Chapter 3 DIVINE — Immersive 3D Debugging Information Visualization System Using Metaphor

3.1 An Overview of DIVINE

DIVINE is a debugging information visualization system. It visualizes the information generated by HILCADT, a high-level debugger of Java language. When debugging a debuggee with HILCADT, a user may wish to visualize the data structures of an object. He just needs to give a command to HILCADT, and click the button labeled ‘Visualize’. A snap shot of HILCADT is shown as Figure 3.1.

Figure 3.1 – The user interface of HILCADT
3.2 Architecture of DIVINE

Figure 3.2 figures the architecture of DIVINE. Object info nodes are managed by DIVINE itself and these nodes store the debugging information of object’s data structure sent from HILCADT via TCP/IP. This information contains: type, data members, and other information of requested structure. In addition, unique identity which is generated by debugger is also stored in object info node. But object info node contains no information regarding rendering. All information of rendering is stored in render nodes.

Render nodes, store the information about how to display after getting an object info node, are managed by each metaphor module. There is only one metaphor module in active state at the same time. Once a metaphor module is not active; render nodes stored in this module are released from memory. What information should be stored in render node depends on the designer of the metaphor. For example, coordinates and geometry data are stored in render nodes of the metaphor described in next chapter. And then the drawing method implemented in metaphor module will render these geometries in the 3D scene.
The number of render nodes does not equal to the number of object info nodes.

For example, there is a metaphor which visualizes data structure in a cone tree. Even
though the number of object info nodes and render nodes which draw nodes of the cone tree are the same, there are still some left render nodes which draw edges of the cone tree.

Each metaphor, a plug-in module of the DIVINE, is compiled as a DLL (Dynamic Linking Library). If a new metaphor is added in the future, we just need to copy that DLL file into the specified folder. There is no need to understand how DIVINE works in details when creating a new metaphor.

Current input devices are keyboard and mouse. In future, these devices may contain data gloves, 3D tracker (used to locate the coordinate of an object in virtual space) or the other haptic devices. Because there are various input devices, we design a unified interface to communicate with various devices. The change of input devices is isolated from metaphors. The unified interface is consisted of a set of action codes. Each action of input devices is translated to an action code to communicate with metaphor.

When metaphor gets an action code, it might tell DIVINE to send a request to HILCADT, tell DIVINE to switch a metaphor, or simply move the camera position in the 3D world. Different metaphors may take different action for the same action code, but a lot of actions should be same.
As in Figure 3.1, a command “get obj dn at line 41 in pro:main;” is given in HILCADT. This command is to get the information of an object named ‘dn’ defined in the ‘main’ function in class ‘pro’. After the command is parsed, HILCADT gathers the information of that object, and sends it to DIVINE.

When DIVINE gets this information, DIVINE will parse that. The result after parsing is an object info node with specified format and then store this node into the memory if this object information does not exist in the memory yet, and notify the active metaphor of that a new object info node needs to be processed.

Once a metaphor is notified, it would read all object info nodes stored, and generate some render nodes which would be drawn on the display device (such as monitor, or helmet). A render node contains the information of an object in the 3D scene (such as the shape, the coordinate, and else). What described above is the pipeline of DIVINE.

### 3.3 Graph Structure Support

To support graph structure, what have to face is cycle-contained graph. If a cycle exists in the visualization data, DIVINE might suffer from that memory is consumed. Fortunately, the object information sent from HILCADT contains a unique identity. That is useful to check if an object info node has been stored in memory by DIVINE.
3.4 Parser and Object Info Nodes

The parser is an important part of DIVINE. It is used to parse the information sent from HILCADT and store parsed information in object info nodes if an object has never been stored in the DIVINE.

Data types in Java can be separated into three sets:

- Atomic or primitive type: a data type whose elements are single, no decomposable data items;
- Composite type: a data type whose elements are composed of multiple data items;
- Structured composite type: an organized collection of components in which the organization determines the means of accessing in individual data components or subsets of the collection.

A sketch of these data types is shown in Figure 3.3, and the relationship between these is shown in Figure 3.4.

Figure 3.3 – A sketch of Java data types
For debugger, only composite data type has unique identity, for example: an object, an array. The object info node is stored in memory only when it has unique identity. However, there is a problem: a primitive type array is a composite data type with a unique identity, but the elements of this array are primitive type data without any identity. And then the information of these elements can’t be stored. And then these data can’t be visualized because they are not stored.

This is a big problem for debugging because users hope to use debugging tools to find out the error in their program, and they may need to see the whole information. Therefore, the parser has to handle this problem.

As getting the information of a primitive type array, the parser will automatic to communicate with HILCADT to get the information of all elements of this array. And then the parser converts the information into a member of this array. For example: content of an integer array is “1, 2, 3”, content of a character array is “this is a string”. And then these data can be visualized again.
3.5 Input Interface and Action Codes

An input translator intercepts signals of specific input device, such as mouse and keyboard. Once a signal is intercepted, the input translator will translate it into an action code and then send to the active metaphor. Different device has a different translator. Action codes are a set of flags. Action codes are used to isolate metaphor and input devices. For example a same action may be triggered by mouse, keyboard or haptic devices, such as data glove or head mounted display. However, a metaphor does not need to distinguish these sources. When a metaphor gets an action code, it will do its designated task. The action codes with keyboard and mouse are listed as Table 3.1.
<table>
<thead>
<tr>
<th>Action Code</th>
<th>Comment</th>
<th>Current Triggered By</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_SEPCIAL_k</td>
<td>Do a special action with numerical meaning.</td>
<td>Device</td>
</tr>
<tr>
<td>AC_DIVINE_QUIT</td>
<td>Quit DIVINE / Processed by DIVINE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>AC_DIVINE_SWITCH</td>
<td>Switch metaphor / Processed by DIVINE</td>
<td>Tab</td>
</tr>
<tr>
<td>AC_CURSOR_MOVE</td>
<td>Move cursor on screen / Processed by DIVINE</td>
<td>Mouse’s movement</td>
</tr>
<tr>
<td>AC_PICKUP</td>
<td>Do an action on cursor’s position with intent to pick up something.</td>
<td>Press left button</td>
</tr>
<tr>
<td>AC_SELECT</td>
<td>Do an action on cursor’s position with intent to select something</td>
<td>Click left button</td>
</tr>
<tr>
<td>AC_PICKUP_AUX</td>
<td>Do an action on cursor’s position with intent to pick up something, and this action is not triggered usually.</td>
<td>Mouse</td>
</tr>
<tr>
<td>AC_SELECT_AUX</td>
<td>Do an action on cursor’s position with intent to select something, and this action is not triggered usually.</td>
<td>Click right button</td>
</tr>
<tr>
<td>AC_SELECT_SQUARE</td>
<td>Do an action on the range of the dragging square.</td>
<td>Dragging</td>
</tr>
<tr>
<td>AC_CAMERA_ROTATE</td>
<td>Rotate camera’s orientation / Processed by DIVINE</td>
<td>Mouse &amp; KB</td>
</tr>
<tr>
<td>AC_CAMERA_MOVE</td>
<td>Move camera’s position / Processed by DIVINE</td>
<td>Mouse &amp; KB</td>
</tr>
</tbody>
</table>

Table 3.1 – The existing action codes