the rule associated with the user interface component into the data model, the rule being merged into the data model in a namespace that is separate from a namespace associated with a different component.

208. The method as recited in Claim 207, wherein in response to a modification to the data model, the user interface component is configured to:

receive a variable describing a location in the data model that was modified; and
if the rules in the data model have been modified, execute the rule handler to insure the integrity of the data model relative to the user interface component.

209. The method as recited in Claim 208, wherein the modification to the data model is represented in a plurality of objects, each object providing an XBind that references a location in an XML document where a modification to the data model occurred.

210. A method implemented in computer-executable instructions that, when executed by a computer processor, performs a partial update of data that is displayed by a user interface component, the method comprising:

binding (1600) a user interface component to a set of data in a data model; and
in response to a modification to the data model, the user interface component is configured to:

receive (2208) a variable describing a location in the data model that was modified;

identify (2216) a subset of data currently displayed by the user interface component that is affected by the modification to the data model; and

update (2220) the binding associated with the user interface component and cause the subset of data currently displayed by the user interface component affected by the data update to be re-rendered.

211. The method as recited in Claim 210, wherein data displayed by the user interface component that is unaffected by the modification to the data model is not re-rendered.

212. The method as recited in Claim 210, wherein binding a user interface component to a set of data in a data model includes configuring the user interface component to satisfy runtime queries for an XBind that references a location in the data model that the user interface component is bound.

213. The method as recited in Claim 210, wherein the modification to the data model is represented in a plurality of objects, each object providing an XBind that references a location in an XML document where a modification to the data model occurred.

214. The method as recited in Claim 210, wherein the variable describing a location in the data model that was modified includes an XPath expression for selecting a subset of data in an XML document.

215. The method as recited in Claim 210, wherein to identify a subset of data currently displayed by the user interface component that is affected by the modification to the data model includes evaluating an XPath expression at runtime while the application is executing.

216. The method as recited in Claim 210, wherein the user interface component is further configured to utilize a rule to interpret content in the data model, the rule allowing a generic component to be adapted for use with a plurality of XML schemas.

217. The method as recited in Claim 216, wherein the rule associated with the user interface component is maintained in a namespace that is separate from a rule associated with a different object bound to the same document.

218. The method as recited in Claim 210, wherein the user interface component is one selected from a tree component, a list component, and a grid component.

219. The method as recited in Claim 210, wherein binding a user interface component to a set of data in the data model, includes:

determining whether the user interface component is experiencing an initial data update; and

if an initial data update is being performed, causing a complete representation of the underlying data referenced in an XBind to be displayed by the user interface component.

220. A method implemented in computer-executable instructions that, when executed by a computer processor, creates a user interface utilizing an Application Programming Interface, the method comprising:

accepting one or more API calls from an application directed at creating a new user interface component, the one or more API calls including call parameters that define at least the computational logic for setting (1608) data from a data model on the user interface component and updating (2218) data that is visually rendered when the data model experiences a data update; and

instantiating (1308) a runtime object that implements the computational logic as defined in the one or more API calls and initializing (2312) the computational logic of the runtime object on a visual rendering of the user interface component.

221. The method as recited in Claim 220, wherein the call parameters of the one or more API calls includes logic to perform both a partial and full update of the data that is visually rendered by the user interface component, wherein a partial update causes a subset of data currently displayed by the user interface component to be re-rendered.

222. The method as recited in Claim 220, wherein the call parameters of the one or more API calls further includes logic that defines an unload function for performing memory management when the user interface component is closed.

223. The method as recited in Claim 220, wherein the runtime object that implements the computational logic for setting data on the user interface component is created without allowing the application to define any source or scripting code.

224. The method as recited in Claim 220, wherein initializing the computational logic of the runtime object, includes:

causing a view of the application to be rendered; and

connecting at least one graphical element in a rendering of the user interface component represented in the application view to the runtime object that provides the computational logic for setting data on the user interface component.

225. The method as recited in Claim 220, wherein logic for visually rendering the user interface component is described in an XSLT that is defined independently from the runtime object that provides the computational logic for setting data on the user interface component.

226. The method as recited in Claim 220, wherein the user interface component created utilizing the Application Programming Interface is configured to inherit properties from a parent component.

227. A method implemented in instructions executed by a computer processor of initializing an application's user interface components, the method comprising:

connecting (2306) an application view to events generated when a user interacts with a visual display of a computing device;

using (2308) logic that describes the visual layout of the application view to render graphical elements of at least one user interface component;

instantiating (2310) a runtime object that provides computational logic of the user interface component, wherein the computational logic is defined independently from logic used to visually render the user interface component; and

initializing (2312) the computational logic of the runtime object on the visual rendering of the user interface component.

228. The method as recited in Claim 227, wherein connecting an application view to events generated when a user interacts with a visual display of a computing device, includes:

registering a listener on a DOM object used by a Web browser to render the application view; and

inserting a unique identifier in the DOM object to associate a graphical element with the runtime object.

229. The method as recited in Claim 228, further comprising:

receiving notice of a user-generated event;
identifying a unique identifier associated with the event from the DOM object;
and
notifying the runtime object activated by the user-generated event.

230. The method as recited in Claim 227, wherein connecting an application view to events generated when a user interacts with a visual display of a computing device includes utilizing a data type recognizer to determine that a document contains an application's user interface logic.

231. The method as recited in Claim 227, wherein connecting an application view to events generated when a user interacts with a visual display of a computing device includes determining if a previously created dialog still maintained in memory to be reassigned to the application view.

232. The method as recited in Claim 227, wherein to render graphical elements of at least one user interface component includes transforming the XML user interface logic into an HTML-based format capable of being rendered by a Web Browser and wherein the computational logic of the user interface component is defined in a programming language that is different from the XML user interface logic.

233. The method as recited in Claim 227, wherein to render graphical elements of at least one user interface component includes accessing the native graphics primitives of a computing device without using the resources of a Web Browser.

234. The method as recited in Claim 227, wherein initializing the computational logic of the runtime object, includes setting default states of the user interface component utilizing a method defined in accordance with component APIs without having an application provide any source or script code.

235. The method as recited in Claim 227, further comprising binding the user interface component to a document in a data model.

236. A computer-readable medium having computer executable instructions that, when executed by a computer processor, performs a method of visually rendering a user interface component, the method comprising:

in a document (650) containing a description of an application's view logic, inserting an XLink that points to view logic associated with the user interface component;

rendering (2300) the application's view logic without activating the XLink;

in response to an executing application causing the XLink to be activated, obtaining (1116) the view logic associated with the user interface component; and

using (2300) the logic obtained when the XLink is activated to render the user interface component.

237. The computer-readable medium as recited in Claim 236, wherein the application's view logic represented in the document is rendered when the application is launched and the user interface component is rendered on demand when accessed by a user.

238. The computer-readable medium as recited in Claim 236, wherein the XLink is inserted as an element in an XML document and provides a bidirectional link between two documents.

239. The computer-readable medium as recited in Claim 236, wherein the view logic referenced in the XLink is not currently stored on a local computing device and obtaining the view logic associated with the user interface component includes loading a document using a channel that abstracts communications with a network service.

240. The computer-readable medium as recited in Claim 236, wherein the view logic referenced in the XLink provides an XML semantic description of the visual layout of the user interface component that does not have any source or script code.

241. The computer-readable medium as recited in Claim 236, wherein the inserted XLink provides the application with sufficient information to obtain the view logic associated with the user interface component without exposing information to a client computing device that would require a separate permission check to access data stored at a network file system.

242. A computing device operative to render an application view and initialize an application's user interface components, comprising:

a processor (300);

a memory (304) having computer-executable modules for visually rendering an application view and initializing computational logic of the user interface component, comprising:

a graphical rendering module (408) operative to render graphical elements on a computer display and receive notice of user-generated events;

at least one user interface component (706) associated with logic used to visually render the user interface component on a computer display and an independently defined runtime object (1562) that provides the computational logic to satisfy user-generated events; and

a view module (1558) configured to connect graphical elements rendered by the graphical rendering module with computational logic provided by the runtime object.

243. The computing device as recited in Claim 242, wherein the graphical rendering module is a Web Browser and wherein to render graphical elements on a computer display includes performing a transform of XML-formatted data to an HTML format.

244. The computing device as recited in Claim 242, wherein the graphical rendering module utilizes native graphics primitives provided by a machine operating system without using the resources of a Web Browser.

245. The computing device as recited in Claim 242, further comprising a binding module for binding data set on the user interface component to a location in a data model identified by an XPath expression.

246. The computing device as recited in Claim 245, wherein the runtime object is further configured to add a rule for interpreting content in the data model to a rule handler.

247. A computer-readable medium having computer executable instructions, that, when executed by a computer processor, performs a method to apply a data manipulation primitive on an XML document, the method comprising:

identifying (1202) an operation in an application that provides logic to modify data in the XML document using a data manipulation primitive;

generating (1203) a variable adhering to a common data type for referencing a location in the XML document affected by the data manipulation primitive; and

calling (2002) a runtime representation of the XML document (e.g., "document object") to implement the data manipulation primitive as specified in the operation, wherein the data manipulation primitive is implemented by the document object at runtime while the application is executing.

248. The computer-readable medium as recited in Claim 247, wherein the data manipulation primitive that is implemented on the XML document at runtime is one selected from the group of replace, replaceText, append, prepend, insert, remove, remove children, new, and delete.

249. The computer-readable medium as recited in Claim 247, wherein to implement the data manipulation primitive as specified in the operation, includes:

causing a transaction to be generated that represents a relative change to the XML document; and

causing the transaction to be transmitted over the network to implement the relative change on a remote computing device.

250. The computer-readable medium as recited in Claim 247, wherein to implement the data manipulation primitive as specified in the operation includes evaluating an expression into text, comprising:

identifying a user interface component referenced in the expression; and

if the user interface component is atomic and not associated with a data binding, accessing text input into the user interface component directly from a corresponding runtime object.

251. The computer-readable medium as recited in Claim 247, wherein generating a variable adhering to a common data type, includes evaluating an expression

in the application into an XBind having a URL and a base path that points to a subset of data in the XML document.

252. The computer-readable medium as recited in Claim 247, wherein logic in the application adheres to the MVC design paradigm and wherein the data manipulation primitive is used in the process logic of the application to modify contents of a data model.

253. The computer-readable medium as recited in Claim 247, wherein to implement the data manipulation primitive as specified in the operation includes performing a transform of data from a source format into a destination format and inserting the data in the destination format into the XML document.

254. The computer-readable medium as recited in Claim 253, wherein the transform is an Extensible Style Sheet Language transform that defines how data formatted in accordance with a first XML schema is translated into a format that adheres to a second XML schema.

255. The computer-readable medium as recited in Claim 247, wherein the data manipulation primitive is defined by an application in a store element having a type attribute for identifying a particular type of data manipulation primitive to implement, an optional select attribute for selecting data to insert into the XML document, and a value attribute for referencing a subset of data in the XML document.

256. A method implemented in computer-executable instructions that, when executed by a computer processor, transforms data from a source format to a destination format, the method comprising:

identifying (1202) an operation in an application that provides logic for performing a data transform;

obtaining (1908) data in the source format from a document identified in the logic of the operation; and

applying (1910) the transform defined in the operation on the data in the source format, wherein the transform is applied at runtime while the application is executing.

257. The method as recited in Claim 256, wherein the transform is applied within an operation that is further configured to insert the data in the destination format into an XML document. .

258. The method as recited in Claim 257, wherein to insert the transformed data into the XML document, includes:

creating a transaction that represents a relative change to an XML document; and
causing a data bus to implement the relative change represented in the transaction.

259. The method as recited in Claim 256, wherein the transformed data in the destination format provides the basis for applying logic defined in a second application.

260. The method as recited in Claim 256, wherein the transform is an Extensible Style Sheet Language transform that defines how data formatted in accordance with a first XML schema is translated into a format that adheres to a second XML schema.

261. The method as recited in Claim 256, wherein obtaining data in the source format from a document identified in the logic of the operation includes evaluating an expression in the application into an XBind that points to a subset of data in the document.

262. The method as recited in Claim 256, wherein to apply the transform defined in the operation includes providing an XML virtual machine configured to execute applications in which application functionality is separated in accordance with the MVC design paradigm, the logic of the transform being defined in a model XML document.

263. A computer-readable media having computer-executable components for applying a data manipulation primitive on an XML document, comprising:

an application component (600) that defines a change operation for modifying data in the XML document using a data manipulation primitive;

a document object (710) providing a runtime representation of the XML document having methods for performing a plurality of data manipulation primitives on the XML document;

a change handler configured to:

parse (1902) the change operation defined in the application component;

from logic in the change operation, generate (1904) an XBind that references a location in the XML document affected by the change operation; and

call (2008) a method of the document object to implement the data manipulation primitive, wherein the method call provides the document object (710) with the XBind that references the location in the XML document affected by the change operation.

264. The computer-readable medium as recited in Claim 263, further comprising:

a communication channel operative to abstract network communications between a local client computing device and a network service; and

a transaction manager operative to create a transaction that represents a relative change to the XML document as defined in the change operation and cause the transaction to be transmitted to the network service using the communication channel.

265. The computer-readable medium as recited in Claim 263, wherein the change handler is further configured to perform an Extensible Style Sheet Language transform on data formatted in accordance with a first XML schema into a second XML schema before implementing the data manipulation primitive on the XML document,

266. The computer-readable medium as recited in Claim 263, wherein the data manipulation primitives implemented by the change handler is one selected from the group of replace, replaceText, append, prepend, insert, remove, remove children, new, and delete.

267. A method of creating a localized relationship hierarchy that associates objects used to execute an application with an instance, the method comprising:

instantiating (1128) an instance object that manages the runtime state of an executing application, the instance object associated with an instance identifier;

providing (1012) the instance object with a reference to at least one object configured to execute operations defined in the applications process code;

supplying (1306) an instance identifier to a view object utilized to implement user interface logic in the application; and

providing (1308) the instance object with a reference to the view object.

268. The method as recited in Claim 267, further comprising in response to the applications user interface experiencing a user-generated event:

causing a component object held by the view object to identify the application instance that experienced the event using the instance identifier; and

supplying the appropriate instance object to the at least one object associated with the applications process logic.

269. The method as recited in Claim 267, wherein the at least one object configured to execute operations defined in the applications process logic is stateless and used by each instance of the application that is executing.

270. The method as recited in Claim 267, wherein the instance object is held by an application object that is allocated a memory space by an application manager.

271. The method as recited in Claim 267, wherein the instance object maintains a local MiniView manager that holds the reference to the view object.

272. The method as recited in Claim 267, wherein the instance object maintains a local MiniProcess manager that holds reference to the at least one object configured to execute operations defined in the applications process code.

273. The method as recited in Claim 267, wherein the instance object maintains a local expression engine for evaluating expressions that reference a data model.

274. The method as recited in Claim 273, wherein the application adheres to the MVC design paradigm and wherein an object associated with the controller layer of the application utilizes the expression engine provided by the instance object to query the data model layer without directly interacting with objects in a different layer.

275. The method as recited in Claim 267, wherein a data type recognizer is configured to pass the instance identifier to the view object and return the reference to the view object for tracking by a MiniView manager.

276. The method as recited in Claim 267, wherein a data type recognizer is configured to pass the reference to the at least one object configured to execute operations defined in the applications process code for tracking by a MiniProcess manager.

277. A method of executing a multi-instance application that adheres to the MVC design paradigm, the method comprising:

establishing (1308) a relationship hierarchy between an instance and at least one object associated with the application's process logic and at least one object associated with the application's user interface logic;

in response to the applications user interface experiencing a user-generated event, causing the at least one object associated with the applications user interface logic to identify (1708) the instance that experienced the event; and

notifying (1712) the object associated with the application's process logic of the user-generated event and supplying the appropriate instance.

278. The method as recited in Claim 277, wherein establishing a relationship hierarchy between an instance and at least one object associated with the application's process logic and the at least one object associated with the application's user interface logic, includes:

instantiating an instance object;

wherein the instance object is configured with a local MiniView manager that holds reference to the at least one object associated with the application's user interface logic and a MiniProcess manager that holds reference to the at least one object associated with the application's process logic.

279. The method as recited in Claim 278, wherein supplying the appropriate instance includes evaluating an expression, comprising:

performing a lookup in a local MiniView manager for a reference to a view object referenced in the expression; and

if the view object is not in the local MiniView manager, performing a lookup in a global view manager for the view object.

280. The method as recited in Claim 277, wherein establishing a relationship hierarchy includes supplying an instance identifier to the at least one object associated with the applications user interface logic.

281. The method as recited in Claim 277, wherein the instance is held by an application object having a memory space that is allocated to the application object by an application manager.

282. The method as recited in Claim 277, wherein notifying the object associated with the application's process logic of the user-generated event is performed utilizing an event manager configured to register an object associated with the controller layer in a MVC application as a listener for an event generated by an object associated with the user interface layer in the MVC application.

283. A system that connects a controller layer of an application to a data model and view layers of the application, the system comprising:

- an expression engine (1260) configured to provide declarative access to the model layer from the process logic of the application without having an object in the controller layer interact directly with an object in the model layer;

- an event manager (914) operative to allow the process logic of the application register a listener for a trigger that originates from an object within the view layer of the application;

- a process component (600) configured to:

- cause (1708) expressions that reference the data model layer to be evaluated by the expression engine relative to the appropriate application instance;

- register (1124) one or more triggers with the event manager; and

- receive (1712) notice from the event manager (914) when an object within the view layer of the application experiences a trigger.

284. The system as recited in Claim 283, wherein to cause expressions referencing the data model layer to be evaluated by the expression engine relative to the

appropriate application instance includes establishing a relationship hierarchy between an instance object and at least one object associated with the controller layer and at least one object associated with the view layer.

285. The system as recited in Claim 283, wherein to cause expressions referencing the data model layer to be evaluated by the expression engine relative to the appropriate application instance includes configuring an instance object to hold the expression engine.

286. The system as recited in Claim 283, wherein receiving notice of the trigger includes causing the object within the view layer of the application to identify an application instance that experienced the trigger.

287. A method of providing an object with current namespace information when a data update is performed that affects one or more rules in a data model, the method comprising:

- receiving (2202) notice when the rules maintained in a document in the data model experience a data update;

- invoking (2206) a rule handler associated with the document that experienced the data update; and

- providing (2208) current namespace information to the object, the current namespace information allowing the object to identify content in the data model associated with the object's rules.

288. A system of managing data synchronization for an executing application, the system comprising:

- a transaction manager (910) configured to receive a request to perform a data update on a document and create a transaction that represents a relative change to the document;

- a document (710) in a data model operative to receive the transaction generated by the transaction manager (910), implement the data update represented in the transaction, and notify one or more listeners of the data update to the data model; and

a data binding component (706) configured to register as a listener for data updates to the document (710), receive notice when the document (710) experienced a data update, and perform an update to the information that is displayed on a user interface to reflect the data update.

289. A system that provides a data source abstraction layer for an application, the system comprising:

a communication manager (2412) configured to create and hold one or more communication channels (2414-2418) for obtaining a resource from an external source, receive a request to obtain the resource from the external source, identify the communication channel configured to handle the request, and cause the communication channel to obtain the resource from the external source; and

a document manager (912) configured to provide a request to the communication manager (2412) to obtain the resource if the resource is only accessible from the external source and hold one or more document objects (2404-2408) that allows the application to access the resource.

290. A system for managing the flow of control in an executing application, comprising:

a process step (612) having one or more operations that define a sequential thread of execution, wherein the flow of execution is returned to a calling object once all of the operations (614-618) in the process step (612) have been executed; and

a call statement within the body of the process step (612) that overrides the default flow of execution and causes execution to be directed to a target process step identified in the call statement, wherein flow of execution returns to the call statement once the target process step completes execution.

291. The system as recited in Claim 290, wherein the call statement is configured to be a time delay call in which flow of execution is directed to the target process step after a specified period of time specified in the call statement.

292. The system as recited in Claim 290, wherein the call statement is configured to be a time interval call in which flow of execution is returned to the process step from the target process step after an interval of time specified in the call statement.

293. The system as recited in Claim 290, wherein the call statement is configured to be an asynchronous call in which flow of execution in the process step continues without waiting for the target process step to complete execution.

294. The system as recited in Claim 290, further comprising a decision operation within the process step that performs a comparison between at least two alternatives, the result of the comparison overriding the default flow of execution and causing execution to be directed to a target process step identified within the body of the decision operation without returning flow of execution back to the process step.

295. The system as recited in Claim 294, wherein to perform a comparison within the body of the decision operation includes evaluating an XPath expression defined within the body of the decision operation.

296. The system as recited in Claim 290, further comprising an open operation that causes the applications user interface logic to be obtained and one or more user interface components to be visually rendered.

297. The system as recited in Claim 290, further comprising a trigger configured to:

- cache the identity of a process step to execute in response to a triggering event;
- and

- in response to the occurrence of the triggering event, directing the flow of execution to the identified process step.

298. A method of managing the flow of executing in an application, the method comprising:

- directing (1130) flow of execution to a process step (612) in the application having one or more operations that define a sequential thread of execution;

upon encountering a call statement between operations in the process step (612), directing (1712) execution to a target process step identified in the call statement; and returning (1200) flow of execution to the call statement once the target process step completes execution.

299. The method as recited in Claim 298, further comprising:

upon encountering a decision operation in the process step, performing a comparison between at least two alternatives; and

based on the results of the comparison, directing flow of execution to a target process step identified in one of the alternatives defined in the body of the decision operation without returning flow of execution back to the process step.

300. The method as recited in Claim 299, wherein performing a comparison between at least two alternatives includes evaluating an XPath expression defined within the body of the decision operation.

301. The method as recited in Claim 298, wherein the call statement is configured to be a time delay call in which flow of execution is directed to the target process step after a specified period of time specified in the call statement.

302. The method as recited in Claim 298, wherein the call statement is configured to be a time interval call in which execution is returned to the process step from the target process step after a specified period of time specified in the call statement.

303. The method as recited in Claim 298, wherein the call statement is configured to be asynchronous and flow of execution in the process step continues without waiting for the target process step to complete execution.

304. The method as recited in Claim 298, wherein directing flow of execution to a process step in the application, includes:

identifying and registering a trigger defined by the application; and

in response to an event that activates the trigger, directing flow of execution from an object that activated the trigger to the identified process step.

305. The method as recited in Claim 298, wherein directing flow of execution as indicated in the call statement causes an asynchronous request for data to be transmitted over a communication channel to a network service.

306. A method of extending an XML virtual machine (512) using a process operation Application Programming Interface, the method comprising:

establishing the XML syntax structure that may exist in the body of an operation (614-618), including:

providing a definition of at least one child element (622) having an associated name and data type; and

providing a definition of at least one attribute associated with the at least one child element (622) having an associated name, data type, and use that specifies whether the attribute is required;

generating an operation handler (1402) for interpreting logic defined in the at least one child element (622), the operation handler (1402) configured to interface with an execute method that passes the same parameter types to each operation handler invoked by an application being executed by the XML virtual machine (510).

307. The method as recited in Claim 306, wherein the parameters passed to the operation handler includes an instance object, process object, and an evaluated XBind.

308. The method as recited in Claim 307, wherein the operation handler is supplied everything used to interpret logic in the body of the operation without returning data to the execute method.

309. The method as recited in Claim 306, wherein the operation handler is a standalone function that is accessed from a network service without being executed directly on a client computing device.

310. A method for obtaining a list of files in a folder, the method comprising:

receiving (2908) a folder identifier in a request for a list of files in the folder that may be accessed by the user;

querying (2910) an indexing server to retrieve a list of files associated with the folder identifier;

searching (2912) the list of files for one or more restricted files, each restricted file having associated data that indicates that the user does not have access rights to the file;

removing (2914) one or more files from the list of files; and

transmitting (2916) the list of files to a client computing device.

311. A method of providing privacy within the context of a group membership, the method comprising:

obtaining (3204) a global user name utilized to reference a user in a community that does not have a defined membership;

creating (3300) a group having an associated folder that stores a user-defined handle for referencing the user within the context of the group;

mapping (2704) the global user name to a user-defined handle of one or more groups in which the user is a member; and

in communications to group members, resolving (3100) the identity of the user between the global user name and the user-defined handle.

312. A method of providing a composite application that accesses a plurality of remote data sources, the method comprising:

defining one or more communication channels (2414-2418) for obtaining data from each of the remote data sources;

binding (1600) at least one component to a document in a data model that abstracts data received over the one or more communication channels; and

updating (2300) the user interface of the component when data is received over the one or more communication channels (2414-2418) from a remote source.

313. Computer readable media, computing devices, apparatus, methods, and systems as illustrated and described herein.