

Introduction

There are numerous tools through which application developers can develop user interfaces for querying, updating, and browsing data that is stored in a commercial database system. The following are some of the options available:

- A report writer, such as Crystal Reports
- A spread sheet tool, such as Excel
- A business intelligence tool, such as Business Objects
- A graphics package, such as Visio
- A web tool, such as Front Page
- A fourth generation language, such as PowerBuilder

In addition, application developers often want to put a 'face' (presentation layer) onto applications that they construct. Again, there are a myriad of possible tools.

Most of these tools present data from an ODBC data source onto the user's screen. In general, the designer of the user interface has the standard mechanisms for data presentation. These typically include:

- The location of the data element on the screen (X and Y location)
- The color of the data element
- The rendering of the data element
- The size of the data element

However, the overwhelming majority of these tools have the following three restrictions:

- a) They support a single (or at most a few) paradigms for presenting data. For example, report writers allow one to present data as a database report. However, they do not support other mechanisms for data presentation
- b) They support a single mechanism for "drilling down" into database objects. For example, most fourth generation languages support master/detail forms. By clicking on the master form, a user can bring up the detail form and get more information. Similarly, business intelligence tools allow one to drill into statistical summary data and get more detailed summary data. The detail is obtained by "breaking out" a previously aggregated dimension of data.
- c) They support a single mechanism for navigating to desired data. For example, in an HTML system, one can click on a URL (hyperlink) and be transported to a new web page. Hence, navigation is supported from web page to web page.

It is obvious that there is plenty of CPU horsepower on user's desktops to enable much more sophisticated user interfaces. Moreover, both Windows and Java environments has the necessary graphics libraries to facilitate better displays.

The purpose of Visionary is to set a new level in data presentation.

Visionary can be used both in retrieval-oriented interfaces such as data portals as well as in update-oriented transaction systems. In addition, a Visionary application is incredibly easy for end users, and essentially no training is required. Moreover, for

the application developer, Visionary is an extremely powerful rapid application development (RAD) tool.

In the remainder of this white paper, we explore the history of the Visionary product, then we turn to discussing the concepts on which it is based. Following this discussion, we consider Visionary Studio, the design-time application specification environment. Lastly, we treat the three-tier architecture of Visionary and close with some of the future directions we are contemplating.

Since "a picture is worth 100 words", Visionary screenshots are used freely to illustrate the textual presentation.

1 History

Visionary was originally written as the Tioga/DataSplash system at the University of California, Berkeley in the mid 1990's by a research team under the direction of Michael Stonebraker [AIKE95, AIKE96, WOOD01]. When Illustra was founded to commercialize the Postgres Object-Relational DBMS, it elected to commercialize Tioga as well, and engaged in a complete rewrite of the academic prototype. In 1996, Illustra was acquired by Informix, and Stonebraker became the Chief Technology Officer (CTO) of Informix. A development group in the office of the CTO performed a second rewrite to improve performance and install ease-of-use wizards. A final significant modification converted Visionary into a 3-tier web-oriented product.

In 2001, Informix was acquired by IBM, which inherited the Visionary system. Because IBM didn't want to compete with its partners who were tool vendors, it elected to sell Visionary to Rocket Software, a 200+ person software company in Newton, Mass. Rocket is committed to selling, supporting and further enhancing Visionary.

2 Visionary Concepts

In this section we begin with the basic Visionary nomenclature, and then turn to the three constructs that make Visionary very unique.

2.1 Visionary Nomenclature

Visionary is a very sophisticated presentation system for ODBC and JDBC data sources. It presents the result of a query on an arbitrary-sized **canvas**, which we term a **scene**. The end user controls a **viewer**, which is his terminal screen, and it is positioned a certain **elevation** above the scene. This user can freely **pan** in any directions. In this way, as much information as you want can be rendered on a scene. Moreover, an arbitrary number of scenes can be collected together into a Visionary application, which we call a **world**. Hence, an arbitrary number of scenes can appear in a Visionary world, each displaying disparate data. Every scene renders the result of one of more ODBC queries, as we will presently see.

With this background, we turn now to the three unique Visionary constructs.

2.2 Visionary Multi-modality

The first unique feature of Visionary is that it supports a wide variety of data templates, some 20 in all. These include geographic maps, pie charts, scatter plots, organization charts, and horizon charts. Hence, the user can see data using whatever visual motif that best conveys his information. As such, Visionary is a **multi-modal** system, and is not constrained to one (or a few) modalities.

In addition, a Visionary world can **cascade** modalities. For example, employee data can be displayed on a geographic map, with information on each employee occurring at the geographic location of his home address. This display can use the Visionary geographic map motif. Then, employee salary history could be plated as a graph at this location, thereby nesting a graph inside a map. Figure 1 illustrates a cascade of display motifs.

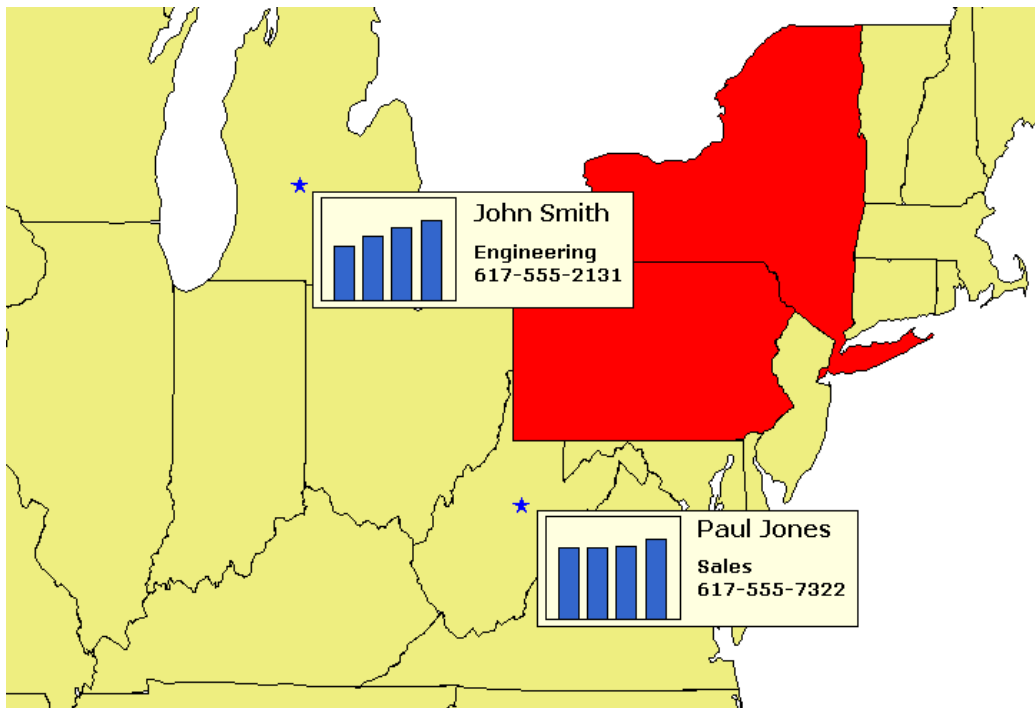


Figure 1: A Sample Visionary Screen

In addition, there are multiple query wizards, which help a user specify a query. The most elegant ones resemble Query-by-Example (QBE) [ZLOO77]. Again, the user is not restricted to a single way of presenting a query.

2.3 Semantic Zoom

In Visionary, the viewer can be **zoomed** into or away from a scene. As one would expect, zooming in shows a smaller area of the scene, while zooming out does the converse. This effect is achieved by defining the **elevation** of the viewer above the scene. Zooming simply changes the elevation of the viewer, and the notion of **physical zoom** is readily achieved. Physical zoom is a feature of some graphics tools.

Visionary revolutionizes this concept, by providing **semantic zoom**. Specifically, a Visionary scene can have an arbitrary number of **levels**, as noted in Figure 2. Here we see an elevation diagram of a scene showing three levels. Level 1 is valid above elevation E1, level 2 is valid between elevations E1 and E2, and so forth.

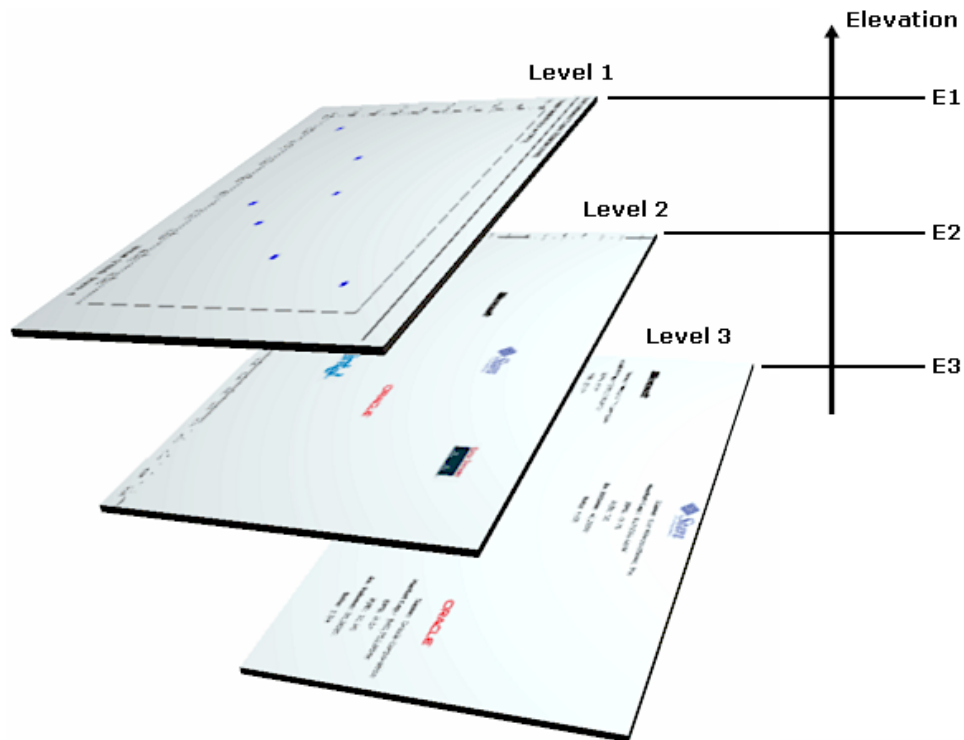


Figure 2: A Visionary Elevation Diagram

Each level of a scene is associated with an SQL query and a presentation motif. Hence, as the viewer is moved toward the scene from a high elevation, the objects of level 1 are visible until the viewer reaches elevation X1. Then, the visible objects become the ones on level 2, and so forth. Using the technology of levels, an employee can be a dot on a map at the location of his home address. Zooming in can result in seeing his picture, while further zooming could make his salary record visible.

Note clearly that the queries at the various levels are not restricted to a single database. Hence, the visible screen objects that result from a zoom can come from different data sources.

Also, notice that a Visionary user is presented with a collection of scenes in his viewer. He can pan (navigate) around the scenes to find information of interest. Then, he can zoom to get increasing amounts of detail. As such, the end user interface is intuitive and easy to use. Hence, essentially no training is required in order to operate a Visionary application.

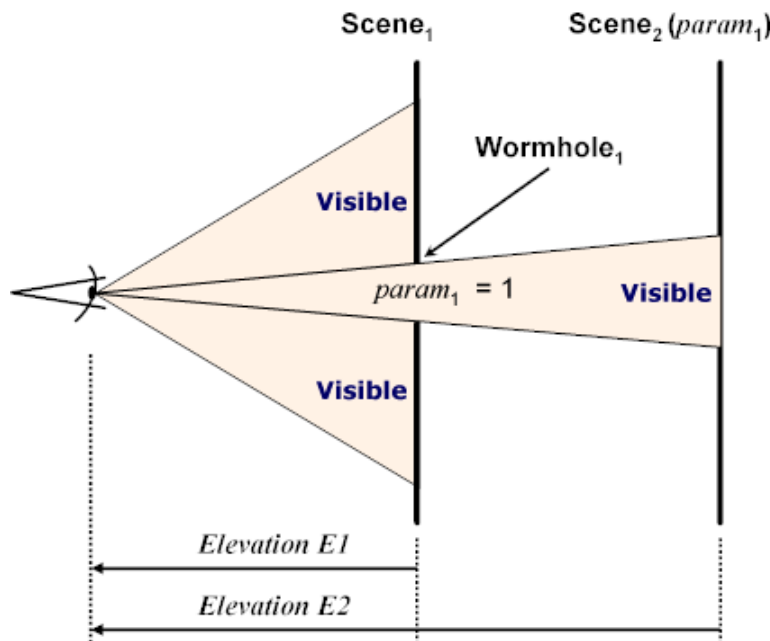
Lastly, all objects in a Visionary scene are **active**; hence the end user can move the mouse to an object and click on it. The world designer can specify the action that is taken as a result of this event. Actions can include launching a new application, jumping (**teleporting**) to a different scene, sending e-mail, and updating the database.

Because of this feature, Visionary can be used for update-oriented applications as well as browsing ones. Also note that Visionary can put a sophisticated visual face on an existing legacy application. One merely defines the routines in the legacy app to be activated by Visionary screen events.

2.4 Multi-lateralness and Visual Navigation

In the previous section we presented three mechanisms through which an end user could navigate to desired data, namely panning, semantic zoom and teleporting. In this section we specify a fourth mechanism, which we term **visual navigation**. As such, Visionary is not limited to a single navigation system, but is **multi-lateral**, and can be programmed to utilize any one of these four navigation paradigms.

The fourth navigation mechanism utilizes the concept of **wormholes**. Any scene can have one or more holes in it, through which other scenes, located behind the current one, are visible. The viewer is merely a greater distance above the partially hidden scene, as noted in Figure 3.



Here we see a viewer which is at elevation E1 above scene 1 and E2 above scene 2, and the geometry shows what is visible in each of the scenes. Since scenes can be arbitrarily nested, the end user can see information on perhaps many scenes at the same time. He is merely closer to some scenes than others.

Figure 3: The Mathematics of Wormholes

Visually, the end user can be presented with much (or perhaps all) of the information that is available in the world. With well designed icons and levels, the end user can see at a glance everything that he might be interested in. Then, he can zoom into information of interest. Also, he can move his viewer through a wormhole. This will refine the scope of what is on the screen by removing from view the topmost scene. This ability to visually move through wormholes from scene to scene will be termed **visual navigation**. A screen shot of the nesting of scenes is shown in Figure 4.

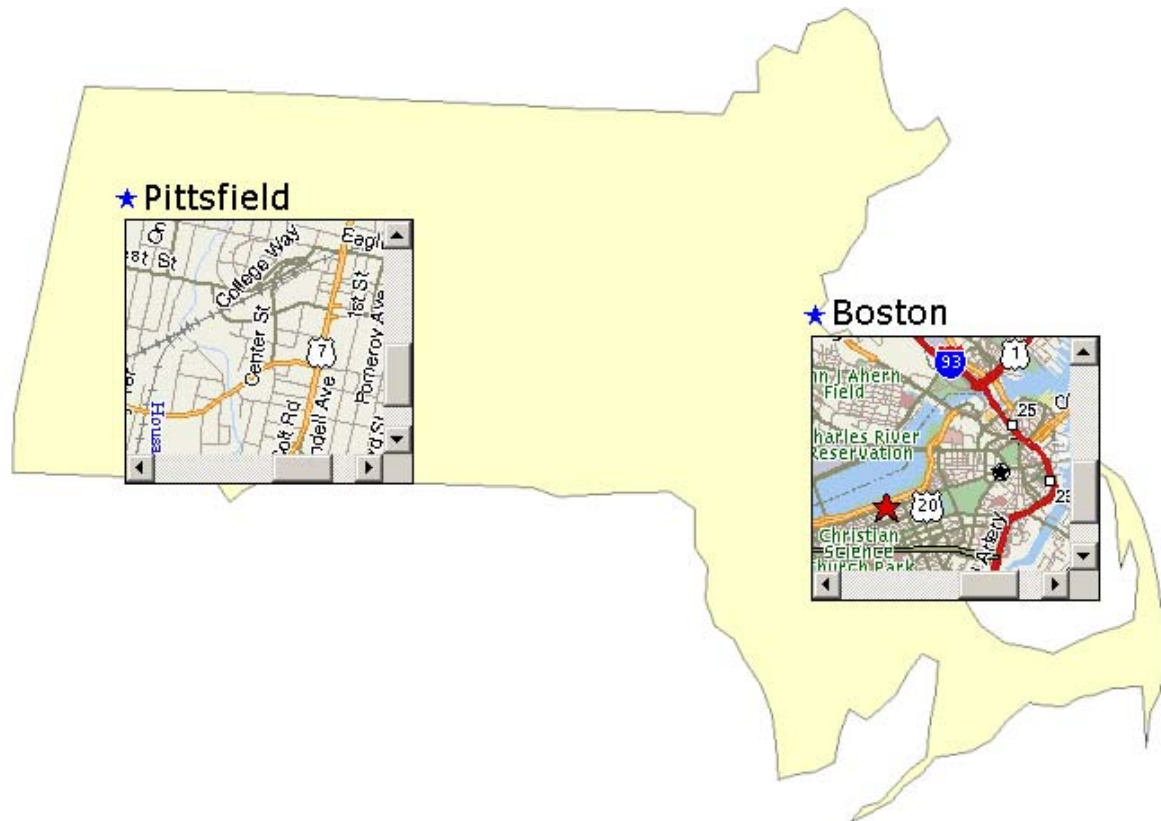


Figure 4: A Screenshot of Multiple Wormholes

Scenes linked by wormholes may also contain parameters that control their content. Such parameters appear as properties of the wormhole that links them to the parent scene. Using this mechanism, it is possible to place multiple copies of a given scene onto the parent scene, each of which appear differently when rendered. For example, a geographic map may place a wormhole within every state, linking to a sales report scene. Each wormhole drives the scene with a distinct state name, thereby producing individual sales reports. Such an example is presented in figure 5.

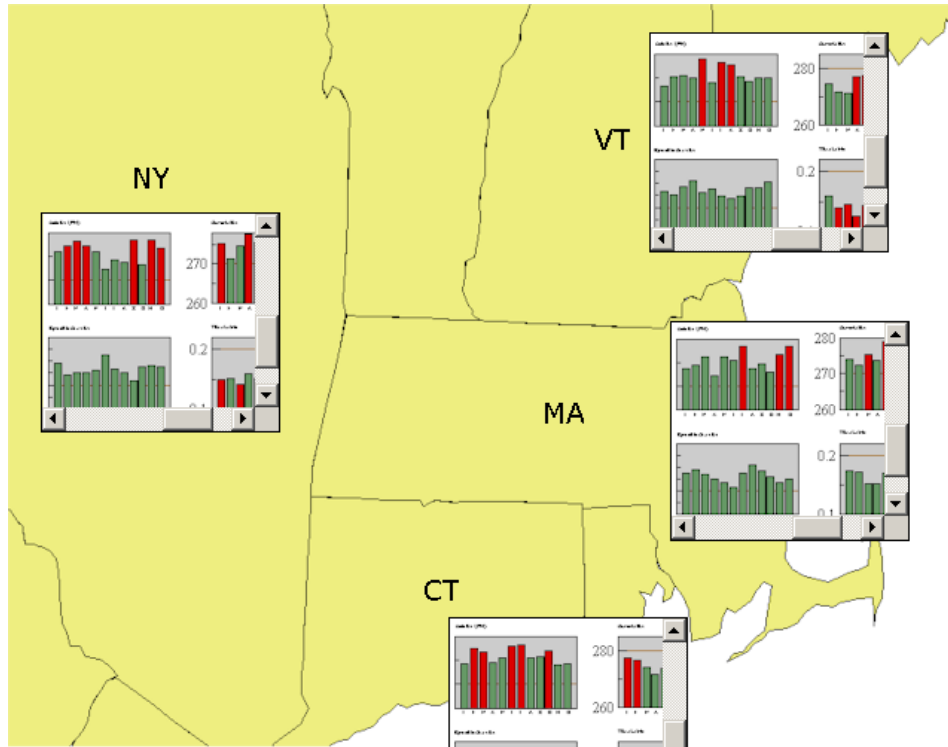


Figure 5: Wormholes driving independent copies of the same scene

In summary, Visionary supports a very easy to use and intuitive end-user interface that consists of:

- Panning around scenes
- Zooming
- Moving through wormholes
- Clicking on objects to take actions

Using these mechanisms the end-user can navigate through a visually rich information landscape to find and perhaps update database objects of interest.

3 Visionary Studio

To build a Visionary world, the application designer must specify:

- The scenes and levels in the world
- The layouts at each level
- The query required for each layout
- The data template for each layout
- The static data (trim) used to decorate the level

This specification is accomplished using the Visionary design-time environment, called **Visionary Studio**.

Studio is a drag-and-drop menu-oriented system that uses **wizards** to help specify the more complex tasks, such as SQL queries and data templates. A screenshot of Visionary Studio is shown in Figure 6.

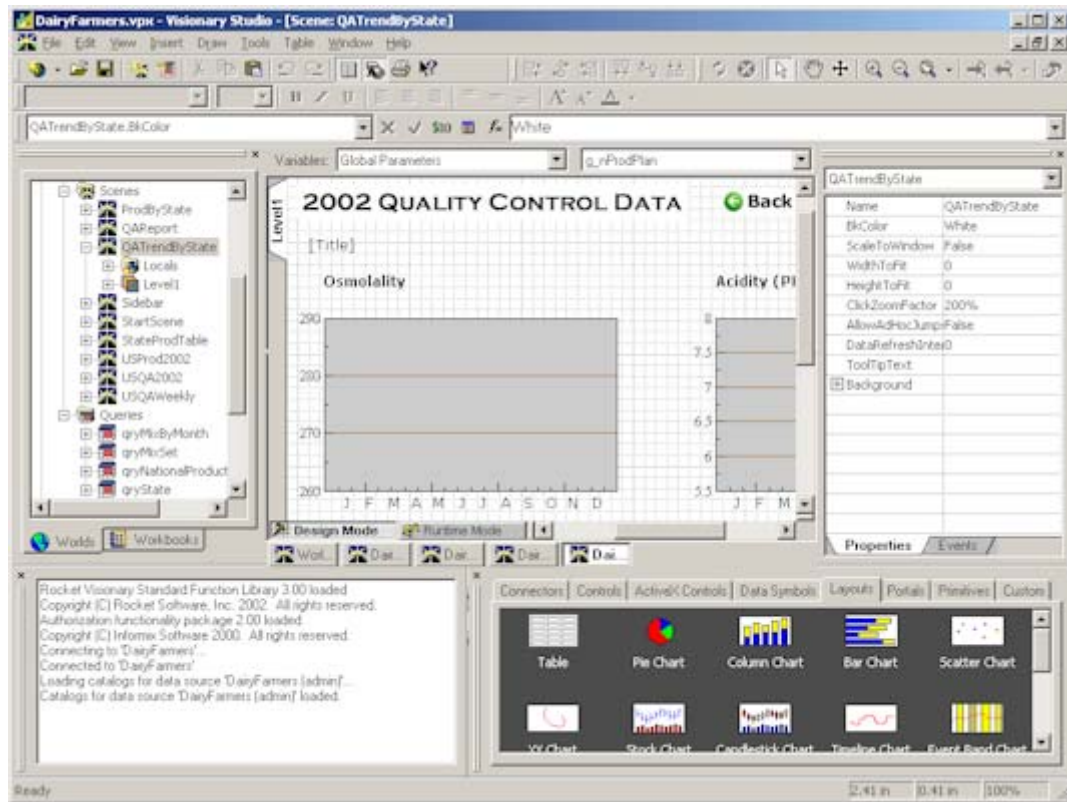


Figure 6: A screenshot of Visionary Studio

Here, we see the palette manager on the lower right that allows the designer to add layouts to a given level in a scene (graphs, charts etc.). Also shown is the world manager (left edge), through which the designer can navigate around the various scenes and the queries that drive them.

We close this section with three final observations. First, Visionary Studio supports the concept of a **workbook**, which represents the schema within a given database, providing a subset or the entire collection of tables, views, procedures and functions contained therein.

Workbooks are created and modified using wizards. Once loaded, the workbook can be saved to disk and serve as a tailored view of the database(s) that serve the Visionary world. As table schemas are provided by the workbook, they allow developers to engage Visionary's query building wizards and query diagrams, even when disconnected from the source database. Workbooks also reduce the development load on the database, particularly useful when developing or testing against production data.

A Visionary world can utilize any number of workbooks. In this way a Visionary world can integrate data from any number of ODBC data sources.

Second, there is no concept of compiling a Visionary world, and the designer can freely switch back and forth between design mode and run mode. Hence, the designer can make a change to a world, and then immediately switch to run mode to see the visual impact of his change. Instantaneous turnaround for changes supports very high designer productivity.

Third, Visionary developers program scenes and layouts by dragging items from the palette manager (toolbox) and setting their properties to control their appearance and behavior. Visionary supports 'dynamic object properties', whereby property values can be expressed as functions or expressions, rather than static values. Leveraging this difference, Visionary developers can apply property expressions that change the appearance and behavior of items that represent individual data rows in a results set. For example, each bar in a bar chart can be colored according to its magnitude and present a 'tool tip' text message that is constructed from several other columns in the query. In fact, any number of columns can directly or indirectly drive the properties of a given layout element, on a row by row basis.

Lastly, Visionary supports the notion of **publishing** worlds, whereby they are packaged (frozen) for subsequent execution by an end user.

4 The Visionary Architecture

The Visionary 3-tier architecture is shown in Figure 6. Here we see a run-time client that can either be a web browser, a stand-alone Windows program or any other object that can accept an ActiveX control. For non-Microsoft environments, we are rewriting the client piece as a Java applet, and this component will be available during Q2/03. The client component is responsible for rendering of Visionary objects on the screen and event handling associated with Visionary events.

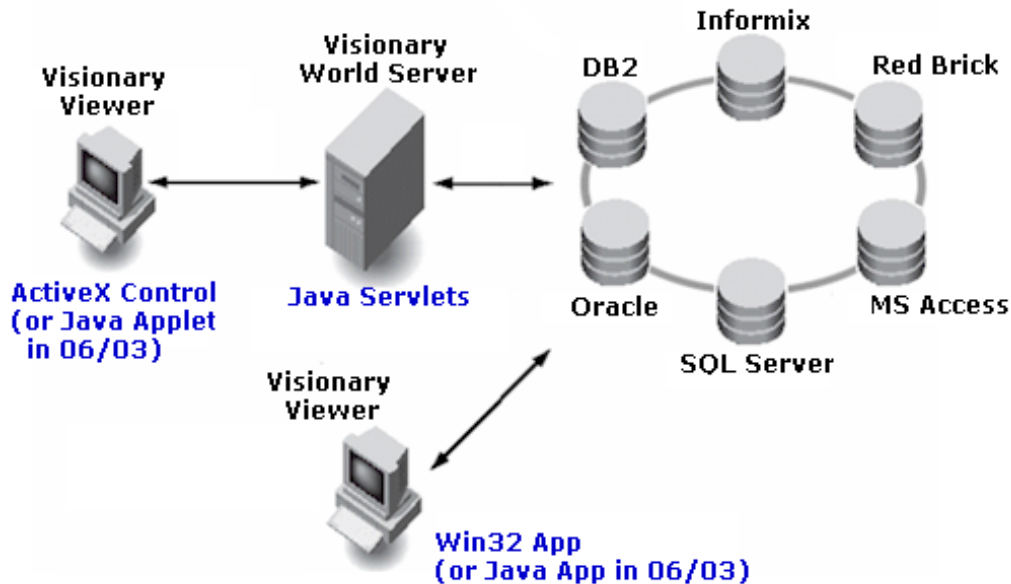


Figure 7: The Visionary Run-time Architecture

In the middle tier, all of the Visionary logic runs as a collection of J2EE components. These can be added to any J2EE compliant application server, such as Tomcat or BEA's WebLogic. It is also possible to run Visionary in "fat client" mode, in which this logic runs in the client tier and communicates directly with the database server(s).

Lastly, the server level is any ODBC or JDBC compliant DBMS. At the present time, Rocket has tested the drivers for Informix, DB2, and SQL Server, Oracle, Red Brick and MS Access. Several additional ODBC gateways, including one to SAP, are planned. Moreover, Rocket is exploring an XQuery interface for Visionary.

Visionary Studio is presently a stand-alone Windows program written in C++. We are planning to write a Java version of Studio, so as to provide a complete Microsoft-independent version of Visionary, to complement the Microsoft-dependent one.

5 Future Plans

There are major enhancements that we plan for the Visionary products in addition to the ones discussed so far (Java applet version of the client, Java version of Studio, and additional ODBC gateways).

First, the reader can notice that there is a high-level powerful retrieval model built into Visionary; however, there is no corresponding update model. Hence, an application designer who wants to implement a transactional system must code his updates using direct SQL or as Java programs with JDBC calls. Obviously, a higher-level interface would help programmer productivity.

Second, the current version of Visionary Studio is oriented toward professional programmers. We propose to rework the design-time interface so it is much more forgiving. The hope is that Visionary can be made accessible to non-programming professionals, such as business analysts.

In addition, it is easy for an end-user to "get lost" in a Visionary world. We plan to implement a "thumbnail world" in a corner of the screen, which will give the user a cue as to where he is in the current world.

Furthermore, Visionary would be a more powerful integration tool if foreign presentation systems could run in a Visionary wormhole. With this capability, Visionary could be used to present both Visionary information as well as information from foreign (non-ODBC) application systems. We plan to explore an implementation of user-defined scenes, which would accomplish this objective.

Another extendibility idea is user-defined presentation wizards. Using this construct, an application designer could add a new presentation motif, if he did not like the ones built into Visionary.

Some users have asked us for a 3-D version of Visionary. One user who constructs large power plants would like to use 3-D Visionary to guide repair personnel to devices that need fixing. As the repair person approached a device, semantic zoom would be used to give him increasing levels of detail about the device.

Not only would 3-D be more CPU intensive, but also Visionary would have to be extended to allow the display to change as one "walked around" a 3-D Visionary object. We plan to explore what could be feasible in this area.

One of the hardest design problems in a Visionary world is getting semantic zoom to "look right". Visionary designers "fiddle" with the location of levels and with font sizes in an effort to get a pleasing look to the screen while zooming. Right now this graphical design is a manual human endeavor. However, we expect to partially automate this process using the principle of constant information density [WOOD98]. Specifically, the problem with a pleasing zoom is that the density of pixels on the screen may fluctuate greatly, leading to visual discomfort. However, a Visionary wizard could adjust the breakpoint between levels to minimize the variation in pixel density. This is much harder than it looks because object density varies across different screen locations. For example, pixel density on a geographic map is lower in Montana than in New York City. Nevertheless, we believe that at least partial automation is possible.

In a large Visionary world, an end user may wish to traverse a substantial distance. For example, if he wishes to move from Montana to New York, then he really doesn't want to see all the details in between. Rather, Visionary should automatically move to a higher elevation when long distances are traversed. Then, when the user slows down, Visionary should gradually return to the original elevation. This notion of "rapid transit" would provide better response time to long distance panning.

Lastly, it would be a great idea to provide "bookmarks" so that a user could indicate that he wished to return to a specific location at a later time.

6 Conclusions

This white paper has described Visionary, a revolutionary presentation system that allows an application to be easily "vis-enabled". Visionary is a multi-modal, multi-lateral system that supports semantic zoom. As such, the end user can be presented with a visually rich environment to navigate to desired information. Moreover, the Visionary world designer is supported by Visionary Studio, which gives him a powerful RAD environment. Most any Visionary world can be built in one day, and we like to think of Visionary as an "app in a day" product.

Visionary is a web-enabled, three tier system. A Microsoft version is available now, and a "Java-only" version will be available by mid-year. Rocket Software is committed to enhancing this product, and we indicated a variety of future enhancements. In addition, QMF for Windows, the ad-hoc query tool for DB2, is actually a Rocket product, although it is sold by IBM under a royalty license. Hence, many of the Visionary features will be added to QMF over time. In the next release, the query and presentation wizards will appear in QMF, making it a tool that supports much more than "rows and columns".

7 References

- [AIKE95] Alexander Aiken, Jolly Chen, Mark Lin, Mybrid Spalding, Michael Stonebraker, Allison Woodruff: The Tioga-2 Database Visualization Environment. Workshop on Database Issues for Data Visualization 1995: 181-207
- [AIKE96] Alexander Aiken, Jolly Chen, Michael Stonebraker, Allison Woodruff: Tioga-2: A Direct Manipulation Database Visualization Environment. ICDE 1996: 208-217
- [WOOD98] A. Woodruff, J. Landay, and M. Stonebraker. "Constant Information Density in Zoomable Interfaces." Proc. Advanced Vis. Interfaces '98, L'Aquila, Italy, May 1998, pp. 57-65.
- [WOOD01] A. Woodruff, C. Olston, A. Aiken, M. Chu, V. Ercegovic, M. Lin, M. Spalding, and M. Stonebraker. "DataSplash: A Direct Manipulation Environment for Programming Semantic Zoom Visualizations of Tabular Data." JVLC(12)5:551-571, Oct. 2001
- [ZLOO77] Moshé M. Zloof: Query-by-Example: A Database Language. IBM Systems Journal 16(4): 324-343 (1977)