

objects in the target scene. This also eliminates the effects of the incremental zoom factor as well as frees up memory used by objects in the first scene. As the user navigates a world and interacts with objects using the mouse, notification events are generated which may trigger behavior. This behavior is determined by the developer at design time and consists of the following two optional steps (in order): setting the value of scene and/or global parameters, and executing an action.

Wormholes provide control over the values of scene parameters through a SceneParameters object listed in the Object Inspector. The SceneParameters object is a child object of the wormhole and contains a property for each parameter. If the target scene is not parameterized, the SceneParameters object may not be displayed in the Object Inspector. The values contained by the object are used to set the scene's properties at runtime.

Three types of parameters drive runtime values in queries and also monitor and control various user context variables: global parameters, scene parameters, and query parameters. Global parameters are exposed in the runtime user as properties to be set or monitored by the container application. Further, built-in global parameters may be used: UserX (tracks or sets the user's current horizontal offset from center in inches); UserY (tracks or sets the user's current vertical offset from center in inches); UserZoom (tracks or sets the user's current zoom (magnification) level); and UserClass (contains the name of the user's profile class).

Query parameters are set by the data source before the query is executed. Scene parameters may be set by wormholes or by event actions. They have the following attributes:

Name—parameter identifier.

Data type—value type.

Description—available for internal documentation.

Default value—value used if not set by a wormhole's

SceneParameters child object or by an event action. If no default value is available, the scene cannot be viewed.

User classes may be used for customizing the behavior or appearance of the virtual world based on user identity. For example, a wormhole to sales forecasts may only be visible to sales personnel and executive staff members, or a hospital floor plan layout may highlight vacant beds for an administrator versus cardiac patient beds for a cardiologist. The default user class in all new worlds is "Anonymous."

The current user class for a world is stored in the UserClass global parameter. A property value or event method can be based on the current user class by use of the IsUser(class\_name) property function which returns a true or false value depending on whether the UserClass variable is a member of the class\_name user class. Standard boolean expressions can also contain the UserClass parameter for direct comparison or display.

User classes can be subclassed in order to refine a class further. A user subclass of one class can also be a subclass of another class, thus resulting in a flexible, multiple-inheritance hierarchy of user classes. When a user class is deleted by the user and it exists as a subclass of more than one other user class, the user may be asked whether or not to delete all occurrences of the user class or just to delete the selected instance.

Turning now to FIG. 14, an object model of a wormhole is shown in more detail. A VcScene class 500 defines the drawing layer (canvas) for all graphical objects displayed in viewing area. VcScene 500 provides a set of parameters which may be referenced by the properties of nodes con-

tained within the scene. Before the scene can be rendered, each parameter must be set similar to the way arguments to a function must be defined before the function is called. A wormhole settings node then supplies the calculated settings for each scene parameter when the scene is viewed through the wormhole.

The VcScene class 500 has properties that are a member of a VcParameter class 502 and a list of nodes derived from a VcDrawingNode base class 504. VcParameter 502 stores information about a scene parameter, including its name and data type, while VcDrawingNode 504 is an abstract base class for all shape and logic nodes which represent the contents of a scene. VcDrawingNode 504 contains properties which may be constant or calculated. Calculated properties may depend on one or more scene parameters.

The VcDrawingNode abstract base class 504 in turn is inherited by a VcShapeNode 506, which is further inherited by VcWormholeNode 508. VcShapeNode 506 is abstract base class for all visible node types. Derived classes of VcShapeNode 506 implement the specific attributes and behavior of each type of shape. VcWormholeNode 508 is a class of shape node in a scene which links to another scene. It contains a pointer to a settings node for setting the values of all scene parameters before the scene is rendered.

VcWormholeNode 508 has a m\_sceneParamSettings property which is a member of the VcWormholeSettingsNode 512 class. VcWormholeSettingsNode 512 is a class of logic node holding wormhole-specific settings for each parameter in the connecting scene. These settings are evaluated and passed to the scene before it is rendered in the wormhole.

VcWormholeSettingsNode 512 is also derived from VcLogicNode 510, which in turn is derived from the VcDrawingNode 504. VcLogicNode 510 is an abstract base class for nodes which defines relationships between shape nodes and the user.

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.

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.