US 20170116251A1

(19) United States

(12) Patent Application Publication

Chasman et al.

(10) Pub. No.: US 2017/0116251 A1

(43) Pub. Date: Apr. 27, 2017

(54) SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR CREATING A VISUAL COMPONENT FOR TENANTS OF AN ON-DEMAND DATABASE SERVICE

(71) Applicant: **salesforce.com, inc.**, San Francisco, CA (US)

(72) Inventors: **Douglas Chasman**, Pittsford, NY (US); **K. Andrew Waite**, Novato, CA (US)

(21) Appl. No.: **15/249,918**

(22) Filed: **Aug. 29, 2016**

Related U.S. Application Data

(63) Continuation of application No. 13/873,126, filed on Apr. 29, 2013, now Pat. No. 9,430,536, which is a continuation of application No. 12/357,994, filed on Jan. 22, 2009, now Pat. No. 8,468,117.

(60) Provisional application No. 61/022,784, filed on Jan. 22, 2008.

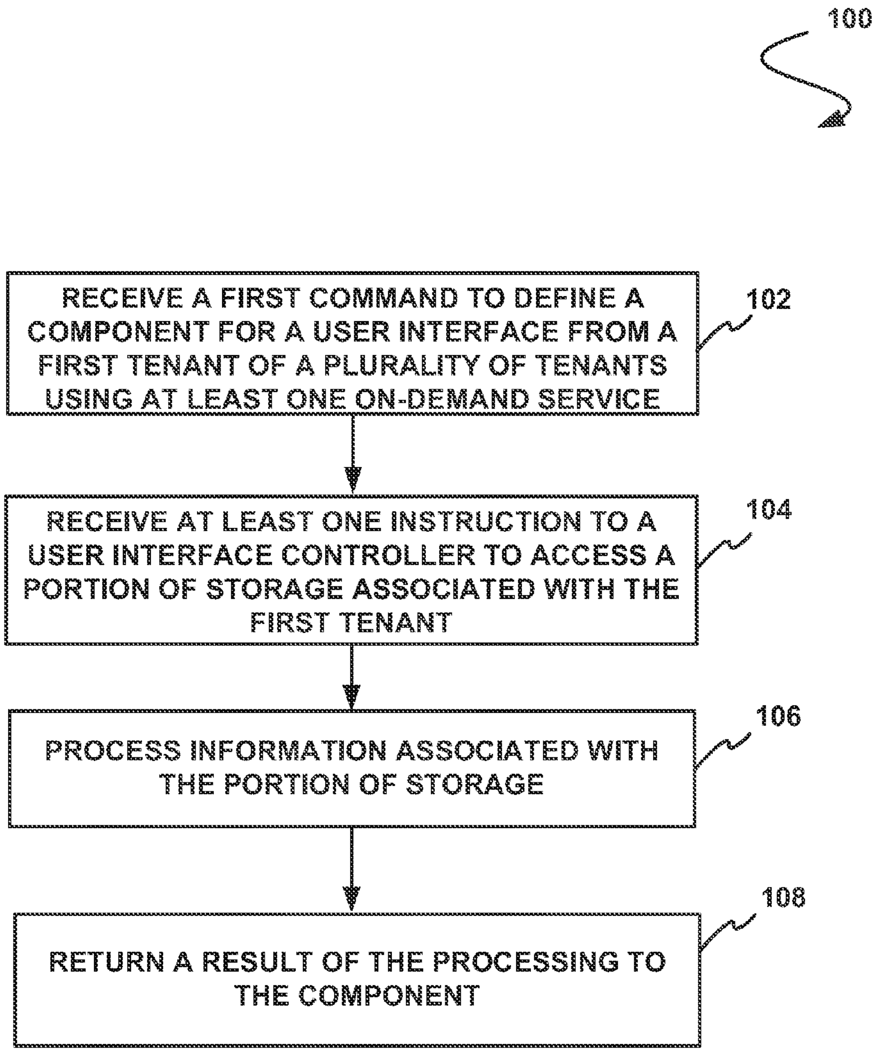
Publication Classification

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
CPC .. ***G06F 17/30371*** (2013.01); ***G06F 17/30557*** (2013.01); ***G06F 17/30312*** (2013.01); *G06F 17/30592* (2013.01)

(57) **ABSTRACT**

In accordance with embodiments, there are provided mechanisms and methods for creating visual components for tenants of an on-demand database service. These mechanisms and methods for creating visual components for tenants of an on-demand database service can enable embodiments to allow tenants to create reusable components for use in user defined interfaces, etc. The ability of embodiments to provide such feature may allow tenants to efficiently and effectively create interfaces.


100
RECEIVE A FIRST COMMAND TO DEFINE A COMPONENT FOR A USER INTERFACE FROM A FIRST TENANT OF A PLURALITY OF TENANTS USING AT LEAST ONE ON-DEMAND SERVICE
102
RECEIVE AT LEAST ONE INSTRUCTION TO A USER INTERFACE CONTROLLER TO ACCESS A PORTION OF STORAGE ASSOCIATED WITH THE FIRST TENANT
104
PROCESS INFORMATION ASSOCIATED WITH THE PORTION OF STORAGE
106
RETURN A RESULT OF THE PROCESSING TO THE COMPONENT
108

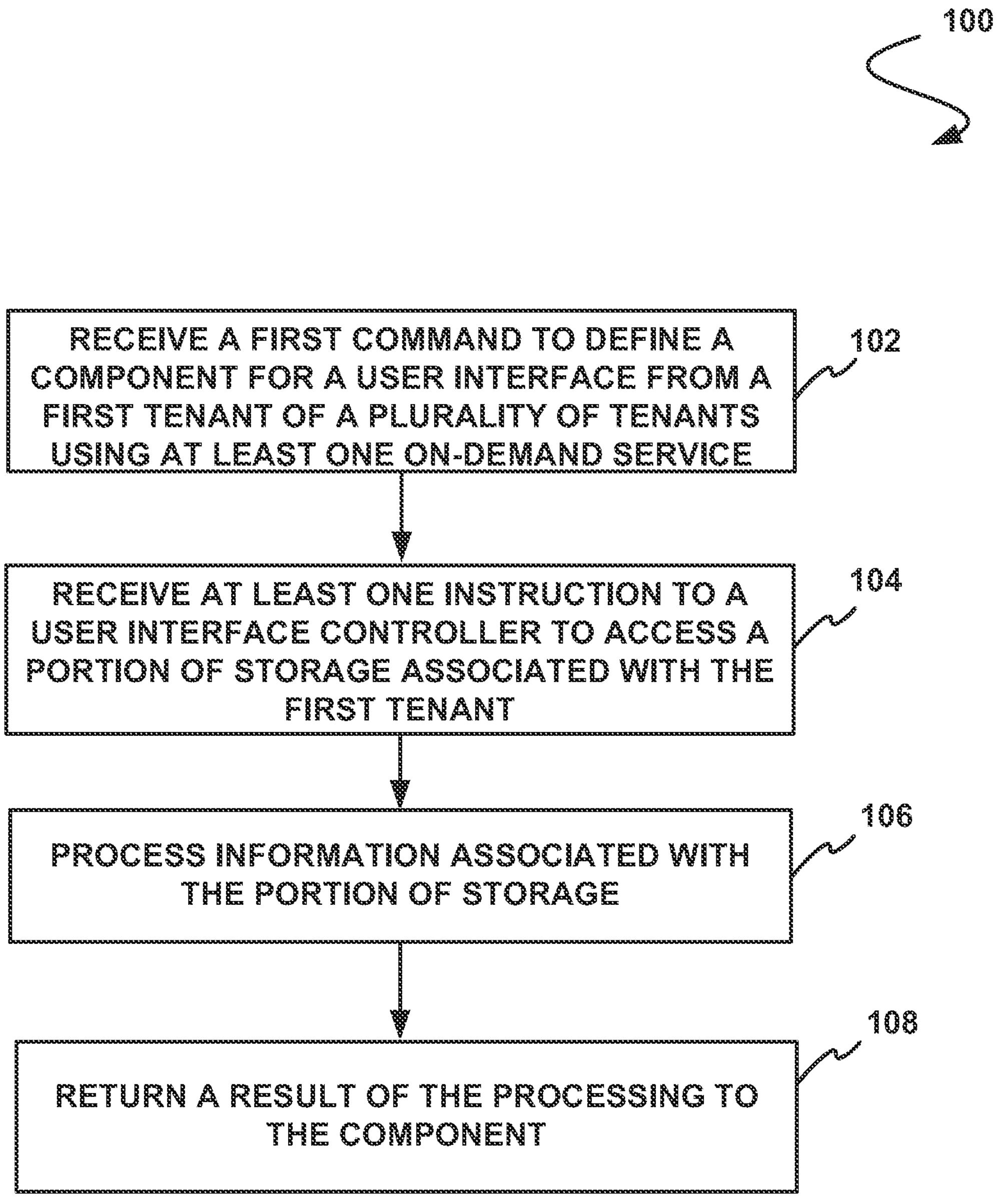

100
RECEIVE A FIRST COMMAND TO DEFINE A COMPONENT FOR A USER INTERFACE FROM A FIRST TENANT OF A PLURALITY OF TENANTS USING AT LEAST ONE ON-DEMAND SERVICE
102
RECEIVE AT LEAST ONE INSTRUCTION TO A USER INTERFACE CONTROLLER TO ACCESS A PORTION OF STORAGE ASSOCIATED WITH THE FIRST TENANT
104
PROCESS INFORMATION ASSOCIATED WITH THE PORTION OF STORAGE
106
RETURN A RESULT OF THE PROCESSING TO THE COMPONENT
108

FIGURE 1

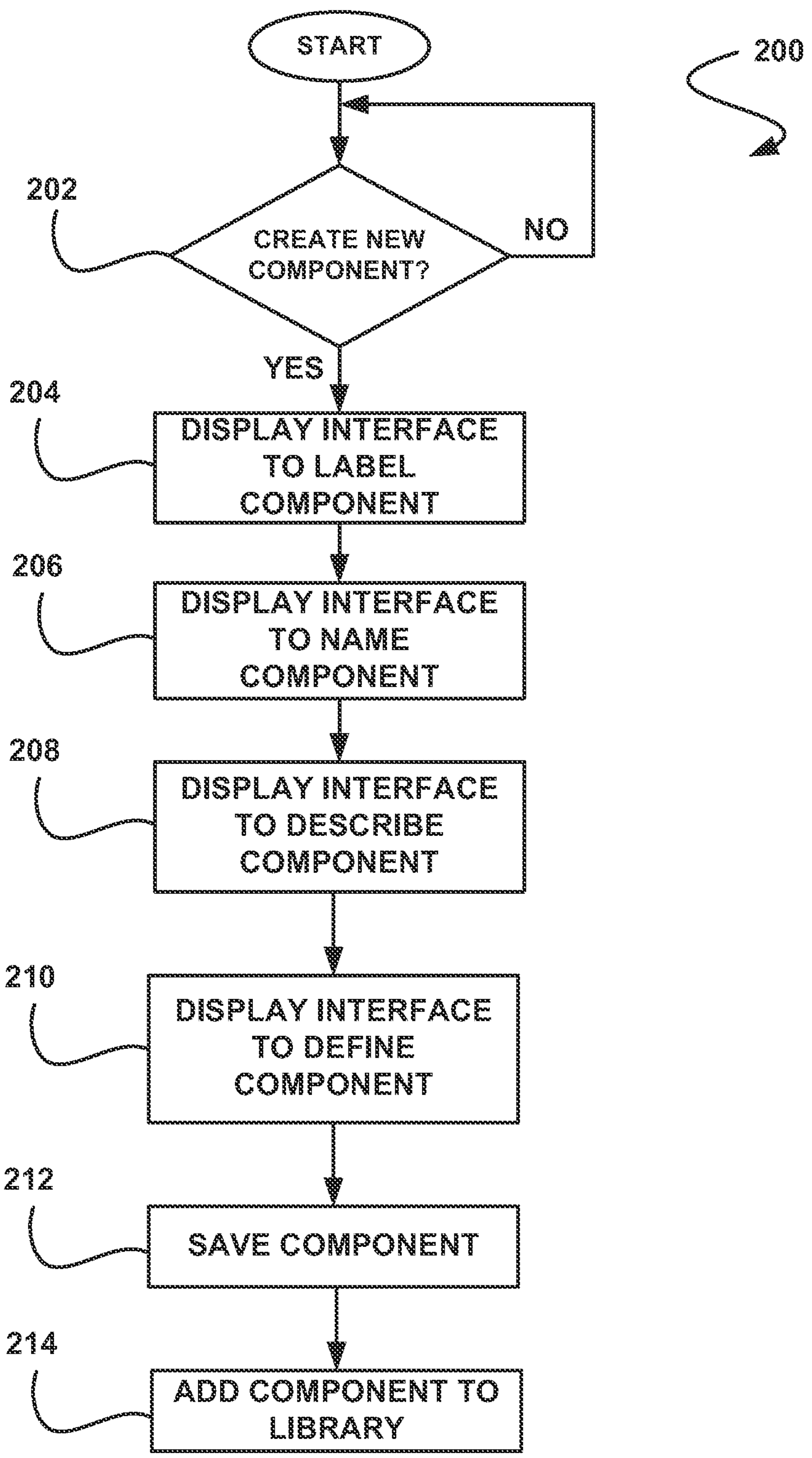

START
200
202
CREATE NEW COMPONENT?
NO
YES
204
DISPLAY INTERFACE TO LABEL COMPONENT
206
DISPLAY INTERFACE TO NAME COMPONENT
208
DISPLAY INTERFACE TO DESCRIBE COMPONENT
210
DISPLAY INTERFACE TO DEFINE COMPONENT
212
SAVE COMPONENT
214
ADD COMPONENT TO LIBRARY

FIGURE 2

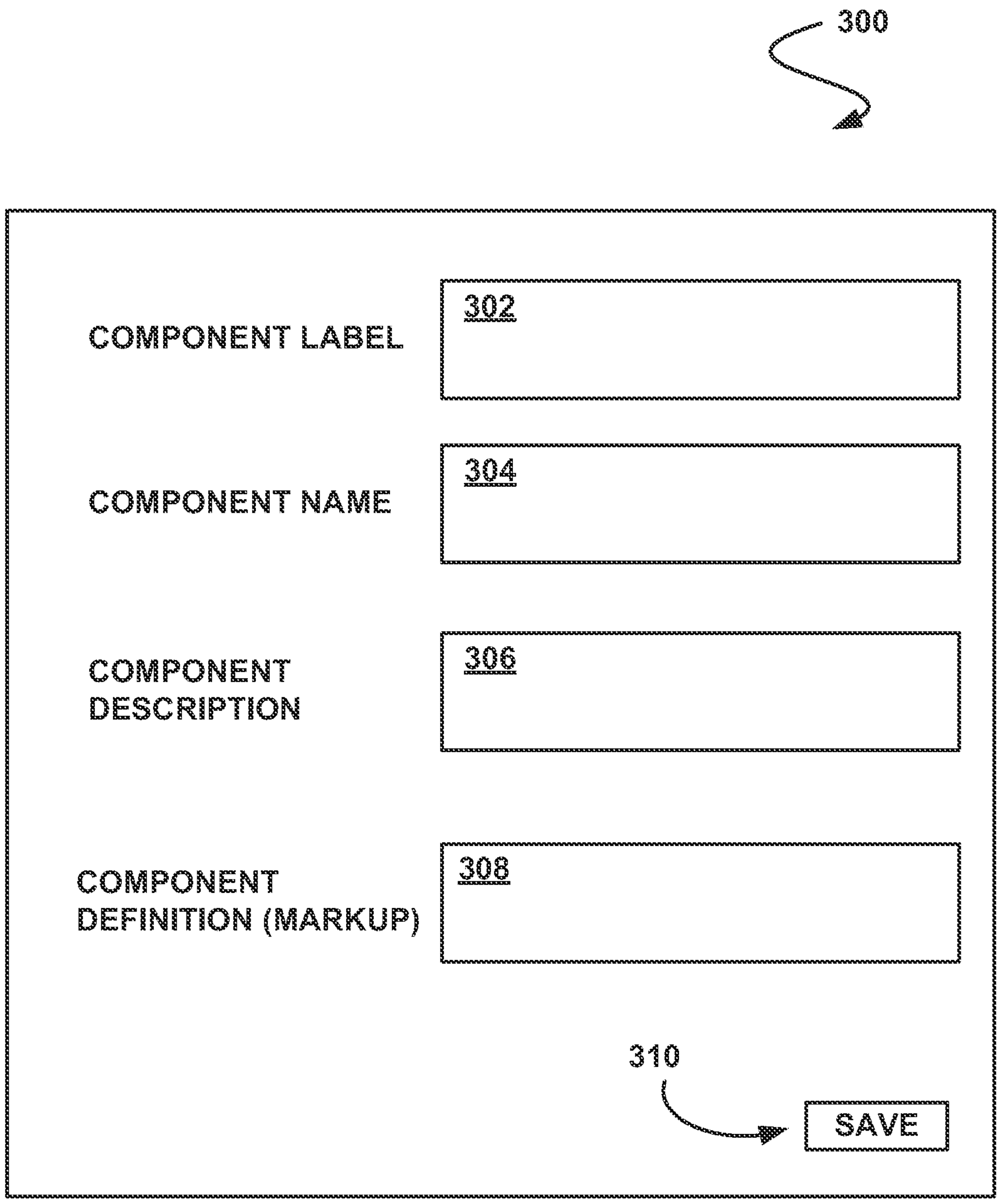

300
COMPONENT LABEL
302
COMPONENT NAME
304
COMPONENT DESCRIPTION
306
COMPONENT DEFINITION (MARKUP)
308
310
SAVE

FIGURE 3

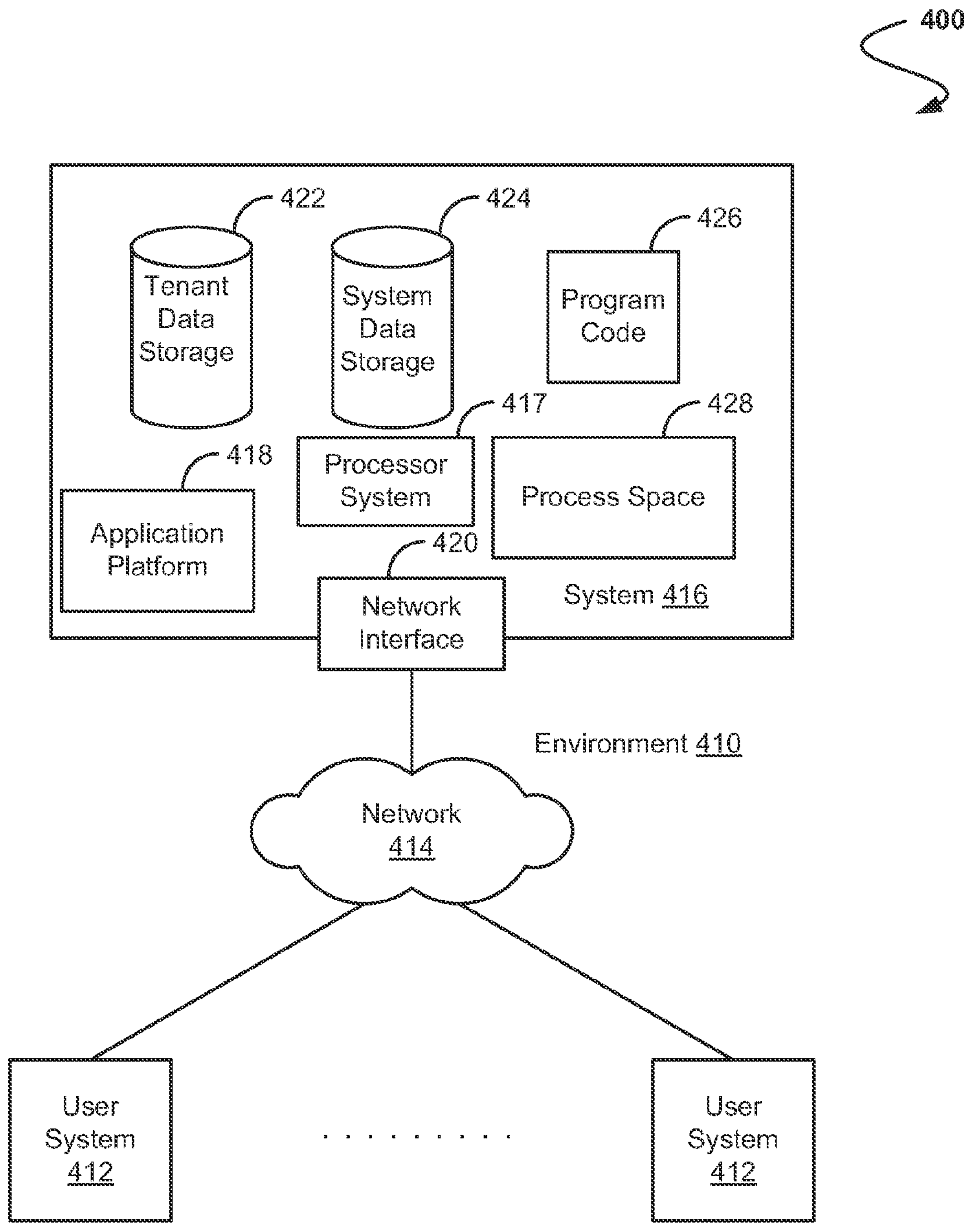

400
422
Tenant Data Storage
424
System Data Storage
426
Program Code
417
Processor System
428
Process Space
418
Application Platform
420
Network Interface
System 416
Environment 410
Network 414
User System 412
User System 412

FIGURE 4

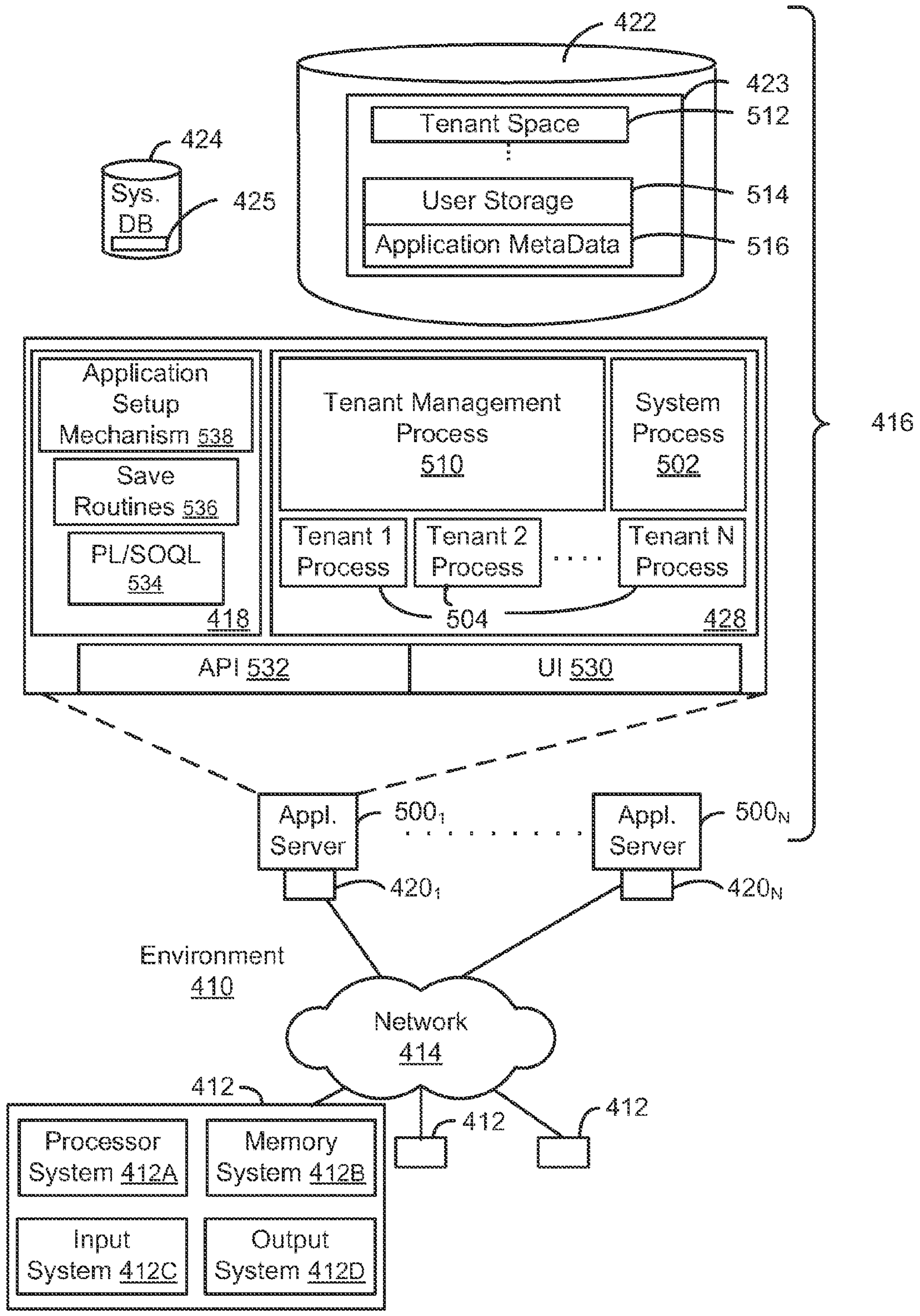

422
423
Tenant Space
512
User Storage
514
Application MetaData
516
424
Sys. DB
425
416
Application Setup Mechanism 538
Save Routines 536
PL/SOQL 534
418
Tenant Management Process 510
System Process 502
Tenant 1 Process
Tenant 2 Process
Tenant N Process
504
428
API 532
UI 530
Appl. Server
500₁
Appl. Server
500N
420₁
420N
Environment 410
Network 414
412
Processor System 412A
Memory System 412B
Input System 412C
Output System 412D
412
412

FIGURE 5

SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR CREATING A VISUAL COMPONENT FOR TENANTS OF AN ON-DEMAND DATABASE SERVICE

CLAIM OF PRIORITY

[0001] This application is a continuation of U.S. application Ser. No. 12/357,994, filed Jan. 22, 2009, which claims the benefit of U.S. Provisional Patent Application No. 61/022,784, filed Jan. 22, 2008, the entire contents of which are incorporated herein by reference.

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0003] The current invention relates generally to on-demand services, and more particularly to creating visual objects for tenants of such systems.

BACKGROUND

[0004] The subject matter discussed in the background section should not be assumed to be prior art merely as a result of its mention in the background section. Similarly, a problem mentioned in the background section or associated with the subject matter of the background section should not be assumed to have been previously recognized in the prior art. The subject matter in the background section merely represents different approaches, which in and of themselves may also be inventions.

[0005] In conventional database systems, users access their data resources in one logical database. A user of such a conventional system typically retrieves data from and stores data on the system using the user's own systems. A user system might remotely access one of a plurality of server systems that might in turn access the database system. Data retrieval from the system might include the issuance of a query from the user system to the database system. The database system might process the request for information received in the query and send to the user system information relevant to the request.

[0006] There is often a desire to allow users to generate and edit user interfaces using information associated with the foregoing database frameworks. However, generating and editing these interfaces requires access to component information associated with such interfaces. In general, the component information is limited to information included in a library maintained and provided by the provider of the database service. This restricts the ability of users to efficiently and effectively edit and generate interfaces that fit the needs of the users.

BRIEF SUMMARY

[0007] In accordance with embodiments, there are provided mechanisms and methods for creating visual components for tenants of an on-demand database service. These mechanisms and methods for creating visual components for tenants of an on-demand database service can enable embodiments to allow tenants to create reusable components for use in user defined interfaces, etc. The ability of embodiments to provide such feature may allow tenants to efficiently and effectively create interfaces.

[0008] In an embodiment and by way of example, a method is provided for creating visual components for tenants of an on-demand database service. In use, a first command to define a component for a user interface is received from a first tenant of a plurality of tenants using at least one on-demand service. Additionally, at least one instruction to a user interface controller to access a portion of storage associated with the first tenant is received. Furthermore, information associated with the portion of storage is processed. Still yet, a result of the processing is returned to the component.

[0009] While the present invention is described with reference to an embodiment in which techniques for creating visual components for tenants of an on-demand database service are implemented in an application server providing a front end for a multi-tenant database on-demand service, the present invention is not limited to multi-tenant databases or deployment on application servers. Embodiments may be practiced using other database architectures, i.e., ORACLE®, DB2® and the like without departing from the scope of the embodiments claimed.

[0010] Any of the above embodiments may be used alone or together with one another in any combination. Inventions encompassed within this specification may also include embodiments that are only partially mentioned or alluded to or are not mentioned or alluded to at all in this brief summary or in the abstract. Although various embodiments of the invention may have been motivated by various deficiencies with the prior art, which may be discussed or alluded to in one or more places in the specification, the embodiments of the invention do not necessarily address any of these deficiencies. In other words, different embodiments of the invention may address different deficiencies that may be discussed in the specification. Some embodiments may only partially address some deficiencies or just one deficiency that may be discussed in the specification, and some embodiments may not address any of these deficiencies.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] FIG. **1** shows a method for creating visual components for tenants of an on-demand database service, in accordance with one embodiment.

[0012] FIG. **2** shows a method for creating a component for use with an interface, in accordance with one embodiment.

[0013] FIG. **3** shows a user interface for creating a component for use with an interface, in accordance with one embodiment.

[0014] FIG. **4** illustrates a block diagram of an example of an environment wherein an on-demand database service might be used.

[0015] FIG. **5** illustrates a block diagram of an embodiment of elements of FIG. **4** and various possible interconnections between these elements.

DETAILED DESCRIPTION

General Overview

[0016] Systems and methods are provided for creating visual components for tenants of an on-demand database service.

[0017] There is often a desire to allow users to generate and edit user interfaces using information associated with the foregoing database frameworks. However, generating and editing these interfaces requires access to component information associated with such interfaces. In general, the component information is limited to information included in a library maintained and provided by the provider of the database service. This restricts the ability of users to efficiently and effectively edit and generate interfaces that fit the needs of the users.

[0018] Thus, mechanisms and methods are provided herein for creating visual components for tenants of an on-demand database service and can enable embodiments to allow tenants to efficiently and effectively create interfaces, etc. The ability of embodiments to provide such feature may lead to a more efficient generation of user interfaces by tenants of on-demand services.

[0019] Next, mechanisms and methods for creating visual components for tenants of an on-demand database service will be described with reference to exemplary embodiments.

[0020] FIG. **1** shows a method **100** for creating visual components for tenants of an on-demand database service, in accordance with one embodiment. As shown, a first command to define a component for a user interface is received from a first tenant of a plurality of tenants using at least one on-demand service. See operation **102**.

[0021] In the context of the present description, an on-demand service refers to any service that that is accessible over a network. In one embodiment, the on-demand service may include an on-demand database service. In this case, an on-demand database service may include any service that relies on a database system that is accessible over a network.

[0022] In one embodiment, the on-demand database service may include a multi-tenant on-demand database service. In the present description, such multi-tenant on-demand database service may include any service that relies on a database system that is accessible over a network, in which various elements of hardware and software of the database system may be shared by one or more customers. For instance, a given application server may simultaneously process requests for a great number of customers, and a given database table may store rows for a potentially much greater number of customers.

[0023] Additionally, in the context of the present description, a tenant refers to any user of the on-demand service. For example, in various embodiments, the tenants may include customers, subscribers, developers, and any other users of the on-demand service.

[0024] It should be noted that the user interface may include any interface capable of being utilized by a user. For example, in various embodiments, the user interface may include a web page, a web browser, an interface provided in an email, an email template, and/or any other interface capable of being utilized by a user.

[0025] Furthermore, a component refers to any item encapsulating at least a definition of one or more visual objects and associated attributes, where the visual objects may be associated with a user interface. For example, in various embodiments, the component may include tags such as HTML, Javapages, and ASP.Net, etc. Additionally, the component may include markup language as well as parameters describing behavior.

[0026] As shown further in FIG. **1**, at least one instruction to a user interface controller to access a portion of storage associated with the first tenant is received. See operation **104**. In this case, the user interface controller may include any controller for controlling the user interface and/or accessing the portion of storage. Furthermore, the user interface controller may include any logic to facilitate such control and/or access.

[0027] Once the portion of the storage associated with the first tenant is received, information associated with the portion of storage is processed. See operation **106**. Furthermore, a result of the processing is returned to the component. See operation **108**.

[0028] For example, in one embodiment, the first command may include at least one attribute. The attribute may include any attribute associated with the component, such as an image attribute (e.g. color, size, etc.), a text attribute (e.g. font, size, etc.), and/or any other attribute associated with the component.

[0029] In this case, accessing the portion of storage associated with the first tenant may include accessing at least one value of the at least one attribute. Thus, the information associated with the portion of storage that is processed may include the at least one value of the at least one attribute. Subsequently, a result of the processing of the at least one value may be returned back to the component for output.

[0030] In this case, the output may include an output to a storage location, an output to a display, an output to a buffer, and an output to various other devices. In one embodiment, the output may include displaying the result of the processing to the first tenant.

[0031] In another embodiment, the at least one instruction to the user interface controller to access a portion of storage, or another instruction, may define the component as a reusable component. In this case, the instruction defining the component as the reusable component may be stored. For example, the instruction defining the component as the reusable component may be stored in a storage area associated with the first tenant. In this case, the storage area associated with the first tenant may include the portion of storage associated with the first tenant.

[0032] In one embodiment, the reusable component may be executed. For example, the execution of the reusable component may be triggered by a second command, different than the first command. In another case, the execution of the reusable component may be triggered by a program initiated command.

[0033] As another option, the execution of the reusable component may be triggered by an event. In this case, the event may include an event external to a computing system executing the reusable component. Of course, the event may include any event capable of triggering the execution of the reusable component.

[0034] FIG. **2** shows a method **200** for creating a component for use with an interface, in accordance with one embodiment. As an option, the present method **200** may be implemented in the context of the functionality of FIG. **1**. Of course, however, the method **200** may be carried out in any desired environment. The aforementioned definitions may apply during the present description.

[0035] As shown, it is determined whether a command is received to create a new component. See operation **202**. For example, in one embodiment, the command may be received at an on-demand database service from a tenant of the on-demand database service. In this case, the tenant of the database service may be presented with a user interface that allows the tenant to select an option to create a new component.

[0036] If it is determined that a new component is to be created, an interface is displayed to the tenant for labeling the new component. See operation **204**. For example, in one embodiment, a text box may be displayed for labeling the new component. In the label text box, text may be entered that is used to identify the new component in a tools menu, for example.

[0037] In addition to the interface for labeling the new component, an interface for naming the new component may be displayed. See operation **206**. For example, a text box for naming the new component may be displayed. In the name text box, text may be entered that is used to identify the new component in markup language. In one embodiment, this name may only contain alphanumeric characters, starting with a letter, and may be unique from all other components in an organization (e.g. an organization associated with the tenant, etc.).

[0038] An interface for describing the new component may also be displayed. See operation **208**. This interface may include a text box for describing the new component. In the description text box, a text description of the new component may be entered. As an option, this description may appear in a component reference library with other standard component descriptions when the new component is saved.

[0039] Once the component is labeled and named, an interface for defining the new component is displayed. See operation **210**. For example, a text box for inserting body text of the component may be displayed. In the body text box, markup language for the custom component definition may be entered.

[0040] It should be noted that any of the aforementioned interfaces and text boxes may be displayed or combined in one interface or may be displayed individually. Once the component is defined, the component may be saved. See operation **212**.

[0041] In one embodiment, the body text may be examined to determine the validity upon saving the component. In this case, invalid text may result in the component being rejected and a user being prompted to remedy the text. If the text defining the component is valid, the component may then be added to a component library. See operation **214**.

[0042] In one embodiment, a library of standard, pre-built components (e.g. <apex:relatedList>, <apex:dataTable>, etc.) may be provided to tenants that may be used to develop user interfaces (e.g. web pages, etc.). Thus, the new components may be built to augment this library. These new components may be referred to as custom components as they are tenant developed components.

[0043] As an option, a common design pattern may be encapsulated in a custom component such that the component may be reused several times in one or more user interfaces or web pages, etc.

[0044] For example, a user may desire to create a photo album using an on-demand service. Each photo in the album may have a unique border, color, and text caption. Rather than repeating the markup language required for displaying every photo in the album, a custom component named "singlePhoto" may be defined that has attributes for image, border color, and caption. These attributes may be used to display the image on the web page. Once defined, every page (i.e. interface) in an organization may utilize the singlePhoto custom component in the same way as a page may utilize standard components from a library, such as <apex:dataTable> or <apex:relatedList>.

[0045] Unlike page templates, which also enable developers to reuse markup language, custom components provide a great deal of power and flexibility. For example, custom components allow developers to define attributes that may be passed into each component. The value of an attribute may then change the way the markup language is displayed on a final page, and may also change controller-based logic that executes for that instance of the component.

[0046] This behavior differs from that of templates, which do not have a way of passing information from the page that uses a template to the definition of the template itself. Additionally, custom component descriptions may be displayed in a component reference dialog of an application, alongside standard component descriptions. Template descriptions, on the other hand, may only be referenced through a setup area/menu because they are defined as pages.

[0047] In one embodiment, all markup language for a custom component may be defined within an <apex:component> tag. This tag may be the top-level tag in a custom component definition.

[0048] It should be noted that the <apex:component> tag may include a variety of attributes. For example, an "access" attribute may be provided for indicating whether the component can be used outside of any page in the same namespace as the component. In this case, possible values may include "public" access and "global" access. The global value may be used to indicate the component can be used outside of the namespace of the component.

[0049] In one embodiment, to use a custom component in a page, the name of the component may be prefixed with the namespace in which the component was defined. For example, if a component named "myComponent" is defined in a namespace called "myNS," the component may be referenced in a page as <myNS:myComponent>.

[0050] As an option, for ease of use, a component that is defined in the same namespace as an associated page may also use a "c" namespace prefix. Consequently, if the page and component from the example above are defined in the same namespace, the component may be referenced as <c:myComponent>.

[0051] In one embodiment, apart from standard markup language, the body of an <apex:component> tag may also specify the attributes that can be passed in to the custom component when it is used in a page. The values of such attributes may then be used directly in the component, or within the controller of the component, if applicable.

[0052] As an option, attributes may be defined with the <apex:attribute> tag. The <apex:attribute> tag may require values for its name, description, and type attributes. In this case, the name attribute may define how the custom attribute may be referenced in pages.

[0053] The value for this attribute may be unique from the names of all other attributes defined in the component. The description attribute may define the help text for the attribute that appears in the component reference library once the custom component has been saved. The custom component may be listed in the reference library with the standard components that are also available. In one embodiment, the type attribute may define the Apex data type of the attribute.

[0054] As an option, the values of the data types may be limited. For example, in one embodiment, the values of the data types may be limited to primitives (e.g. string, integer, or Boolean, etc.), sObjects (e.g. Account, My_Custom_Object_c, or the generic sObject type, etc.), one-dimensional lists specified using array-notation (e.g. String[ ], or Contact[ ], etc.), and custom Apex classes.

[0055] In one embodiment, custom components may be associated with a controller written in Apex. This association may be made by setting a controller attribute included in the component to a custom controller. The controller may also be used to perform additional logic before returning the markup language of the component to the associated page.

[0056] To access the value of a custom component attribute in an associated custom component controller, a class variable in the custom component controller may be defined to store the value of the attribute, and a setter technique for the class variable may be defined. In the <apex:attribute> tag in the component definition, an "assignTo" attribute may be used to bind the attribute to the class variable that was defined.

[0057] In another embodiment, a custom component may be created in development mode by adding a reference to a custom component that does not yet exist to page markup language. After saving the markup language, a quick fix link may appear that allows the creation of a new component definition (including any specified attributes) based on the name provided for the component.

[0058] In one embodiment, any created pages, components, or custom controllers may be included in an application that is to be exchanged. In this case, the content of a page in a managed package may not be hidden when the package is installed. However, as an option, custom controllers, controller extensions, and custom components may be hidden. In addition, custom components may be restricted, using the access attribute, to run only in a specified namespace.

[0059] FIG. **3** shows a user interface **300** for creating a component for use with an interface, in accordance with one embodiment. As an option, the user interface **300** may be implemented in the context of the functionality of FIGS. **1-2**. Of course, however, the user interface **300** may be implemented in any desired environment. Again, the aforementioned definitions may apply during the present description.

[0060] As shown, the user interface **300** includes a component label text box **302**, a component name text box **304**, a component description text box **306**, a component definition text box **308**, and a save button **310**. In operation, a developer may utilize the user interface to create a component for use with an interface, such as a web page, etc.

[0061] For example, Apex pages may materialize to a user through a browser by virtue of an HTML rendering. In this case, components may produce HTML. One approach for component authoring is to encapsulate markup language with a component definition in a manner consistent with creating a basic page.

[0062] Similar to componentization of code into discreet re-usable elements, user interface elements may be broken down into re-usable components that are used by developers with their respective tags to construct a user experience. A developer may create similar structures or may simply use standard components in a very similar way in a number of pages. The developer may like to materialize a reusable component in the same way the standard components of a library are provided. A custom component may have all the capabilities of a standard component with respect to how it is consumed and utilized by the developer using it, versus or in addition to standard components.

[0063] One difference between a component and a page is that a controller of a component may be limited and interactions with the database may be handled using pass through attribute definitions. In some cases, classes associated to component definitions may only access data that is passed into the component through attributes of the component and may not necessarily directly query or execute DML operations.

[0064] As an option, such actions may be handled by a controller of a consumer and may be exposed as action attributes on the component. Data supplied to the component using the attributes may, in turn, be used by the page directly or by the Apex class associated to the component.

[0065] As an example, a developer may desire to create a component that generates a sorted list of text values using a custom component. First, the component may need to define an attribute for the collection of strings to sort and output in the page. The controller may then need to access the value of this attribute, sort the collection, and return the sorted values back to the component for output.

[0066] In one embodiment, values passed into the component implementation by the consumer through the attributes of the component may be accessible from the controller of the component by defining an appropriate setter for the attribute and binding it through an "assignTo" attribute.

[0067] For example, Table 1 shows an attribute definition in the component markup language, in accordance with one embodiment.

TABLE 1

```
<apex:attribute name="value" type="String" required="true" assignTo="{!value}"/>
```

[0068] Table 2 shows an Apex class associated to the component, in accordance with one embodiment.

TABLE 2

```
class myComponentController {
String value;
public void setValue(String s) { value = s; }
}
```

[0069] Accessing the same value from within the component markup language may be the same as though there is a getter in the component controller. Table 3 shows an attribute definition in the component markup language, in accordance with another embodiment.

TABLE 3

```
<apex:attribute name="value" type="String" required="true"/>
<apex:outputText value="{!value}"/>
```

[0070] In various embodiments, getters for value bindings in component markup language may support either an attribute component tag, as in the example above, or a traditional binding to a getter in the Apex controller class. As an option, these binding mechanisms may be mutually exclusive binding mechanisms for which there may be a compile time check upon saving of the component markup language.

[0071] In one embodiment, when a component is saved, one or more validations may occur. For example, in various embodiments, these validations may include verifying that attribute types are defined types or primitives, verifying that value binding expressions are supported by the appropriate items in the controller class of the component, verifying that a getter method or an attribute tag is defined for a value binding expression, and verifying that the name of the component is unique in the given namespace.

[0072] In some cases, there may be specific issues with components that require either being a descendent in a tree from a form, or that require that they are not a descendent of a form. In this case, when compiling a component, the typical page validation that asserts there is a form may be skipped so a component with inputs may be plugged into a page with an existing form.

[0073] Because components may be used within pages, a dependency may exist between a custom component and any pages that utilize the component. In one embodiment, a component detail page may include a "Where is this used?" button that may present the list of pages that are currently dependent on the component.

[0074] As an option, custom components may appear in the component reference library as soon as they are created. In one embodiment, development of components may occur in a separate organization (e.g. in a sandbox, etc.) and may be deployed to a production environment by virtue of either packaging or a metadata API. Additionally, a custom component may have the ability to support all the reference features that standard components implement.

[0075] After a component has been defined, it may then be used in any page based on the access defined on the component. For example, Table 4 shows a component definition in accordance with one embodiment.

TABLE 4

```
<apex:component name="orderedList" description="This component takes a collection of strings
        and outputs it into an ordered list" access="public">
<apex:attribute name="value" type="List<String>"/>
<ol>
<apex:repeat value="{!value}" var="s">
<li>{!s}</li>
</apex:repeat>
</ol>
</apex:component>
```

[0076] The optional component definition illustrated in Table 4 may be used within the same namespace with the markup language shown in Table 5, in accordance with one embodiment.

TABLE 5

```
<c:orderedList value="{!mystrings}"/>
```

[0077] As another exemplary implementation of creating components, a user may desire to output a pre-ordered set of contact names. In this case, it may be expected that the consumer provides the list of contact records in the desired order.

[0078] Table 6 shows markup language that may be used by the user to include this component in a page of the user.

TABLE 6

```
<c:orderedList value="{!mycontacts}"/>
```

[0079] The component definition in this case may change from the example above simply by altering the attribute type and the output expression. Table 7 shows an example of this alteration, in accordance with one embodiment.

TABLE 7

```
<apex:component name,"orderedList" description="This component takes a collection of
contacts and outputs the names into an ordered list" access="global">
<apex:attribute name="value" type="List<Contact>"/>
<ol>
<apex:repeat value,"{!value}" var="c">
<li><apex:outputText value="{!c.name}"></li>
</apex:repeat>
</ol>
</apex:component>
```

[0080] In this example, instead of the component defining the output of each element in the collection, the user may be afforded that decision by use of the body of the component tag. Table 8 shows an example of this definition, in accordance with one embodiment.

TABLE 8

```
<c:orderedList value="{!mycontacts}" var="c">
<b><apex:outputText value="{!c.name}"/></b>
</c:orderedList>
```

[0081] The component definition may thus need to utilize the componentBody tag as shown in Table 9.

TABLE 9

```
<apex:component name="orderedList" description="This component takes a collection of
contacts and outputs what the user specified in the body of the tag into an ordered list"
access="global">
<apex:attribute name="var"/>
<apex:attribute name="value" type="List<Contact>"/>
<ol>
<apex:repeat value="{!value}" var="v">
<li>
<apex:componentBody>
<apex:variable var="{!var}" value="{!v }"/>
</apex:componentBody>
</li>
</apex:repeat>
</ol>
</apex:component>
```

[0082] In this example, it may be assumed that the collection of contacts passed into the component was not ordered. In this case, the markup of the user may not change from the previous example. The component definition may change slightly with the addition of an Apex class and binding to the appropriate accessors in that class, as shown in Table 10.

TABLE 10

```
public class myComponentCon {
List<Contact> contactList;
public void setValues(List<Contact> cList) { this.contactList = cList; }
public List<Contact> getOrderedValue( ) {
List<Contact> orderedList = new List<Contact>( );
//do sorting
return orderedList;
}
}
<apex:component name="orderedList" description="This component takes a collection of
contacts and outputs what the user specified in the body of the tag into an ordered list"
access="global" controller="myComponentCon">
<apex:attribute name="var"/>
<apex:attribute name="value" type="List<Contact>" assignTo="{!values }"/>
<ol>
```

TABLE 10-continued

```
<apex:repeat value="{!orderedValue}" var="v">
<li>
<apex:componentBody>
<apex:variable var="{!var}" value="{!v}"/>
</apex:componentBody>
</li>
</apex:repeat>
</ol>
</apex:component>
```

[0083] It should be noted that this above tables illustrate code depicting exemplary implementations of the techniques described herein and should not be construed as limiting in any manner.

System Overview

[0084] FIG. **4** illustrates a block diagram of an environment **410** wherein an on-demand database service might be used. As an option, any of the previously described embodiments of the foregoing figures may or may not be implemented in the context of the environment **410**. Environment **410** may include user systems **412**, network **414**, system **416**, processor system **417**, application platform **418**, network interface **420**, tenant data storage **422**, system data storage **424**, program code **426**, and process space **428**. In other embodiments, environment **410** may not have all of the components listed and/or may have other elements instead of, or in addition to, those listed above.

[0085] Environment **410** is an environment in which an on-demand database service exists. User system **412** may be any machine or system that is used by a user to access a database user system. For example, any of user systems **412** can be a handheld computing device, a mobile phone, a laptop computer, a work station, and/or a network of computing devices. As illustrated in FIG. **4** (and in more detail in FIG. **5**) user systems **412** might interact via a network with an on-demand database service, which is system **416**.

[0086] An on-demand database service, such as system **416**, is a database system that is made available to outside users that do not need to necessarily be concerned with building and/or maintaining the database system, but instead may be available for their use when the users need the database system (e.g., on the demand of the users). Some on-demand database services may store information from one or more tenants stored into tables of a common database image to form a multi-tenant database system (MTS). Accordingly, "on-demand database service **416**" and "system **416**" will be used interchangeably herein. A database image may include one or more database objects. A relational database management system (RDMS) or the equivalent may execute storage and retrieval of information against the database object(s). Application platform **418** may be a framework that allows the applications of system **416** to run, such as the hardware and/or software, e.g., the operating system. In an embodiment, on-demand database service **416** may include an application platform **418** that enables creation, managing and executing one or more applications developed by the provider of the on-demand database service, users accessing the on-demand database service via user systems **412**, or third party application developers accessing the on-demand database service via user systems **412**.

[0087] The users of user systems **412** may differ in their respective capacities, and the capacity of a particular user system **412** might be entirely determined by permissions (permission levels) for the current user. For example, where a salesperson is using a particular user system **412** to interact with system **416**, that user system has the capacities allotted to that salesperson. However, while an administrator is using that user system to interact with system **416**, that user system has the capacities allotted to that administrator. In systems with a hierarchical role model, users at one permission level may have access to applications, data, and database information accessible by a lower permission level user, but may not have access to certain applications, database information, and data accessible by a user at a higher permission level. Thus, different users will have different capabilities with regard to accessing and modifying application and database information, depending on a user's security or permission level.

[0088] Network **414** is any network or combination of networks of devices that communicate with one another. For example, network **414** can be any one or any combination of a LAN (local area network), WAN (wide area network), telephone network, wireless network, point-to-point network, star network, token ring network, hub network, or other appropriate configuration. As the most common type of computer network in current use is a TCP/IP (Transfer Control Protocol and Internet Protocol) network, such as the global internetwork of networks often referred to as the "Internet" with a capital "I," that network will be used in many of the examples herein. However, it should be understood that the networks that the present invention might use are not so limited, although TCP/IP is a frequently implemented protocol.

[0089] User systems **412** might communicate with system **416** using TCP/IP and, at a higher network level, use other common Internet protocols to communicate, such as HTTP, FTP, AFS, WAP, etc. In an example where HTTP is used, user system **412** might include an HTTP client commonly referred to as a "browser" for sending and receiving HTTP messages to and from an HTTP server at system **416**. Such an HTTP server might be implemented as the sole network interface between system **416** and network **414**, but other techniques might be used as well or instead. In some implementations, the interface between system **416** and network **414** includes load sharing functionality, such as round-robin HTTP request distributors to balance loads and distribute incoming HTTP requests evenly over a plurality of servers. At least as for the users that are accessing that server, each of the plurality of servers has access to the MTS' data; however, other alternative configurations may be used instead.

[0090] In one embodiment, system **416**, shown in FIG. **4**, implements a web-based customer relationship management

(CRM) system. For example, in one embodiment, system **416** includes application servers configured to implement and execute CRM software applications as well as provide related data, code, forms, webpages and other information to and from user systems **412** and to store to, and retrieve from, a database system related data, objects, and Webpage content. With a multi-tenant system, data for multiple tenants may be stored in the same physical database object, however, tenant data typically is arranged so that data of one tenant is kept logically separate from that of other tenants so that one tenant does not have access to another tenant's data, unless such data is expressly shared. In certain embodiments, system **416** implements applications other than, or in addition to, a CRM application. For example, system **416** may provide tenant access to multiple hosted (standard and custom) applications, including a CRM application. User (or third party developer) applications, which may or may not include CRM, may be supported by the application platform **418**, which manages creation, storage of the applications into one or more database objects and executing of the applications in a virtual machine in the process space of the system **416**.

[0091] One arrangement for elements of system **416** is shown in FIG. **5**, including a network interface **420**, application platform **418**, tenant data storage **422** for tenant data **423**, system data storage **424** for system data accessible to system **416** and possibly multiple tenants, program code **426** for implementing various functions of system **416**, and a process space **428** for executing MTS system processes and tenant-specific processes, such as running applications as part of an application hosting service. Additional processes that may execute on system **416** include database indexing processes.

[0092] Several elements in the system shown in FIG. **4** include conventional, well-known elements that are explained only briefly here. For example, each user system **412** could include a desktop personal computer, workstation, laptop, PDA, cell phone, or any wireless access protocol (WAP) enabled device or any other computing device capable of interfacing directly or indirectly to the Internet or other network connection. User system **412** typically runs an HTTP client, e.g., a browsing program, such as Microsoft's Internet Explorer browser, Netscape's Navigator browser, Opera's browser, or a WAP-enabled browser in the case of a cell phone, PDA or other wireless device, or the like, allowing a user (e.g. subscriber of the multi-tenant database system) of user system **412** to access, process and view information, pages and applications available to it from system **416** over network **414**. Each user system **412** also typically includes one or more user interface devices, such as a keyboard, a mouse, trackball, touch pad, touch screen, pen or the like, for interacting with a graphical user interface (GUI) provided by the browser on a display (e.g. a monitor screen, LCD display, etc.) in conjunction with pages, forms, applications and other information provided by system **416** or other systems or servers. For example, the user interface device can be used to access data and applications hosted by system **416**, and to perform searches on stored data, and otherwise allow a user to interact with various GUI pages that may be presented to a user. As discussed above, embodiments are suitable for use with the Internet, which refers to a specific global internetwork of networks. However, it should be understood that other networks can be used instead of the Internet, such as an intranet, an extranet, a virtual private network (VPN), a non-TCP/IP based network, any LAN or WAN or the like.

[0093] According to one embodiment, each user system **412** and all of its components are operator configurable using applications, such as a browser, including computer code run using a central processing unit such as an Intel Pentium® processor or the like. Similarly, system **416** (and additional instances of an MTS, where more than one is present) and all of their components might be operator configurable using application(s) including computer code to run using a central processing unit such as processor system **417** of FIG. **4**, which may include an Intel Pentium® processor or the like, and/or multiple processor units. A computer program product embodiment includes a machine-readable storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the embodiments described herein. Computer code for operating and configuring system **416** to intercommunicate and to process webpages, applications and other data and media content as described herein are preferably downloaded and stored on a hard disk, but the entire program code, or portions thereof, may also be stored in any other volatile or non-volatile memory medium or device as is well known, such as a ROM or RAM, or provided on any media capable of storing program code, such as any type of rotating media including floppy disks, optical discs, digital versatile disk (DVD), compact disk (CD), microdrive, and magneto-optical disks, and magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data. Additionally, the entire program code, or portions thereof, may be transmitted and downloaded from a software source over a transmission medium, e.g., over the Internet, or from another server, as is well known, or transmitted over any other conventional network connection as is well known (e.g. extranet, VPN, LAN, etc.) using any communication medium and protocols (e.g. TCP/IP, HTTP, HTTPS, Ethernet, etc.) as are well known. It will also be appreciated that computer code for implementing embodiments of the present invention can be implemented in any programming language that can be executed on a client system and/or server or server system such as, for example, C, C++, HTML, any other markup language, Java™, JavaScript, ActiveX, any other scripting language, such as VBScript, and many other programming languages as are well known may be used. (Java™ is a trademark of Sun Microsystems, Inc.).

[0094] According to one embodiment, each system **416** is configured to provide webpages, forms, applications, data and media content to user (client) systems **412** to support the access by user systems **412** as tenants of system **416**. As such, system **416** provides security mechanisms to keep each tenant's data separate unless the data is shared. If more than one MTS is used, they may be located in close proximity to one another (e.g. in a server farm located in a single building or campus), or they may be distributed at locations remote from one another (e.g. one or more servers located in city A and one or more servers located in city B). As used herein, each MTS could include one or more logically and/or physically connected servers distributed locally or across one or more geographic locations. Additionally, the term "server" is meant to include a computer system, including processing hardware and process space(s), and an associated storage system and database application (e.g. OODBMS or RDBMS) as is well known in the art. It should also be understood that "server system" and "server" are often used interchangeably herein. Similarly, the database object described herein can be implemented as single databases, a distributed database, a collection of distributed databases, a database with redundant online or offline backups or other redundancies, etc., and might include a distributed database or storage network and associated processing intelligence.

[0095] FIG. **5** also illustrates environment **410**. However, in FIG. **5** elements of system **416** and various interconnections in an embodiment are further illustrated. FIG. **5** shows that user system **412** may include processor system **412**A, memory system **412**B, input system **412**C, and output system **412**D. FIG. **5** shows network **414** and system **416**. FIG. **5** also shows that system **416** may include tenant data storage **422**, tenant data **423**, system data storage **424**, system data **425**, User Interface (UI) **530**, Application Program Interface (API) **532**, PL/SOQL **534**, save routines **536**, application setup mechanism **538**, applications servers **500**$_1$-**500**$_N$, system process space **502**, tenant process spaces **504**, tenant management process space **510**, tenant storage area **512**, user storage **514**, and application metadata **516**. In other embodiments, environment **410** may not have the same elements as those listed above and/or may have other elements instead of, or in addition to, those listed above.

[0096] User system **412**, network **414**, system **416**, tenant data storage **422**, and system data storage **424** were discussed above in FIG. **4**. Regarding user system **412**, processor system **412**A may be any combination of one or more processors. Memory system **412**B may be any combination of one or more memory devices, short term, and/or long term memory. Input system **412**C may be any combination of input devices, such as one or more keyboards, mice, trackballs, scanners, cameras, and/or interfaces to networks. Output system **412**D may be any combination of output devices, such as one or more monitors, printers, and/or interfaces to networks. As shown by FIG. **5**, system **416** may include a network interface **420** (of FIG. **4**) implemented as a set of HTTP application servers **500**, an application platform **418**, tenant data storage **422**, and system data storage **424**. Also shown is system process space **502**, including individual tenant process spaces **504** and a tenant management process space **510**. Each application server **500** may be configured to tenant data storage **422** and the tenant data **423** therein, and system data storage **424** and the system data **425** therein to serve requests of user systems **412**. The tenant data **423** might be divided into individual tenant storage areas **512**, which can be either a physical arrangement and/or a logical arrangement of data. Within each tenant storage area **512**, user storage **514** and application metadata **516** might be similarly allocated for each user. For example, a copy of a user's most recently used (MRU) items might be stored to user storage **514**. Similarly, a copy of MRU items for an entire organization that is a tenant might be stored to tenant storage area **512**. A UI **530** provides a user interface and an API **532** provides an application programmer interface to system **416** resident processes to users and/or developers at user systems **412**. The tenant data and the system data may be stored in various databases, such as one or more Oracle™ databases.

[0097] Application platform **418** includes an application setup mechanism **538** that supports application developers' creation and management of applications, which may be saved as metadata into tenant data storage **422** by save routines **536** for execution by subscribers as one or more tenant process spaces **504** managed by tenant management process **510** for example. Invocations to such applications may be coded using PUSOQL **534** that provides a programming language style interface extension to API **532**. A detailed description of some PUSOQL language embodiments is discussed in commonly owned U.S. Provisional Patent Application 60/828,192 entitled, "PROGRAMMING LANGUAGE METHOD AND SYSTEM FOR EXTENDING APIS TO EXECUTE IN CONJUNCTION WITH DATABASE APIS," by Craig Weissman, filed Oct. 4, 2006, which is incorporated in its entirety herein for all purposes. Invocations to applications may be detected by one or more system processes, which manage retrieving application metadata **516** for the subscriber making the invocation and executing the metadata as an application in a virtual machine.

[0098] Each application server **500** may be communicably coupled to database systems, e.g., having access to system data **425** and tenant data **423**, via a different network connection. For example, one application server **500**$_1$ might be coupled via the network **414** (e.g., the Internet), another application server **500**$_{N-1}$ might be coupled via a direct network link, and another application server **500**$_N$ might be coupled by yet a different network connection. Transfer Control Protocol and Internet Protocol (TCP/IP) are typical protocols for communicating between application servers **500** and the database system. However, it will be apparent to one skilled in the art that other transport protocols may be used to optimize the system depending on the network interconnect used.

[0099] In certain embodiments, each application server **500** is configured to handle requests for any user associated with any organization that is a tenant. Because it is desirable to be able to add and remove application servers from the server pool at any time for any reason, there is preferably no server affinity for a user and/or organization to a specific application server **500**. In one embodiment, therefore, an interface system implementing a load balancing function (e.g., an F5 Big-IP load balancer) is communicably coupled between the application servers **500** and the user systems **412** to distribute requests to the application servers **500**. In one embodiment, the load balancer uses a least connections algorithm to route user requests to the application servers **500**. Other examples of load balancing algorithms, such as round robin and observed response time, also can be used. For example, in certain embodiments, three consecutive requests from the same user could hit three different application servers **500**, and three requests from different users could hit the same application server **500**. In this manner, system **416** is multi-tenant, wherein system **416** handles storage of, and access to, different objects, data and applications across disparate users and organizations.

[0100] As an example of storage, one tenant might be a company that employs a sales force where each salesperson uses system **416** to manage their sales process. Thus, a user might maintain contact data, leads data, customer follow-up data, performance data, goals and progress data, etc., all applicable to that user's personal sales process (e.g., in tenant data storage **422**). In an example of a MTS arrangement, since all of the data and the applications to access, view, modify, report, transmit, calculate, etc., can be maintained and accessed by a user system having nothing more than network access, the user can manage his or her sales efforts and cycles from any of many different user systems. For example, if a salesperson is visiting a customer and the customer has Internet access in their lobby, the salesperson can obtain critical updates as to that customer while waiting for the customer to arrive in the lobby.

[0101] While each user's data might be separate from other users' data regardless of the employers of each user, some data might be organization-wide data shared or accessible by a plurality of users or all of the users for a given organization that is a tenant. Thus, there might be some data structures managed by system **416** that are allocated at the tenant level while other data structures might be managed at the user level. Because an MTS might support multiple tenants including possible competitors, the MTS should have security protocols that keep data, applications, and application use separate. Also, because many tenants may opt for access to an MTS rather than maintain their own system, redundancy, up-time, and backup are additional functions that may be implemented in the MTS. In addition to user-specific data and tenant-specific data, system **416** might also maintain system level data usable by multiple tenants or other data. Such system level data might include industry reports, news, postings, and the like that are shamble among tenants.

[0102] In certain embodiments, user systems **412** (which may be client systems) communicate with application servers **500** to request and update system-level and tenant-level data from system **416** that may require sending one or more queries to tenant data storage **422** and/or system data storage **424**. System **416** (e.g., an application server **500** in system **416**) automatically generates one or more SQL statements (e.g., one or more SQL queries) that are designed to access the desired information. System data storage **424** may generate query plans to access the requested data from the database.

[0103] Each database can generally be viewed as a collection of objects, such as a set of logical tables, containing data fitted into predefined categories. A "table" is one representation of a data object, and may be used herein to simplify the conceptual description of objects and custom objects according to the present invention. It should be understood that "table" and "object" may be used interchangeably herein. Each table generally contains one or more data categories logically arranged as columns or fields in a viewable schema. Each row or record of a table contains an instance of data for each category defined by the fields. For example, a CRM database may include a table that describes a customer with fields for basic contact information such as name, address, phone number, fax number, etc. Another table might describe a purchase order, including fields for information such as customer, product, sale price, date, etc. In some multi-tenant database systems, standard entity tables might be provided for use by all tenants. For CRM database applications, such standard entities might include tables for Account, Contact, Lead, and Opportunity data, each containing pre-defined fields. It should be understood that the word "entity" may also be used interchangeably herein with "object" and "table".

[0104] In some multi-tenant database systems, tenants may be allowed to create and store custom objects, or they may be allowed to customize standard entities or objects, for example by creating custom fields for standard objects, including custom index fields. U.S. patent application Ser. No. 10/817,161, filed Apr. 2, 2004, entitled "CUSTOM ENTITIES AND FIELDS IN A MULTI-TENANT DATABASE SYSTEM," which is hereby incorporated herein by reference, teaches systems and methods for creating custom objects as well as customizing standard objects in a multi-tenant database system. In certain embodiments, for example, all custom entity data rows are stored in a single multi-tenant physical table, which may contain multiple logical tables per organization. It is transparent to customers that their multiple "tables" are in fact stored in one large table or that their data may be stored in the same table as the data of other customers.

[0105] It should be noted that any of the different embodiments described herein may or may not be equipped with any one or more of the features set forth in U.S. patent application Ser. No. 12/174,975, titled "SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT FOR EDITING AN ON-DEMAND DATABASE SERVICE GRAPHICAL USER INTERFACE," filed Jul. 17, 2008, which is incorporated herein by reference in its entirety for all purposes; or one or more of the following published applications: US2003/0233404, titled "OFFLINE SIMULATION OF ONLINE SESSION BETWEEN CLIENT AND SERVER," filed Nov. 4, 2002; US2004/0210909, titled "JAVA OBJECT CACHE SERVER FOR DATABASES," filed Apr. 17, 2003, now issued U.S. Pat. No. 7,209,929; US2005/0065925, titled "QUERY OPTIMIZATION IN A MULTI-TENANT DATABASE SYSTEM," filed Sep. 23, 2003: US2005/0223022, titled "CUSTOM ENTITIES AND FIELDS IN A MULTI-TENANT DATABASE SYSTEM," filed Apr. 2, 2004; US2005/0283478, titled "SOAP-BASED WEB SERVICES IN A MULTI-TENANT DATABASE SYSTEM," filed Jun. 16, 2004; and/or US2006/0206834, titled "SYSTEMS AND METHODS FOR IMPLEMENTING MULTI-APPLICATION TABS AND TAB SETS," filed Mar. 8, 2005; which are each incorporated herein by reference in their entirety for all purposes.

[0106] While the invention has been described by way of example and in terms of the specific embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, it is intended to cover various modifications and similar arrangements as would be apparent to those skilled in the art. Therefore, the scope of the appended claims should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements.

1-17. (canceled)

18. A method, comprising:

a database system storing a definition of a component for displaying a visual object within a user interface, wherein the definition includes a first value for a configurable attribute that affects a manner in which the visual object is displayed within the user interface, and wherein the definition of the component is stored with definitions of other components in a first storage portion of the database system;

the database system receiving a request to modify the configurable attribute of the definition for a first user of a plurality of users of the database system to have a second value different from the first value;

the database system storing the second value separately from the first storage portion in a second storage portion within the database system associated with the first user; and in response to a request to render a user interface for the first user that includes the component:

the database system accessing the second storage portion associated with the first user that stores the second value; and

the database system returning the second value to the component such that the visual object can be displayed according to the second value of the configurable attribute.

19. (canceled)

20. The method of claim **18**, wherein the database system is a multi-tenant database system configured to store data for different users in at least one database table.

21. The method of claim **18**, further comprising:

the database system instructing a web browser associated with the first user to display the user interface.

22. (canceled)

23. The method of claim **18**, wherein the configurable attribute defines at least one of color or size for an image of the component, or defines at least one of a font or a size for text of the component.

24. (canceled)

25. A non-transitory, computer-readable medium having computer instructions stored thereon that are executable by one or more processors to cause operations comprising:

storing a definition of a component for displaying a visual object within a user interface, wherein the visual object is displayed based on a first value for a configurable attribute included in the definition, and wherein the definition of the component is stored with definitions of other components in a first storage portion of a database system;

storing a second value in a second storage portion within the database system associated with a first user, wherein the second value is usable in lieu of the first value for the configurable attribute; and

in response to a request to render a user interface for the first user that includes the component:

accessing the second storage portion associated with the first user that stores the second value; and

returning the second value to the component such that the visual object can be displayed according to the second value of the configurable attribute.

26. The computer-readable medium of claim **25**, wherein the database system is a multi-tenant database system configured to store data for different users in at least one database table.

27. The computer-readable medium of claim **25**, wherein the operations further comprise:

in response to receiving a request to create the component, providing an interface for defining the component.

28-29. (canceled)

30. The computer-readable medium of claim **25**, wherein the operations further comprise:

instructing a web browser associated with the first user to display the user interface.

31. The computer-readable medium of claim **25**, wherein the component is reusable in a plurality of user interfaces.

32. An apparatus, comprising:

a system memory; and

one or more processors configured to execute instructions stored in the system memory to implement a multi-tenant database system that is configured to:

store a definition of a component for displaying a visual object within one or more user interfaces, wherein the definition includes an attribute having a first value that affects how the visual object is displayed within the one or more user interfaces, and wherein the definition of the component is stored with definitions of other components in a library of the multi-tenant database system;

receive a request to modify the attribute of the definition for a first tenant of a plurality of tenants of the multi-tenant database system to have a second value;

store the second value in a storage portion within the multi-tenant database system associated with the first tenant that is separate from the library; and

in response to a request to render a user interface for the first tenant that includes the component:

access the storage portion associated with the first tenant that stores the second value; and

return the second value to the component such that the visual object can be displayed according to the second value of the attribute.

33-34. (canceled)

35. The apparatus of claim **32**, further configured to:

in response to receiving a request to create the component, provide an interface for defining the component.

36. The apparatus of claim **32**, wherein the component is reusable in a plurality of user interfaces.

37. The apparatus of claim **32**, wherein the attribute defines at least one of color or size for an image of the component or defines at least one of a font or a size for text of the component.

38. The method of claim **18**, wherein the component is reusable in a plurality of user interfaces.

39. The method of claim **18**, further comprising:

the database system displaying a textual description of the component provided by a user of the plurality of users of the database system.

40. The method of claim **18**, further comprising:

in response to receiving a request to create the component, providing an interface for defining the component.

41. The computer-readable medium of claim **25**, wherein the operations further comprise:

displaying a textual description of the component supplied by a user of a plurality of users of the database system.

42. The apparatus of claim **32**, wherein the multi-tenant database system is configured to store data for different tenants in at least one database table.

43. The apparatus of claim **32**, wherein the multi-tenant database system is further configured to:

instruct a web browser associated with the first tenant to display the user interface.

44. The apparatus of claim **32**, wherein the multi-tenant database system is further configured to:

display a textual description of the component supplied by a tenant of the multi-tenant database system.

* * * * *