Peter D. Kochevar
Leonard R. Wanger

Tecate: A Software Platform for Browsing and Visualizing Data from Networked Data Sources

Tecate is a new infrastructure on which applications can be constructed that allow end users to browse for and then visualize data within networked data sources. This software platform capitalizes on the architectural strengths of current scientific visualization systems, network browsers like Netscape, database management system front ends, and virtual reality systems. Applications layered on top of Tecate are able to browse for information in databases managed by database management systems and for information contained in the World Wide Web. In addition, Tecate dynamically crafts user interfaces and interactive visualizations of selected data sets with the aid of an intelligent system. This system automatically maps many kinds of data sets into a virtual world that can be explored directly by end users. In describing these virtual worlds, Tecate uses an interpretive language that is also capable of performing arbitrary computations and mediating communications among different processes.

All people share the need to find and assimilate information. Data from which information is created is increasingly available electronically, and that data is becoming more and more accessible with the proliferation of computer networks. Therefore, the world is quickly becoming abstracted as a collection of networked data spaces, where a data space is a data source or repository whose access is controlled by means of a well-defined software interface. Some examples of data spaces are a database managed by a database management system, the World Wide Web (WWW or Web), and any data object that resides in a computer's main memory and whose components are accessible through the object's methods.

The need to locate data and then map it to a form that is readily understood lies at the core of learning, conducting commerce, and being entertained. To address this need, interactive tools are required for exploring data spaces. These tools should allow any end user to browse the contents of data spaces and to inspect, measure, compare, and identify patterns in selected data sets. Combining both tasks into one tool is both elegant and utile in that end users need to learn only one system to seamlessly switch back and forth between browsing for data and assimilating it. Before such applications can be constructed, however, a firm foundation must be defined that provides an interface to data spaces, helps map data into a visual representation, and manages user interactions with elements in the visualizations.

This paper describes one such software platform, called Tecate, which has been implemented as a research prototype to help understand the issues involved in exploring data spaces. With Tecate, the emphasis has been on developing the tools needed to build end-to-end applications. Such applications can access data spaces, automatically create virtual worlds that represent data found in data spaces, and give end users the ability to navigate and interact with those worlds as the mechanism for exploring data spaces. Because of this emphasis, Tecate's development concentrated on understanding what system components are needed to create end-to-end applications and how those components interact rather than on the functionality of individual components. As a consequence,

the tools provided by Tecate can be used to build applications of only modest capabilities.

Historically, Tecate grew out of the Sequoia 2000 project, which was initiated jointly by Digital Equipment Corporation and the University of California in 1991. The primary purpose of the Sequoia 2000 project was to develop information systems that would allow earth scientists to better study global environmental change. Sequoia 2000 participants needed to browse for data sets on which to test scientific hypotheses and then to interactively visualize the data sets once found. The data can be quite varied in content and structure, ranging from text and images to time-varying, multidimensional, gridded or polyhedral data sets. Such data may stream from many different sources, e.g., databases managed by a database management system, a running simulation of some physical process, or the WWW. Therefore, a tool was required that could interface to any such source. To be of maximum use, though, the tool had to be easy to use so that the scientists themselves could make sophisticated data queries and then experiment with the query results using a wide variety of data visualization techniques.

Generalizing from its Sequoia 2000 roots, the design of Tecate is intended to achieve four goals:

1. Interface to general data spaces wherever they may reside.
2. Saliently visualize most kinds of data, e.g., scientific data and the listings in a telephone book.
3. Dynamically craft user interfaces and interactive visualizations based on what data is selected, who is doing the visualizing, and why the user is exploring the data.
4. Allow end users to interact with elements in visualizations as a means to query data spaces, to explore alternate ways of presenting information, and to make annotations.

There are systems available today that have some of these capabilities, but no one system possesses all four. Data visualization systems such as AVS, Khoros, or Data Explorer are capable of visualizing scientific data; however, they are poor at interfacing to general data spaces, they provide only limited interactivity within visualizations themselves, and they require visualizations to be crafted by hand by knowledgeable end users.[1,2,3] Network browsers such as Netscape are good at fetching data from certain types of data spaces but are limited in the variety of data they can directly visualize without having to rely on external viewer programs. Moreover, most network browsers offer a restricted type of interactivity where only hyperlinks can be followed and text can be submitted through forms. Finally, front ends to database management systems provide elaborate querying mechanisms for selecting data from a database, but they lack a sophisticated means for visualizing and further exploring query results.

The Tecate architecture borrows from that of visualization systems, network browsers, and database management systems as well as from virtual reality systems like Alice and the Minimal Reality Toolkit/Object Modeling Language (MR/OML).[4,5] One major contribution of the Tecate system is that it incorporates the architectural strengths of these systems into a coherent whole. In addition, Tecate possesses at least two novel features that are not found in other data visualization systems. One feature is Tecate's use of an interpretive language that can describe three-dimensional (3-D) virtual worlds. This language is more than a markup language in that it is capable of performing arbitrary computations and facilitating communication among different processes. The second novel component of Tecate is the presence of an expert system that automatically crafts interactive visualizations of data. This system is intended to make data space exploration easier to perform by having end users simply state their goals while leaving the details of implementing a visualization to attain those goals to the expert system.

The remainder of the paper outlines Tecate's system model and architecture and then identifies and describes Tecate's major components. Finally, the paper sketches Tecate's capabilities by discussing two simple applications that have been implemented on top of the Tecate software framework. The first application is a tool for visualizing earth science data residing in a database managed by a database management system. The second application is a Web browser that uses 3-D graphics as an underlying browsing paradigm rather than depending solely on the medium of hypertext.

Tecate's System Model

After presenting an overview of Tecate's system model, this section provides details of the object model and the interpretive, object-oriented language used to describe virtual world objects.

Overview

From the standpoint of an applications programmer, Tecate is a distributed, object-oriented system. All major components of Tecate, as well as entities appearing in virtual worlds created by Tecate, are objects that communicate with one another by means of message passing. The main focus within Tecate is on object-object interactions. These interactions occur primarily when objects send messages to one another. An object can also send a message to itself, which has the effect of making a local function call. Unlike with graphics systems such as Open Inventor, rendering is not a central activity within Tecate; rather it is just a side effect

of object-object interactions.[6] In this sense, Tecate is like virtual reality programming systems such as Alice and MR/OML, although Tecate is far more flexible.

In the Tecate system, objects can create and destroy other objects and can alter the properties of existing objects on-the-fly. Such capabilities make Tecate very extensible and give it great power and flexibility. These capabilities can also cause problems for applications programmers, however, if care is not taken when writing programs. Presently, all of an object's properties are visible to all other objects, and hence those properties can be manipulated from outside the object. In the future, some form of selective property hiding needs to be added so that designated properties of an object cannot be altered by other objects.

A powerful feature of Tecate is its ability to dynamically establish object-subobject relationships. This feature provides a mechanism for building assemblies of parts similar to the mechanisms in classical hierarchical graphics systems like Doré or Open Inventor.[7] This feature also provides the capability of creating sets or aggregates of objects that share some trait, such as being highlighted. Tecate allows all objects within a set to be treated en masse by providing a means of selectively broadcasting messages to groups of objects. A message that is sent to an object can be forwarded to all the object's subobjects. Thus, for example, one object can serve as a container for all other objects that are highlighted; the highlighted objects are merely subobjects of the container. To unhighlight all highlighted objects, a single unhighlight message can be sent to the container object, which then forwards the message to all its subobjects. In general, an object can be the subobject of any number of other objects and thus simultaneously be a member of many different sets.

The handling of user input within Tecate is intended to appear the same as ordinary object-object interactions. All physical input devices that are known to Tecate have an agent object associated with them that acts as a device handler. All objects that wish to be informed of a particular input event register with the appropriate agent. When an input event occurs, the agent sends all registered objects a message notifying them of the event. Complex events, such as the occurrence of event A and event B within a specified time period, can easily be defined by creating new handler objects. These handlers register to be informed of separate events but then, in turn, inform other objects of the events' conjunction.

The Object Model

Tecate uses an object model in which no distinction is made between classes and instances, as is done in languages like C++.[8] In Tecate, there is a single object creation operation called cloning. Any object in the system can serve as a prototype from which a copy can be made through the clone operation. A clone inherits properties from its prototype by copying the prototype's properties, but any such property can be altered or removed, either by another object or by the clone itself, so that a clone can take on an identity of its own.

The object model is based on delegation. When Tecate clones an object to produce a new object, the prototype's properties are not explicitly copied. Instead, the new object retains a reference to the object from which it was cloned. When a reference to a property is made within an object, the system looks for the property value locally within the object. If no property value is found locally, then the object's prototype is searched to associate a value with the reference. If the prototype is itself a clone, the prototype's prototype is recursively searched to resolve the reference, and so on. This type of "lazy" evaluation of property references is called delegation.

Note that with delegation, a change in value for a property in an object may affect the values of all other objects that can trace their ancestry through prototype-clone relationships to the original object. This type of semantics is useful for establishing class-instance-like relationships between objects. For example, one object may represent a particular class of automobile tire, and all clones of the object would represent class instances. If a class-level change is needed that would affect all instances, e.g., a new tread pattern is to be introduced, only the object representing the tire class needs to change.

The clone-prototype chaining implied by delegation can be overridden by changing the property values locally. Thus, if one particular tire instance is to have a new tread pattern, then the pattern is altered in that instance only. References to the tread pattern for that object will use the local tread value rather than chain back to the tire class object. All other instances will continue to reference the value present in the tire class object.

All Tecate objects possess four classes of properties:

1. Appearance—attributes that affect an object's visual appearance, such as geometric and topological structure, color, texture, and material properties
2. Behaviors—a set of methods that are invoked upon receipt of messages from other objects
3. State—a collection of variables whose values represent an object's state
4. Subobjects—a list of objects that are parts of a given object, just as a wheel is part of a car

Although most users of the system uniformly see communicating objects, a distinction is actually made between two kinds of objects based on how they are implemented by applications programmers. Resource objects are implemented primarily as external processes using some compilable, general-purpose programming language such as C or Fortran. Objects that have

compute-intensive behaviors or whose behavior executions are time-critical are generally implemented as resource objects. For instance, most Tecate objects that provide system services, such as rendering or database management, are implemented as resource objects.

Objects populating virtual worlds that represent data features are implemented differently than resource objects by using an interpretive programming language called the Abstract Visualization Language (AVL). Such objects are called dynamic objects because they may be created, destroyed, and altered on-the-fly as a Tecate session unfolds. Nonetheless, the ability to dynamically add, remove, and alter object properties is not solely endemic to dynamic objects. Resource properties may also be changed on-the-fly because resources are actually implemented with a dynamic object that interfaces to the portion of the resource that is implemented as an external process.

The Abstract Visualization Language

AVL is essential to the Tecate system; it is throughAVL that applications programmers write applications that use Tecate's features.[9] AVL is an interpretive, object-oriented programming language that is capable of performing arbitrary computations and facilitating communication among different processes. Through this language, applications programmers specify and manipulate object properties and invoke object behaviors by sending messages from one object to another.

AVL is a typeless language that manipulates character strings; it is based on the Tcl embeddable command language.[10] AVL extends Tcl by adding object-oriented programming support, 3-D graphics, and a more sophisticated event-handling mechanism. Although AVL is a proper superset of Tcl, the relationship between AVL and Tcl is much like that between C and C++. By adding a small set of new constructs to Tcl, the way applications programmers structure AVL programs differs markedly from how they structure Tcl programs, just as the C++ language extensions to C greatly alter the C programming style.

One use of AVL is to describe virtual worlds that represent data sets. Through AVL, objects that populate these worlds can be assigned behaviors that are elicited through user interaction. For instance, selecting a 3-D icon can cause a Universal Resource Locator (URL) to be followed out into the WWW. In this sense, AVL is somewhat like the Hypertext Markup Language (HTML) that underlies all Web browsers today, or, more fitting, it is similar to the Virtual Reality Modeling Language (VRML) that has been proposed as a 3-D analog of HTML.[11] AVL does, however, differ markedly from HTML and VRML, which are only markup languages. Because AVL is a full-fledged programming language that has sophisticated interaction handling built in, it is philosophically more similar to interpretive languages like Telescript, NewtonScript, and PostScript.[12,13,14] Like Telescript, for instance, AVL programs can encode "smart agents" that can be sent across a network to perform user tasks at a remote machine, if an AVL interpreter resides there. Note, however, that in the present version of Tecate, there is no notion of security when arbitrary AVL code runs on a remote machine.

AVL includes some additional commands that augment the Tcl instruction set, for instance, clone and delete. The clone command is the object creation command within AVL, and the delete command is the complementary operation to delete objects from the system. Object properties are specified and manipulated using the add command and deleted using the remove command. Behaviors in one object are initiated by another object using the send command, which specifies the behavior to invoke and the arguments to be passed. Queries about object properties can be made using the inquire command. The which command is used to determine where an object's properties are actually defined in light of Tecate's use of delegation to resolve property references. Finally, AVL provides a rich set of matrix and vector operators that are useful when positioning objects within 3-D scenes.

As an example of how AVL is used in practice, Figure 1 depicts a code fragment similar to one that appears in the WWW application described later in the paper. The code fragment creates a 3-D Web site icon that is positioned on a world map. The code begins with the definition of the *Hyperlink* object from which all Web site icons are cloned. The *Hyperlink* object is itself cloned from the *Visual* object that is predefined by Tecate at system start-up. The *Visual* object contains properties that relate to the viewing of objects within scenes. For instance, objects that are cloned from the *Visual* object inherit behaviors to rotate themselves and to change their color. To the properties that are inherited from the *Visual* object, the *Hyperlink* object adds the state variables *url* and *desc,* which will be used to store respectively a URL and its textual description. In addition, objects cloned from the *Hyperlink* object will inherit the default appearance of a solid blue sphere having unit radius.

The specification for the *Hyperlink* object also defines three behaviors: *init, openUrl,* and *showDesc.* The *init* behavior replaces the *init* method inherited from the *Visual* object. When an object cloned from the *Hyperlink* object receives an *init* message, it sets its *url* and *desc* state variables, positions itself within the scene whose name is given by the argument *scene,* and registers itself with the mouse handler agent to receive two events. When mouse button 1 is depressed, the agent sends the object the *openUrl* message, which in turn requests the WWW Interface to fetch the data pointed to by the object's URL. Depressing button 2 invokes the *showDesc* message, causing the Web site URL and description to be displayed by a previously

```
# Define a prototype for all Web icons
clone Hyperlink Visual

add Hyperlink {
   state {
      url ""
      desc ""
   }
   appearance {
      shape {sphere}
      diffuseColor {0.0 0.0 1.0}
      repType {surface}
   }
   behavior {
      # Initialize hyperlink
      init {url desc pos scene window} {
         addstate url $url
         addstate desc $desc
         send [getself] move "add $pos"
         add $scene "subobject [getself]"
         send $window addEvent "[getself] {Button-1 {openUrl {}}} {Button-2 {showDesc {}}}"
      }

      # Open the URL
      openUrl {} {send www fetch "[getstate url]"}

      # Display the description
      showDesc {} {send metaViewer display "[getstate desc]"}
   }
}

# Initialize an informational landscape
clone scene Visual
clone window Viewer
send window init {scene}

# Create a Web site icon
clone hlink Hyperlink
send hlink init {"http://www.sdsc.edu/Home.html"
      "SDSC home page" "-2.3 -2.0 1.0" scene window}

# Use the SDSC model geometry
add hlink {appearance {shape {box}}}
```

Figure 1
An Implementation of a World Wide Web Icon in the Abstract Visualization Language

defined interface widget called the *metaViewer.* The AVL command getself, which is used within the *init* behavior body, returns the name of the object on which the behavior was called, thus allowing applications programmers to write generic behaviors. The other AVL commands, getstate and addstate, are shorthand for "get [getself] state ..." and "add [getself] {state ...}."

Once the *Hyperlink* object is defined, a *scene,* a display window, and a Web site icon are created. The Tecate *scene* object is cloned from the *Visual* object. The *window* object, cloned from the predefined *Viewer* object, is the viewport into which the scene is to be rendered. Finally, *hlink* is a Web site icon whose appearance differs from that which is inherited from the *Hyperlink* object. Rather than being spherical, the shape of the *hlink* icon is a unit cube.

Tecate's Architecture

The general structure of Tecate and how it relates to application programs is depicted in Figure 2. Tecate consists of a kernel, a set of basic system services, and a toolkit of predefined objects. The Tecate kernel, which is shown in Figure 3, is an object management system called the Abstract Visualization Machine; AVL is its native language. The Abstract Visualization Machine is responsible for creating, destroying, altering, rendering, and mediating communication between objects. The two major components of the Abstract Visualization Machine are the Object Manager and the Rendering Engine.

The Object Manager is the primary component of the Abstract Visualization Machine. It is responsible for interpreting AVL programs, managing a database of

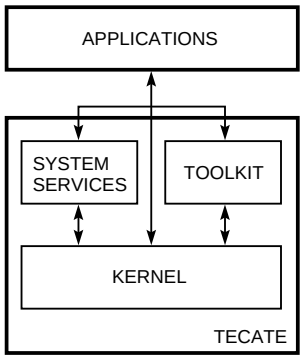


Figure 2
The Tecate System and Its Relationship to Application Programs

objects, mediating communication between objects, and interfacing with input devices. The Object Manager is itself a resource object that is distinguished by the fact that all other resource objects are spawned from this one object. In addition, the Object Manager is responsible for creating a distinguished dynamic object, called *Root,* from which all other dynamic objects can trace their heritage through prototype-clone relationships.

The Object Manager is implemented on a simple, custom-built thread package. Each object within Tecate can be thought of as a process that has its own thread of control. Each thread can be implemented either as a lightweight process that shares the same machine context as the Object Manager's operating system process or as its own operating system process separate from that of the Object Manager. Lightweight processes are so named because their use requires little system overhead, which enables thousands of such processes to be active at any given time. Within Tecate, dynamic objects are implemented as lightweight processes, whereas resource objects are implemented as heavyweight operating system processes, which may or may not be paired with a lightweight, adjunct process. A low-level function library is provided to handle the creation and destruction of threads and to handle interthread communication regardless of how the threads are implemented.

Closely allied with the Object Manager is the Rendering Engine, which is a special resource object wholly contained within the Abstract Visualization Machine. The Rendering Engine is responsible for creating a graphical rendition of a virtual world that is specified by AVL programs interpreted by the Object Manager. When interpreting an AVL program, the Object Manager strips off appearance attributes of objects and sends appropriate messages to the Rendering Engine so that it can maintain a separate display list that represents a virtual world. Display lists are represented as directed, acyclic graphs whose connectivity is determined by object-subobject relationships that are specified within AVL programs.

The present Rendering Engine implementation uses the Doré graphics package running on a DEC 3000 Model 500 workstation.[7] The display lists that are created by invoking behaviors within the Rendering Engine are actually built up and maintained through Doré. The set of messages that the Rendering Engine responds to represents an interface to a platform's graphics hardware that is independent of both the graphics package and the display device.

Layered on top of the Abstract Visualization Machine are Tecate's system services and the object toolkit. The system services consist of a collection of resource objects that are automatically instantiated at system start-up. These resources include an expert system called the Intelligent Visualization System, the Database Interface, the WWW Interface, and a

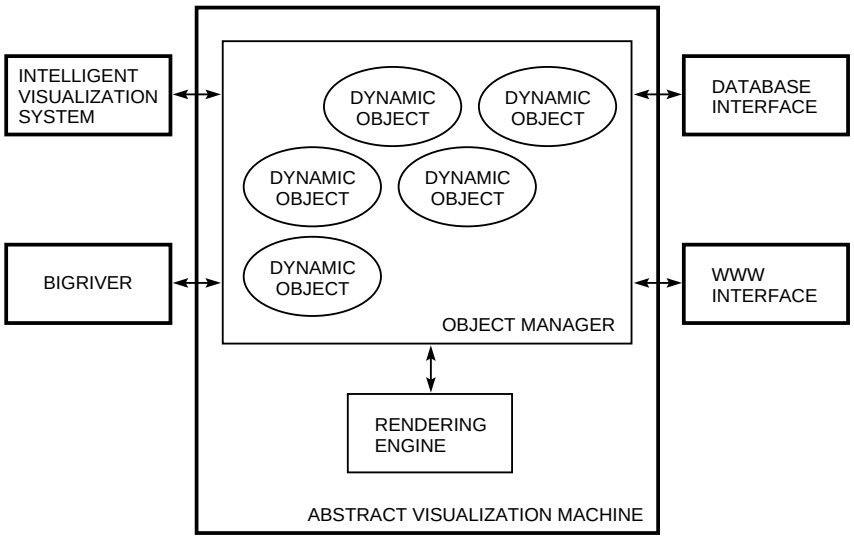


Figure 3
Detail of Tecate's Kernel (the Abstract Visualization Machine) and the System Services Provided by Tecate

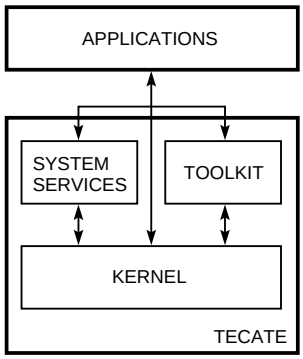


Figure 2
The Tecate System and Its Relationship to Application Programs

objects, mediating communication between objects, and interfacing with input devices. The Object Manager is itself a resource object that is distinguished by the fact that all other resource objects are spawned from this one object. In addition, the Object Manager is responsible for creating a distinguished dynamic object, called *Root,* from which all other dynamic objects can trace their heritage through prototype-clone relationships.

The Object Manager is implemented on a simple, custom-built thread package. Each object within Tecate can be thought of as a process that has its own thread of control. Each thread can be implemented either as a lightweight process that shares the same machine context as the Object Manager's operating system process or as its own operating system process separate from that of the Object Manager. Lightweight processes are so named because their use requires little system overhead, which enables thousands of such processes to be active at any given time. Within Tecate, dynamic objects are implemented as lightweight processes, whereas resource objects are implemented as heavyweight operating system processes, which may or may not be paired with a lightweight, adjunct process. A low-level function library is provided to handle the creation and destruction of threads and to handle interthread communication regardless of how the threads are implemented.

Closely allied with the Object Manager is the Rendering Engine, which is a special resource object wholly contained within the Abstract Visualization Machine. The Rendering Engine is responsible for creating a graphical rendition of a virtual world that is specified by AVL programs interpreted by the Object Manager. When interpreting an AVL program, the Object Manager strips off appearance attributes of objects and sends appropriate messages to the Rendering Engine so that it can maintain a separate display list that represents a virtual world. Display lists are represented as directed, acyclic graphs whose connectivity is determined by object-subobject relationships that are specified within AVL programs.

The present Rendering Engine implementation uses the Doré graphics package running on a DEC 3000 Model 500 workstation.[7] The display lists that are created by invoking behaviors within the Rendering Engine are actually built up and maintained through Doré. The set of messages that the Rendering Engine responds to represents an interface to a platform's graphics hardware that is independent of both the graphics package and the display device.

Layered on top of the Abstract Visualization Machine are Tecate's system services and the object toolkit. The system services consist of a collection of resource objects that are automatically instantiated at system start-up. These resources include an expert system called the Intelligent Visualization System, the Database Interface, the WWW Interface, and a

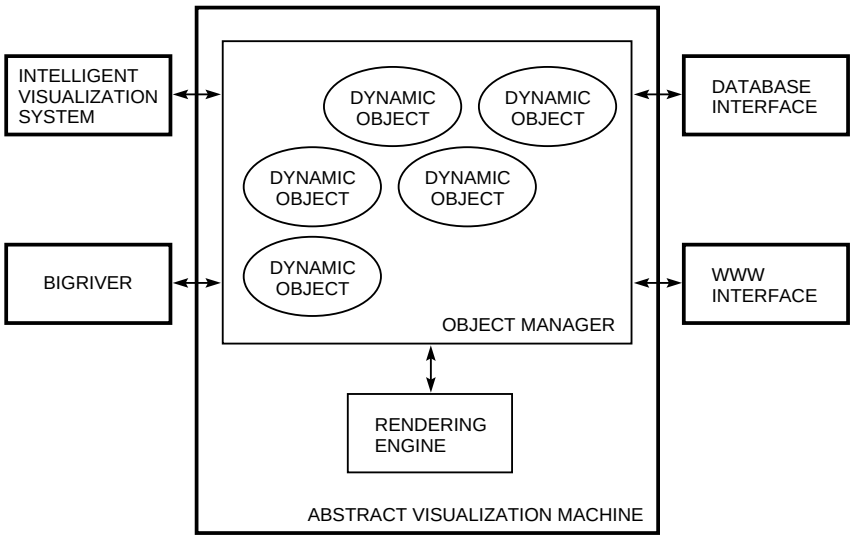


Figure 3
Detail of Tecate's Kernel (the Abstract Visualization Machine) and the System Services Provided by Tecate

visualization programming system called BigRiver. Figure 3 shows these resources in relationship to Tecate's kernel. Each resource is a Tecate object that has a number of predefined behaviors that can be useful to applications programmers. For instance, the WWW Interface has a behavior that fetches a data file referred to by a URL and then translates the file's contents into an appropriate AVL program.

The toolkit within Tecate is a set of predefined dynamic objects that programmers can use to develop applications. These objects are considered abstract objects in the sense that they are not intended to be used directly. Rather, they serve as prototypes from which clones can be created. The toolkit consists of objects such as viewports, lights, and cameras that are used to illuminate and render virtual worlds. The toolkit also contains a modest collection of 3-D user interface widgets that can be used within virtual worlds created by an applications programmer. These widgets include sliders, menus, icons, legends, and coordinate axes.

One useful object in the toolkit that aids in simulating physical processes and helps in performing animations is a clock. This object is an event generator that signals every clock tick. If objects wish to be informed of a clock pulse, those objects register themselves with the clock object just like objects register themselves with input device agent objects. The default clock object can be cloned, and each clone can be instantiated with a different clock period down to a resolution of one millisecond. Any number of clocks can be ticking simultaneously during a Tecate session. Since new clocks can be created dynamically, and objects can register and unregister to be informed of clock pulses on-the-fly, clocks can be used as timers and triggers, and as pacesetters.

Application Resources

Tecate's system services are predefined application resources that aid in interactively visualizing data. As mentioned previously, these objects include the Intelligent Visualization System, the Database Interface, the WWW Interface, and the BigRiver visualization programming system. In addition, an applications programmer can easily add new application resources using tools provided with the base Tecate system. Such new resources can be built around either user-written programs or commercial off-the-shelf applications. To create a new application resource, a programmer needs to provide a set of functions that can be invoked by other Tecate objects. These functions correspond to behaviors that are called when the resource receives a message from other objects. Tools are provided to register the behaviors with Tecate and to manage the communication between a resource and other Tecate objects.[15]

The Intelligent Visualization System

The Intelligent Visualization System allows Tecate to dynamically build interactive visualizations and user interfaces that aid nonexpert end users in exploring data spaces. This knowledge-based system is similar in concept to other expert visualization systems, as the literature describes.[16–21] The Intelligent Visualization System differs from other expert visualization systems in two important ways. First, the Intelligent Visualization System does not merely create a presentation of information as do most other systems. Instead, the Intelligent Visualization System creates virtual worlds with which end users can interact to alter the way data is presented, to make queries for additional data, and to store new data back into data spaces.

The second way the Intelligent Visualization System differs from expert visualization systems is that it takes a holistic approach to fashioning a visualization. Most systems decompose data into elementary components, determine how to visualize each component separately, and then recompose the individual visualizations into a final presentation. In contrast, Tecate's Intelligent Visualization System analyzes the full structure of data by relying on a sophisticated data model based on the mathematical notion of fiber bundles.[22–24] One way to view fiber bundles is as a generalization of the concept of graphs of mathematical functions. Depending on the character of a fiber bundle's independent and dependent variables, certain visualization techniques are more applicable than others.

In general, the Intelligent Visualization System automatically crafts virtual worlds based on a task specification and a description of the data that is to be visualized. A task specification represents a high-level data analysis goal of what an end user hopes to understand from the data. For instance, an end user may wish to determine if there is any correlation between temperature and the density of liquid water in a climatology data set. Usually, task specifications must be input by an end user, although at times they can be inferred automatically by the system. Tecate provides a simple task language from which task specifications can be built, and it provides a point-and-click tool for end users to create these specifications when needed. Data descriptions, on the other hand, do not require any end-user input because they are provided automatically by a data-space interface when data is imported into the system.

From the data description and task specification, a Planner within the Intelligent Visualization System produces a dataflow program that when executed builds an appropriate virtual world that represents a selected data set. The Planner uses a collection of rules, definitions, and relationships that are stored in a knowledge base when building a visualization that addresses a given task specification. Contents of the knowledge base include knowledge about data

models, user tasks, and visualization techniques. The Planner functions by constructing a sentence within a dataflow language defined by a context-sensitive graph grammar. At each step in the construction of the sentence, rules in the knowledge base dictate which productions in the grammar are to be applied and when. Presently, the knowledge base is implemented using the Classic knowledge representation system; the Planner is implemented in CLOS.[25,26]

BigRiver

The dataflow program produced by the Intelligent Visualization System is written in a scripting language that is interpreted by BigRiver, a visualization programming system similar to AVS and Khoros.[1,2] From a technical standpoint, BigRiver is not particularly innovative and will eventually be reimplemented using some existing visualization system that has more functionality. The reason that BigRiver was created from scratch was to better understand how existing visualization programming systems work and to overcome limitations within those systems. These limitations are their inability to be embedded within other applications, their lack of comprehensive data models, and their inability to work with user-supplied renderers. The latest generation of visualization programming systems, such as Data Explorer and AVS/Express, overcome many of these limitations.[3,27]

Like most of the existing visualization systems, BigRiver consists of a collection of procedures called modules, each of which has a well-defined set of inputs and outputs. Functional specifications for these modules represent some of the knowledge contained in the Intelligent Visualization System's knowledge base. Visualization scripts that are interpreted by BigRiver specify module parameter values and dictate how the outputs of chosen modules are to be channeled into the inputs of others.

BigRiver modules come in three varieties: I/O, data manipulators, and glyph generators. All modules use self-describing data formats based on fiber bundles. One format is used for manipulation within memory; the other is an on-the-wire encoding intended for transporting data across a network. An input module is responsible for converting data stored in the on-the-wire encoding into the in-memory format. The data manipulator modules transform fiber bundles of one in-memory format into those of another. The glyph generators take as input fiber bundles in the in-memory format and produce AVL programs that when executed build virtual worlds containing objects that represent features of selected data sets. A single display module takes as input AVL code and passes it to the Abstract Visualization Machine. By means of the Rendering Engine, the Abstract Visualization Machine uses the appearance attributes of objects to create an image of a virtual world that contains the objects.

The Database Interface

The Database Interface provides the means to interact with a database management system, which in the current version of Tecate can be either POSTGRES or Illustra.[28,29] Database queries, written in POSTQUEL for POSTGRES-managed databases or in SQL for Illustra databases, are sent to the Database Interface by Tecate objects where they are passed to a database management system server for execution. The server returns the query results to the Database Interface, which then attempts to package them up as an on-the-wire encoding of a fiber bundle buffered on local disk. If the result is a set of tuples in the standard format returned by POSTGRES or Illustra, the Database Interface performs the fiber bundle translation. For most other nonstandard results, the so-called binary large objects (BLOBs) of the database realm, the Database Interface cannot yet arbitrarily perform the translation into the on-the-wire fiber bundle encoding. The only BLOBs that the Database Interface can deal with presently are those that are already encoded as on-the-wire fiber bundles. The difficult problem of automated data format translation was not addressed during Tecate's initial development, although the intent is to address this issue in the future.

Once query results are buffered on disk, a description of the fiber bundle and the location of the buffer are sent back to the object that made the query request of the Database Interface. That object might then request the Intelligent Visualization System to structure a virtual world whose image would appear on the display screen by way of BigRiver and the Rendering Engine. Objects in the virtual world can be given behaviors that are elicited by user interactions. These behaviors might then result in further database queries and so on. Chains of events such as these provide a means for browsing databases through direct manipulation of objects within a virtual world.

The World Wide Web Interface

The WWW Interface functions similarly to the Database Interface but instead of accessing data in a database, the WWW Interface provides access to data stored on the World Wide Web. Messages that contain URLs are passed to the WWW Interface, which then fetches the data pointed to by the URLs. In retrieving data from the Web, the WWW Interface uses the same CERN software libraries used by Web browsers like Netscape.

Once a data file is fetched, the WWW Interface attempts to translate its contents into an AVL program, which is then passed to the Object Manager for interpretation. AVL either specifies the creation of a new virtual world that represents the data file's contents or specifies new objects that are to populate the current world being viewed. If the fetched data file contains a stream of AVL code, the WWW Interface

merely forwards the file to the Object Manager. If the file contains general data in the form of an on-the-wire encoding of a fiber bundle, the WWW Interface appeals to the Intelligent Visualization System to structure an appropriate virtual world. If the data file contains a stream of HTML code, the WWW Interface invokes an internal translator that translates HTML code into an equivalent AVL program, which is then interpreted by the Object Manager. This interpreter actually understands an extended version of HTML that supports the direct embedding of AVL within HTML documents. Through this mechanism, 3-D objects with which users can interact can be embedded directly into a hypertext Web page—something that few if any other Web browsers can do today.

Example Applications

Applications that browse the contents of data spaces and then interactively visualize selected results have the same overall structure. One browser application component acts as a data space interface, and through this interface queries are posed, query results are imported into the application, and data generated by the application is stored back into a data space. Once data has been imported into the application, a second component must map the data into some appropriate virtual world. Finally, a third component must manage any interactions that may take place between an end user and elements that populate the virtual worlds that are created.

In creating an application using Tecate, the Database Interface and the WWW Interface represent resources that can be used to form the application's data space interface. The mapping of data into a representative virtual world can utilize Tecate's Intelligent Visualization System and the BigRiver visualization programming system. Finally, the management of these worlds can take place through AVL programs that exercise the features of Tecate's Abstract Visualization Machine. The following two examples that were implemented in AVL illustrate how Tecate can be used to create applications that browse data spaces.

Visualizing Data in a Database

A simple example of an application that exploits Tecate's features is one that browses for earth science data in a database and then provides visualizations of that data. The initial user interface for this application is built using a collection of user interface widgets, where each widget is a Tecate dynamic object. Because the Tecate system does not yet have a comprehensive 3-D widget set, some widgets still rely on two-dimensional (2-D) constructs provided by the Tk widget set that is implemented on top of the Tcl language.[30]

Figure 4 depicts the flow of messages between some of the more important objects that are used within the application. One object is the Map Query Tool that is used to make certain graphical queries for earth science data sets whose geographical extents and time stamps fall within user-specified constraints. The tool is built around a world map on which regions of interest can be specified (see Figure 5). When a user marks a region of interest on the map and selects a temporal range, a query message is sent to the Database Interface. The result of the query is returned to the Map Query Tool, which then forwards a description of the result to the Intelligent Visualization System. To structure an appropriate visualization, an inferred select task directive accompanies the result. The ensuing script produced by the Intelligent Visualization System is executed by BigRiver, which produces a stream of AVL code that is sent to the Abstract Visualization Machine for interpretation.

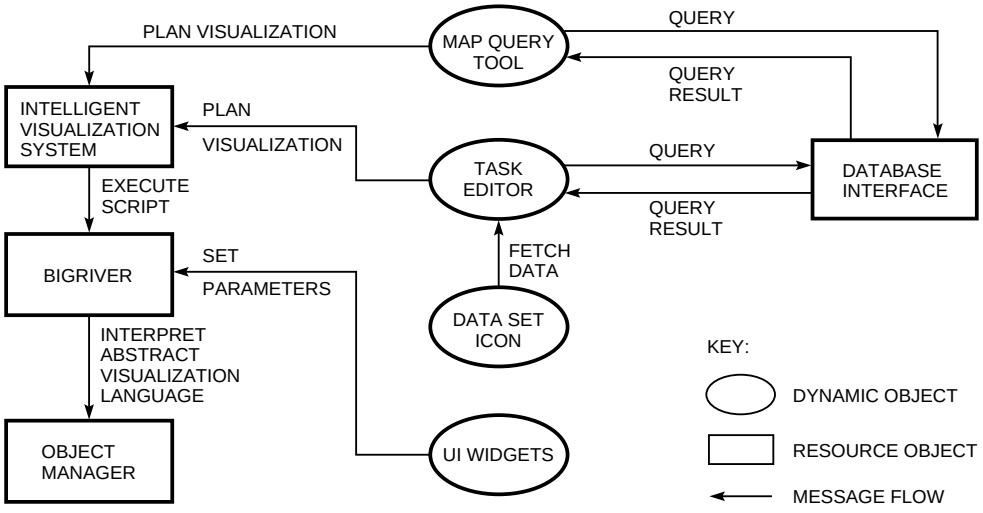


Figure 4
Message Flow between Important Objects in the Earth Science Application

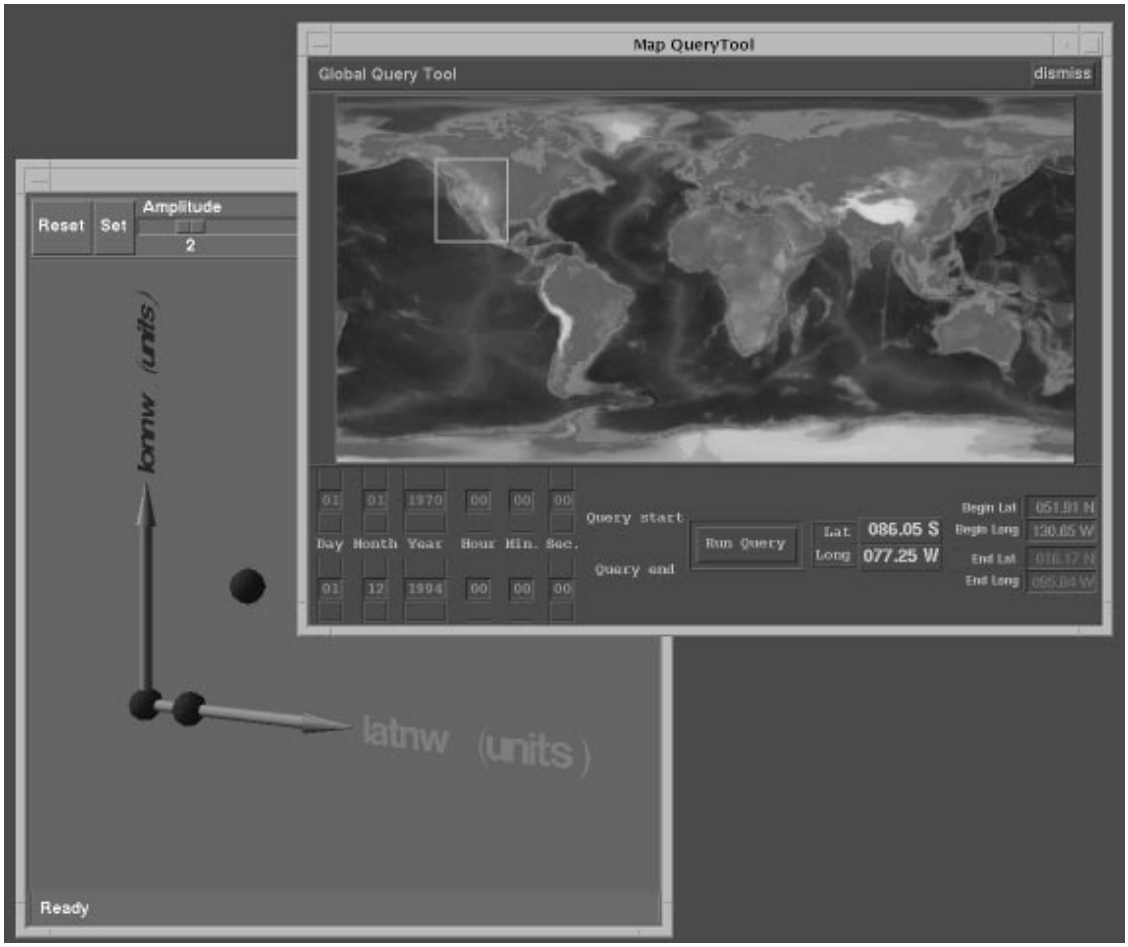


Figure 5
The Map Query Tool Showing a Visualization of a Query Result

This AVL program creates a new virtual world that consists of a collection of 3-D objects. Each object acts as an icon that corresponds to one data set that was returned as the result of the initial query (see Figure 5). The Intelligent Visualization System also builds in two behaviors for each icon. Depending on how a user selects an icon, either the metadata associated with the data set represented by the icon is displayed in a separate window or a query message is sent to the Database Interface requesting the actual data. In the latter case, the Map Query Tool again forwards the query result to the Intelligent Visualization System, and another virtual world containing objects representing data features is created and displayed with the aid of BigRiver and the Abstract Visualization Machine. In general, data exploration proceeds this way by creating and discarding virtual worlds based on interactions with objects that populate prior worlds.

After selecting an icon to actually view the data associated with it, an end user is asked by the Intelligent Visualization System to input a task specification using a Task Editor. Generally, data sets can be visualized in many different ways. The Intelligent Visualization System uses the task specification to select the one visualization that best satisfies the stated task. After a task specification is entered, a visualization of the selected data set appears on the screen. The BigRiver dataflow program that the Intelligent Visualization System creates to do that visualization can be edited by hand by knowledgeable end users to override the decisions made by the system.

Figure 6 shows a Task Editor and a visualization crafted by the Intelligent Visualization System after an end user selected a data-set icon. The visualization represents hydrological data that consists of a collection of tuples, each corresponding to a set of measurements made at discrete geographical locations. Based on the task specification that the end user entered, the Intelligent Visualization System chose to map the data into a coordinate system that has axes that represent latitude, longitude, and elevation. Each sphere represents an individual measurement site, whose color is a function of the mean temperature. When an end user selects a sphere, the actual data values associated with

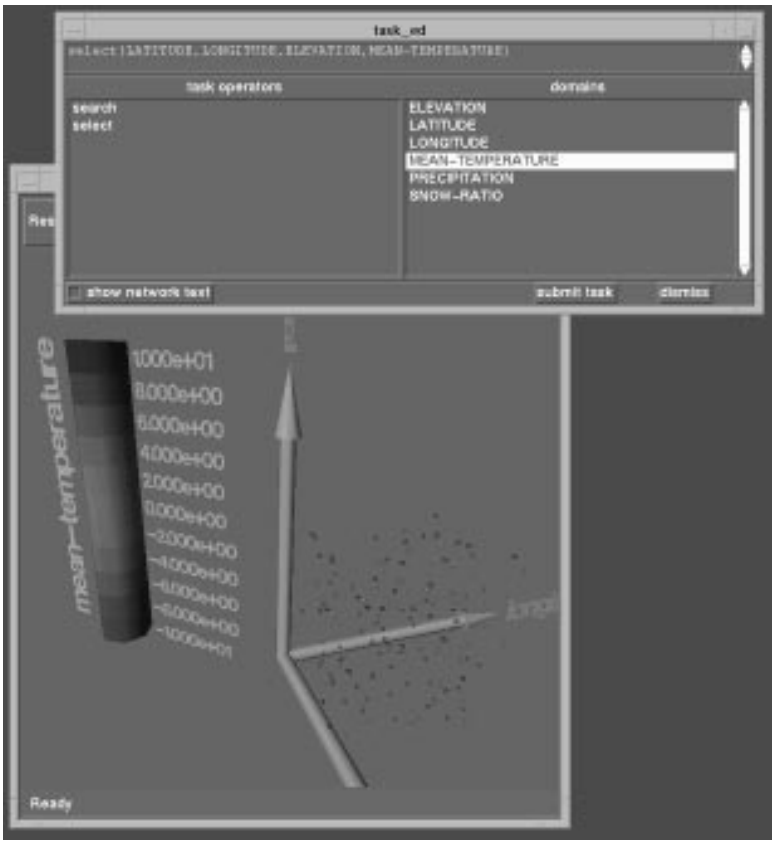


Figure 6
Task Editor Showing a Visualization of Hydrological Data

the location represented by the sphere are displayed. In addition, the Intelligent Visualization System automatically places into the virtual world of the visualization a color legend to help relate sphere colors to mean temperature values.

Figure 7 depicts another virtual world showing a visualization of data-set output from a regional climate model program. The data set is a 3-D array indexed by latitude, longitude, and elevation. Each array element is a tuple that contains cloud density, water content, and temperature values. In this instance, the end user entered a task specification that stated that the spatial variation in temperature was of primary importance. The Intelligent Visualization System responded by specifying a visualization that represented the temperature data as an isosurface, i.e., a surface whose points all have the same value for the temperature. Included in the virtual world is a widget that can be used to change the isosurface value and the field variable that is being studied.

The isosurface widget that appears in the visualization shown in Figure 7 is of special interest because of the way that it is implemented. Embedded in the tool is a slider that is used to change the isosurface value. As with most sliders, the slider value indicator automatically moves when a mouse button is held down while pointing at one of the slider ends. To achieve this simple animation, Tecate's clock object is used. When the mouse button is first depressed while the cursor is over a slider end, the slider indicator registers itself to be informed of clock ticks. From then on, at every clock tick, the indicator receives an update message from the clock, at which time the indicator repositions itself and increments or decrements the current slider value. When the mouse button is released, the slider sends a message to BigRiver indicating that a new isosurface is to be calculated and displayed. In addition, the slider indicator unregisters itself from the clock signaling that it no longer is to receive the update messages. In general, applications can use this same clock mechanism to perform more elaborate animations.

A 3-D World Wide Web Browser

In the Tecate Web browser, exploration of the World Wide Web and its contents occurs by placing an end user onto an informational landscape. This landscape is a 3-D virtual world whose appearance reflects the content and the structure of a designated subset of the entire Web. Upon application start-up, an end user is presented with an initial informational landscape that consists of a planar map of the earth embedded in a 3-D space, as shown in Figure 8. In general, the

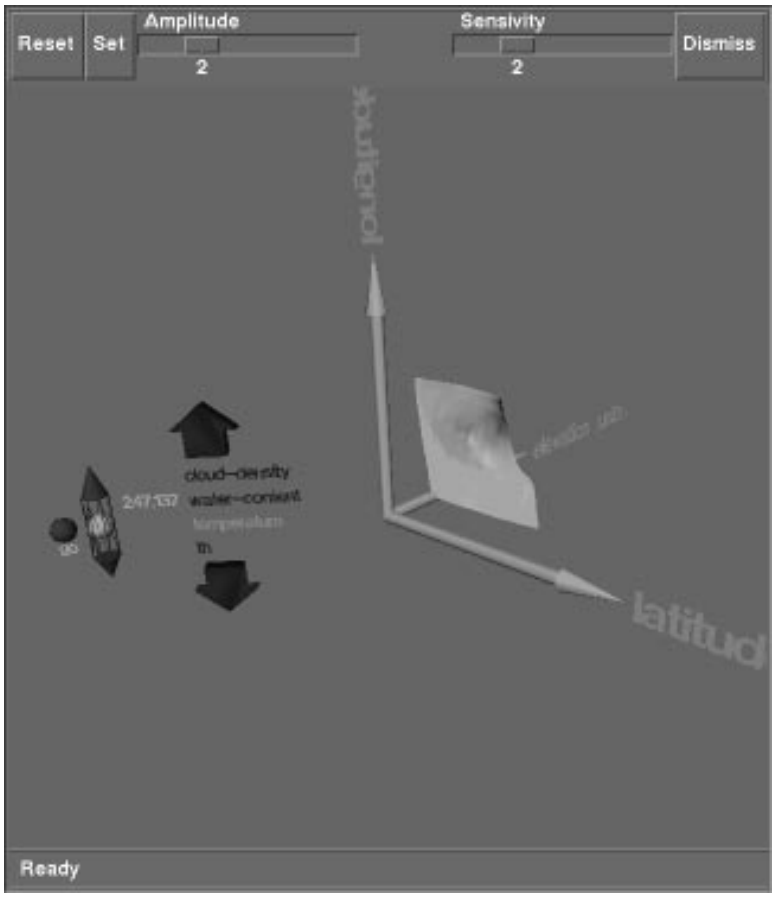


Figure 7
Task Editor Showing a Visualization of Regional Climate Data, Including an Isosurface and a User Interface Widget

initial informational landscape can be any 3-D scene and does not have to be geographically based. For instance, an informational landscape might be a virtual library where books on shelves serve as anchors for hyperlinks to different Web sites.

In the present browser application, selected Web sites appear as 3-D icons on the world map. These icons are positioned either in locations where Web servers physically reside or in locations referenced within Web documents (see Figure 8). A user places information that describes these sites into a database that serves as an elaboration of the hot list of current hypertext-based browsers. When the browser application is first started, it sends a query for the initial complement of Web sites to the Database Interface. The browser application then invokes a BigRiver script that visualizes the results by placing icons representing each site onto the world map.

Suspended above the world map is a 3-D user interface widget that is used to query a database of Web sites that are of interest to an end user (see Figure 8). This database, where the initial set of Web sites is stored, includes information such as URLs, keywords, geographical locations, and Web site types. Currently,

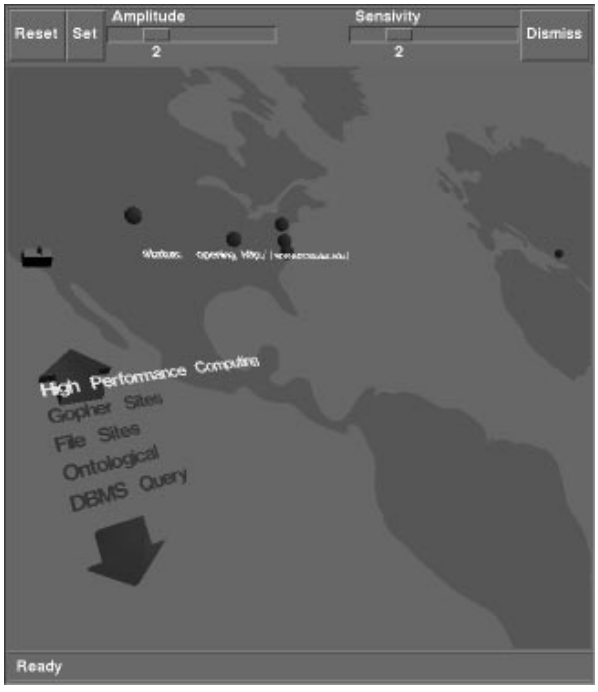


Figure 8
Tecate Web Browser Informational Landscape Showing WWW Sites Depicted as 3-D Icons on a Map of the World

individual users are responsible for maintaining their own databases by adding or removing Web site entries by hand. An automated means for building these databases can be easily added to the browser application so that Web information could be accumulated based on where and when an end user travels on the Web.

During a browsing session, the Web Query Tool allows arbitrary SQL queries to be posed to the database by an end user. In addition, the Web Query Tool has provisions to allow packaged queries to be initiated by a simple click of a mouse button. In both cases, queries are sent to the Database Interface for forwarding to the appropriate database server. The Database Interface packages up the query results as on-the-wire fiber bundles which are returned to the Web Query Tool. The Web Query Tool then invokes a BigRiver script, which converts the fiber bundle data into AVL code. This code, when interpreted by the Object Manager, creates a visualization of the Web sites that satisfies the query. Generally, a visualization such as this consists of placing on the world map a set of 3-D icons whose appearances are a function of the Web site type. However, query result visualizations need not be limited to an organization based on geographical position. For instance, a query for the contents of an end user's own file directory results in a new informational landscape that consists of an evenly spaced grid of icons suspended within a room, as shown in Figure 9.

Each icon that appears within an informational landscape is cloned from an AVL *Hyperlink* abstract object that stores its URL in a state variable. Each Web site icon inherits from the Hyperlink prototype a behavior that causes data pointed to by its URL state variable to be fetched by means of the WWW Interface when the icon is selected. When the data is drawn across the Web, Tecate's WWW Interface attempts to structure a visualization of it. Figure 10 summarizes the message flow between the more important objects within the Web browser application.

If an end user selects an icon and a Web server returns a stream of HTML, the WWW Interface translates the stream into AVL and displays the result on the base of an inverted pyramid whose apex is centered on the chosen icon (see Figure 11). The text and imagery resulting from the HTML appear similarly as they would when visualized using a hypertext-based browser like Netscape. Hyperlinks are represented as highlighted text, which the user can follow by selecting the text. These hyperlinks are Tecate objects that

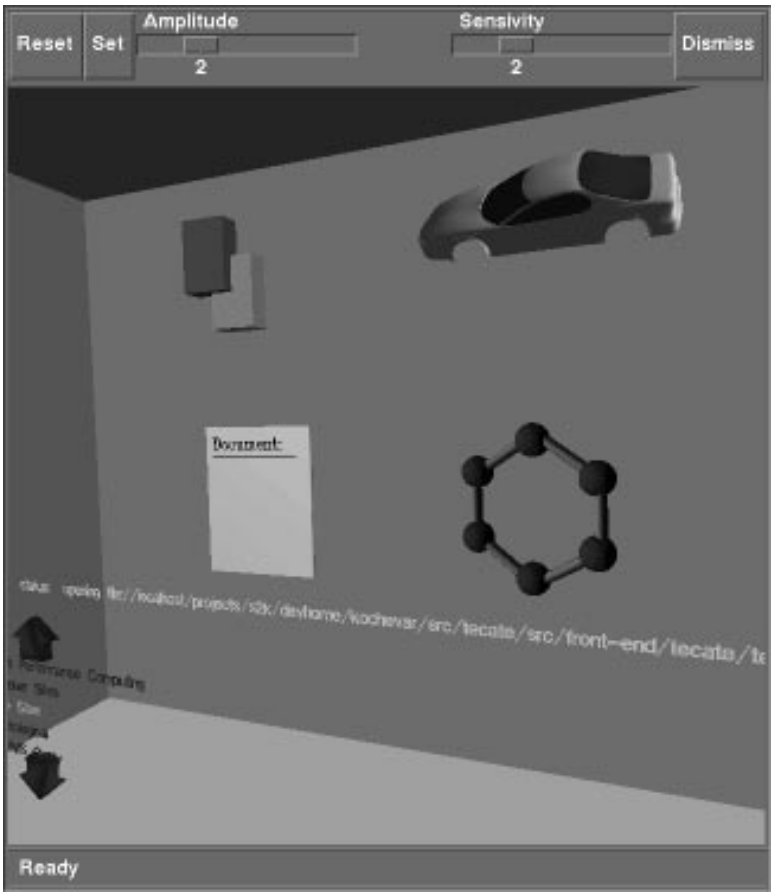


Figure 9
Sample End-user Nongeographical Informational Landscape

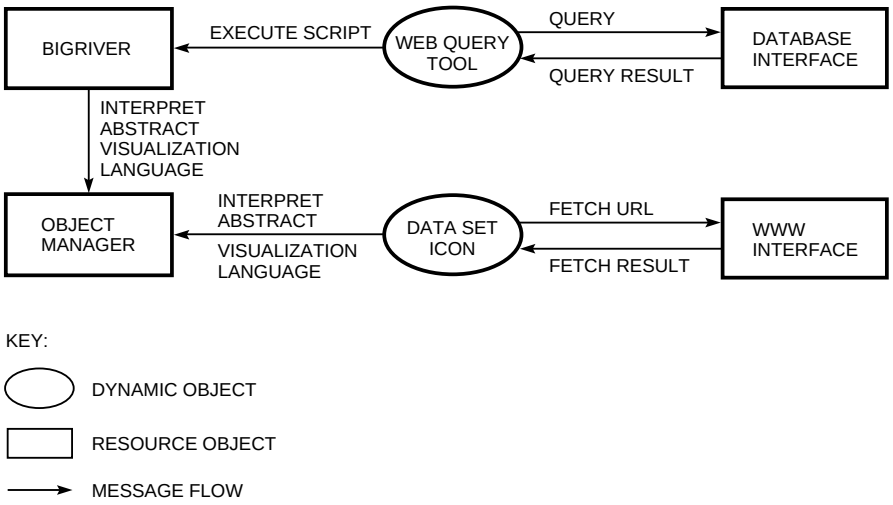


Figure 10
Message Flow between Important Objects in the Web Browser

Figure 11
Results of a Tecate Browsing Session Showing a Hyperlink and a Forest of Pyramids That Represents the User's Travels on the Web

are cloned from the same Hyperlink prototype as the Web site icons. If another HTML document is retrieved by following a hyperlink, that document is viewed on the base of another inverted pyramid whose apex rests on the selected text and so on (see Figure 11). Rather than having to page back and forth between hypertext documents as with most hypertext-based browsers, in Tecate, an end user needs only to move about the virtual world to gain an appropriate viewpoint from which to examine a desired document. Overall, as shown in Figure 11, a browsing session with Tecate's Web browser results in a forest of pyramidal structures that represent a pictorial history of an end user's travels on the Web.

Although Tecate's Web browser is capable of viewing HTML documents, its main purpose is not to emulate what can currently be done using hypertext-based browsers, albeit using 3-D. Rather, the new browser is intended to visualize primarily more complex types of data. When data does not consist of a stream of HTML code, the WWW Interface attempts to visualize what was returned from the Web. These visualizations can take place in virtual worlds separate from the informational landscape from where the data request was initiated, or they can be placed within the original informational landscape. Figure 12 depicts an example of a Web document that has embedded within it a miniature virtual world containing a model of a car. An end user can freely interact with this model to initiate any behavior defined for objects populating the subworld. For instance, selecting the car with the mouse causes the car wheels to spin. Figure 13 shows the AVL code embedded in the HTML page for the Web document shown in Figure 12.

Conclusions

Tecate provides the infrastructure on which applications can be created for browsing and visualizing data from networked data sources. Architecturally, Tecate seeks to bring together into one package useful features found in visualization systems, network browsers, database front ends, and virtual reality systems. As a first prototype, Tecate was created using a breadth-first development strategy. That is, developers deemed it essential to first understand what components were needed to build a general data space exploration utility and then determine how those components interact.

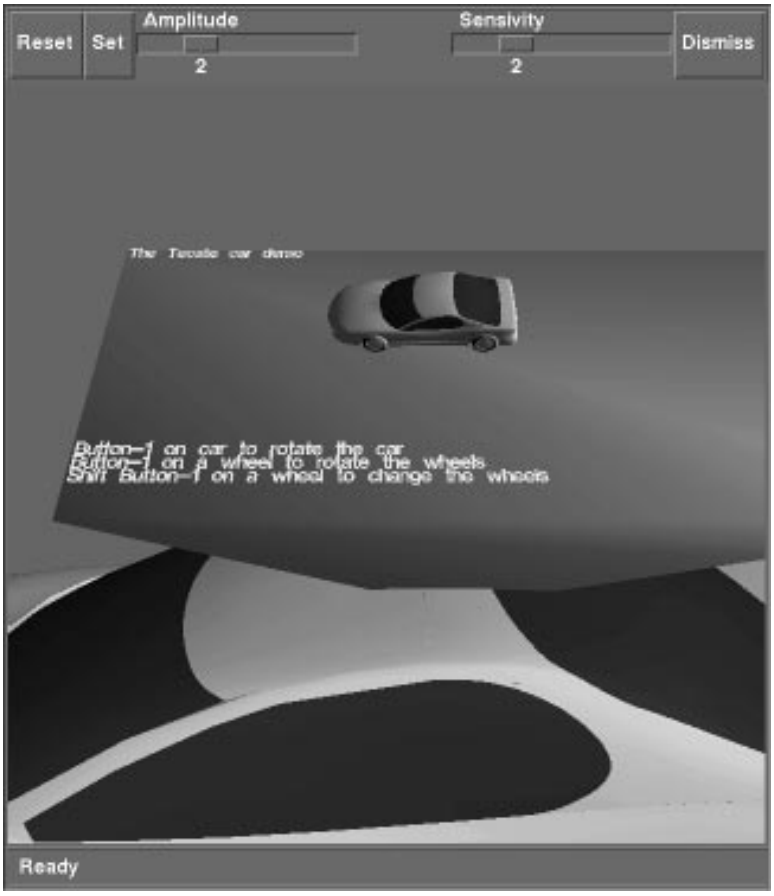


Figure 12
Example of a Web Document with Embedded 3-D Virtual World

```
<HEAD>
<TITLE>The Tecate car demo</TITLE>
</HEAD>

<BODY>

<H1>The Tecate car demo</H1>

<AVL>
# Global variables
global TEC_WEB_PARENT TEC_WEB_WIN
set path "/projects/s2k/sharedata"

# Define car part prototype
clone CarPart Visual
add CarPart {
   state {angle 10}
   appearance {
      repType surface
      interpType surface
   }
   behavior {
      around {args} {
         for {set i 0} {$i < 360} {incr i [getstate angle]} {
            send [getself] rotate "add 0 [getstate angle] 0"
         }
      }
   }
}

# Define car body
clone car_body CarPart
add car_body {
   appearance {
      replacematrix {rotate {0.0 0.0 90.0}}
      shape {AliasObj "$path/car_body.tri"}
   }
}

# Define generic wheel
clone wheel CarPart
        .
        .
        .

# Define car's four wheels
clone back_right CarPart
        .
        .
        .

# Assemble car
clone wheels CarPart
add wheels {subobject {back_right back_left front_left front_right}}
clone car CarPart
add car {
  appearance {replacematrix {translate {28.0 -8.0 3.0} rotate {90.0 90.0 0.0}}}
  subobject {car_body wheels}
}
add $TEC_WEB_PARENT {subobject {car}}

# Bind pick events to car
send $TEC_WEB_WIN addEvent {wheel {Pick-Shift-Button-1 {rot_wheels {}}}}
send $TEC_WEB_WIN addEvent {wheel {Pick-Button-1 {around {}}}}
send $TEC_WEB_WIN addEvent {car {Pick-Button-1 {around {}}}}
</AVL>

<PRE>
Button-1 on car to rotate the car <BR>
Button-1 on a wheel to rotate the wheels <BR>
Shift Button-1 on a wheel to change the wheels <BR>
</PRE>

<HR>
<P>
</BODY>
```

Figure 13
AVL Code Embedded in the HTML Page for the Web Document Example

This development strategy traded off the functionality of individual components for the completeness of a fully running visualization system.

In terms of achieving its design goals, the Tecate effort has been moderately successful. Tecate can now provide interfaces to two kind of data spaces: the World Wide Web and databases managed by the POSTGRES and Illustra database management systems. In addition, interfaces to other data spaces can be implemented easily by creating new resource objects using the tools provided by Tecate. Much work still needs to be done, however. For example, the attendant data translation problem must be satisfactorily solved; data passing through an interface that is stored in one format should be automatically converted into Tecate's favored format and vice versa.

When building visualizations of data, Tecate now understands data that has a specific conceptual structure, in particular, arbitrary sets of tuples and multidimensional arrays where array elements may be tuples. Although data types from many different disciplines possess such a structure, some types remain that do not, for instance, data that has a lattice-like or polyhedral structure. Furthermore, Tecate can now construct only crude visualizations of the data types that it does understand. The primary reason for this shortcoming is that the basic module set within the BigRiver resource is incomplete, and the knowledge base within the Intelligent Visualization System contains limited knowledge of visualization techniques that can be used to transform data into virtual worlds.

At present, Tecate does dynamically craft simple user interfaces and interactive visualizations using its Intelligent Visualization System. This expert system takes into account how data is conceptually structured and end-user tasks regarding what is to be understood from the data. Still, the Intelligent Visualization System does not yet consider data semantics, end-user preferences, or display system characteristics when building visualizations. Nonetheless, Tecate does provide the capabilities to create highly interactive applications. Sophisticated event handling constructs are built into AVL, and the Intelligent Visualization System uses those features to automatically place user interface widgets into the virtual worlds it specifies.

Regarding future work, hopefully, succeeding generations of the Tecate system will include many new features and enhancements. The management of objects needs to be reworked so that thousands of objects can be efficiently handled simultaneously. Although Tecate now builds virtual worlds, virtual reality gadgetry has yet to be integrated into the system. The Abstract Visualization Language needs new features, and it needs to be streamlined. Tecate can also benefit greatly from a more complete toolkit of 3-D widgets that can be used to interact with objects within virtual worlds. Finally, the Doré graphics system that Tecate uses should be replaced with a more mainstream system like OpenGL, which will allow Tecate to run on a wide variety of hardware platforms.

Tecate is an exciting system to use and an excellent foundation from which to pursue further research and development in the exploration of general data spaces. Tecate advances the state of the art by demonstrating a comprehensive means to graphically browse for data and then interactively visualize data sets that are selected. Tecate accomplishes these tasks by using an expert system that automatically builds virtual worlds and by exploiting the flexibility of an interpretive, object-oriented language that describes those worlds.

Acknowledgments

The work described in this paper was supported by Digital Equipment Corporation, the University of California, and the San Diego Supercomputer Center as part of the Sequoia 2000 project. We would like to give special thanks to Frank Araullo, Mike Kelley, Jonathan Shade, and Colin Sharp for their help in constructing the Tecate prototype.

References

1. *AVS User's Guide* (Waltham, Mass.: Advanced Visual Systems Inc., May 1992).
2. *Khoros User's Manual* (Albuquerque, N. Mex.: The Khoros Group, Department of Electrical and Computer Engineering, University of New Mexico, 1992).
3. *IBM Visualization Data Explorer: User's Guide* (Armonk, N.Y.: International Business Machines Corporation, 1992).
4. R. Pausch et al., "Alice: A Rapid Prototyping System for Virtual Reality," Course Notes #2: Developing Advanced Virtual Reality Applications, *Proceedings of the ACM SIGGRAPH '94 Conference* (1994).
5. *Object Modeling Language (OML) Programmer's Manual* (Edmonton, Alberta, Canada: Department of Computing Science, University of Alberta, 1992).
6. P. Strauss and R. Carey, "An Object-oriented 3D Graphics Toolkit," *Proceedings of the ACM SIGGRAPH '92 Conference* (1992).
7. *Doré Programmer's Guide* (Santa Clara, Calif.: Kubota Graphics Corporation, 1994).
8. D. Ungar and R. Smith, "Self: The Power of Simplicity," *SIGPLAN Notices,* vol. 22, no. 12 (December 1987): 227–241.
9. P. Kochevar, "Programming in Tecate," available on the Internet at http://www.sdsc.edu/SDSC/Research/Visualization/Tecate/tecate.html (May 1995).
10. J. Ousterhout, "Tcl: An Embeddable Command Language," *Proceedings of the 1990 Winter USENIX Conference* (1990).

11. G. Bell, A. Parisi, and M. Pesce, "The Virtual Reality Modeling Language Specification," available on the Internet at http://vrml.wired.com (November 1994).

12. J. White, "Telescript Technology: The Foundation for the Electronic Marketplace," General Magic white paper (Sunnyvale, Calif.: General Magic, Inc., 1994).

13. J. McKeehan and N. Rhodes, *Programming for the Newton: Software Development with NewtonScript* (Cambridge, Mass.: Academic Press Professional, 1994).

14. Adobe Systems Incorporated, *PostScript Language Reference Manual* (Reading, Mass.: Addison-Wesley Publishing Company, 1990).

15. L. Wanger, "Writing Tecate Resources," available on the Internet at http://www.sdsc.edu/SDSC/Research/Visualization/Tecate/tecate.html (May 1995).

16. S. Casner, "A Task-analytic Approach to the Automated Design of Graphic Presentations," *ACM Transactions on Graphics,* vol. 10, no. 2 (April 1991): 111–151.

17. E. Ignatius and H. Senay, "Visualization Assistant," *Proceedings of the IEEE Visualization Workshop on Intelligent Visualization Systems* (October 1993).

18. J. Mackinlay, "Automating the Design of Graphical Presentations of Relational Information," *ACM Transactions on Graphics,* vol. 5, no. 2 (1986): 110–141.

19. H. Senay and E. Ignatius, "VISTA: A Knowledge-based System for Scientific Data Visualization," Technical Report GWU-IIST-92-10 (Washington, D.C.: George Washington University, March 1992).

20. Z. Ahmed et al., "An Intelligent Visualization System for Earth Science Data Analysis," *Journal of Visual Languages and Computing* (December 1994).

21. P. Kochevar et al., "An Intelligent Assistant for Creating Data Flow Visualization Networks," *Proceedings of the AVS '94 Conference* (1994).

22. D. Butler and M. Pendley, "A Visualization Model Based on the Mathematics of Fiber Bundles," *Computers in Physics,* vol. 3, no. 5 (September/October 1989).

23. D. Butler and S. Bryson, "Vector-bundle Classes Form Powerful Tool for Scientific Visualization," *Computers in Physics,* vol. 6, no. 6 (November/December 1992): 576–584.

24. R. Haber, B. Lucas, and N. Collins, "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids," *Proceedings of the Visualization '91 Conference* (1991).

25. L. Resnick et al., *CLASSIC Description and Reference Manual for the Common LISP Implementation* (Murray Hill, N.J.: AT&T Bell Laboratories, 1993).

26. G. Steele, Jr., *Common LISP: The Language, Second Edition* (Bedford, Mass.: Digital Press, 1990).

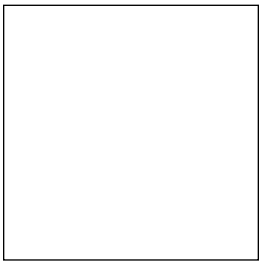
27. *AVS/Express Developer's Reference* (Waltham, Mass.: Advanced Visual Systems Inc., June 1994).

28. M. Stonebraker and G. Kemnitz, "The POSTGRES Next-generation Database Management System," *Communications of the ACM* (October 1991): 78–92.

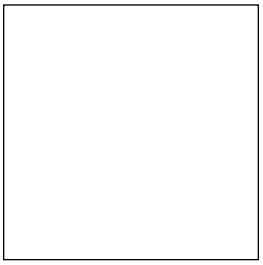
29. *Using Illustra* (Oakland, Calif.: Illustra Information Technologies, Inc., June 1994).

30. J. Ousterhout, "An X11 Toolkit Based on the Tcl Language," *Proceedings of the 1991 Winter USENIX Conference* (1991).

Biographies



Peter D. Kochevar
Peter Kochevar is a principal software engineer in Digital's External Research Program. From 1992 to 1994, he led the data visualization research efforts of the Sequoia 2000 project, which were undertaken at the San Diego Supercomputer Center (SDSC). Currently, Peter is a visiting scientist at the SDSC, where he leads researchers in developing interactive data visualization systems. Peter joined Digital in 1990 as a member of the Workstations Engineering Group. In earlier work, he was a software engineer for the Boeing Commercial Airplane Company. Peter received a B.S. (1976) in mathematics from the University of Michigan and an M.S. (1982) in mathematics from the University of Utah. He also holds M.S. and Ph.D. degrees in computer science from Cornell University.



Leonard R. Wanger
Len Wanger is the head of development at Interactive Simulations, Inc., working on interactive molecular modeling tools. He is also a member of the Computer Science Department staff at the University of California, San Diego, where he researches next-generation visualization systems at the San Diego Supercomputer Center. He received a B.S. in computer science from the University of Iowa in 1987 and an M.S. in architectural science from Cornell University in 1991. His research interests include visual front ends to simulations, database support for visualization systems, navigation in virtual environments, and the perception of complex data spaces.