

VcDrawingNode **504**. VcLogicNode **510** is an abstract base class for nodes which defines relationships between shape nodes and the user.

[**0175**] **FIGS. 15 and 16** show exemplary wormhole usages. **FIG. 15** shows four wormholes: a portfolio-risk-management wormhole **700**, a market-data wormhole **710**, an investments-under-consideration wormhole **720** and a first-call-analyst-recommendation wormhole **730**. The portfolio-risk-management wormhole **700**, in turn shows three detailed wormholes **702**, **704** and **706** displaying a third level view of the scene and a chart **708**. Each of the three detailed wormholes **702**, **704** and **706** shows financial performance associated with three separate funds or portfolios. Moreover, views of a given scene arising from one wormhole representing one fund or portfolio may be different from views of the same scene arising from another wormhole representing a different fund. Thus, for example, by drilling down the portfolio risk management wormhole **700**, through one of the funds **702**, **704** or **706**, and drilling down to a company in a particular portfolio, context information is accumulated with every drill-down so that the resulting view of the company is generated in relationship to the specific fund or portfolio. The information may include the quantity of the company's stock held by the fund, and the duration of ownership of the company's stock, among others.

[**0176**] The scene being presented in each wormhole in **FIG. 16** is parameterized in company_ID in a manner analogous to an argument to a function. In this case, the scene itself has an argument that specifies what company the user is looking at and the scene is accordingly customized. Thus, when the user looks through any of these wormholes, the scene looks different because it takes on the identity of the specific wormhole being viewed by the user.

[**0177**] As discussed above, the system provides dynamic views of data without programming expertise. Users are thus moved closer to the data so that application development time is reduced. User interfaces may be created quickly and easily for information rich databases and for applications such as data warehousing and decision support. Further, limitations inherent in conventional forms-based or report-based applications are avoided.

[**0178**] Moreover, the techniques described here may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs executing on programmable computers that each includes a processor, a storage medium readable by the processor (including volatile and nonvolatile memory and/or storage elements), and suitable input and output devices. Program code is applied to data entered using an input device to perform the functions described and to generate output information. The output information is applied to one or more output devices.

[**0179**] Each program is preferably implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

[**0180**] Each such computer program is preferably stored on a storage medium or device (e.g., CD-ROM, hard disk or

magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described. The system also may be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

[**0181**] Other embodiments are within the scope of the following claims.

What is claimed is:

1. A computer-implemented property entry sheet for contextually assigning a property of an object associated with an application, comprising:

an attribute name section adapted to receive an identification of the property; and

a property input section adapted to receive a functional expression for the property identified by the attribute name section, the functional expression being referencable at run-time as a data value.

2. The property entry sheet of claim 1, wherein the functional expression includes one or more of the following:

a function;

an operator;

a database column name;

a variable; and

a constant.

3. The property entry sheet of claim 1, further comprising:

an attribute name section adapted to receive an identification of the property; and

a property input section adapted to receive a static data value for the property identified by the attribute name section.

4. The property entry sheet of claim 1, wherein the object has a plurality of properties and wherein the attribute name section and the property input section of each property form a name-value pair for each property.

5. The property entry sheet of claim 1, wherein the functional expression is parsed to generate a function which is stored as a run-time value.

6. The property entry sheet of claim 5, further comprising byte code associated with the function.

7. The property entry sheet of claim 1, wherein the function is cloned and stored as a design time value if the function is a constant.

8. The property entry sheet of claim 1, wherein an error message is displayed if the expression is invalid.

9. The property entry sheet of claim 1, wherein an existing byte code execution image is invalidated.

10. The property entry sheet of claim 9, wherein new byte code is generated to replace the existing byte code execution image.

11. A method for editing a computer-implemented object, the object having a property entry sheet for assigning a property of an object associated with an application, comprising: