

A Visual Representation of Complex Relationships for Object-Oriented Databases

Bonghee Hong

Dept. of Computer Engineering, Pusan National
University, Pusan, 609-735, Korea
bhhong@hyowon.cc.pusan.ac.kr

Bonggi Jun

Electronic Switching Operations Research Center,
Korea Telecom, Seoul, 137-140, Korea
bgjun@kingwoo.kotel.co.kr

Abstract

It becomes complicated to represent complex object structures as a network of nodes and arcs when a large volume of objects is to be browsed. This paper presents the issues of designing and implementing an icon-based graphic user interface for representing complex relationship instances as a node-and-link form. The icon-based browser presented in this paper supports the simplification of the *visual representation of complex relationships among instance level objects* by introducing a new browsing method which uses instance-set-icon, active/inactive link, and inactive icon. We also discuss the issues of implementing a two-layer architecture which consists of a stored object layer and a window object layer in order to realize the simplified visual representation of complex relationships. The icon-based browsing for complex object structures has been implemented by using the X11 Motif.

1 Introduction

When we display visually complex object structures, it is very important to browse individual objects easily at the instance level. In general, it is most natural to present relationships by using a network of nodes connected by links. However, when complex object structures are displayed in a node-and-link form, the screen becomes too complicated when a large number of objects are presented. More work needs to be done in managing visual complexity [3,14].

The purpose of icon-based browsing is to allow users to be able to directly navigate complex object structures. However, if all the relationships among instance-level objects are to be displayed in the screen, it is too complex to navigate relationship instances by using icons.

The schema browsing method for showing class hierarchies has been previously addressed by many related works[1, 6, 9, 15]. However, the complexity of browsing relationships among instance-level objects by using both nodes and links has been overlooked. Most of the work has emphasized schema visualization. We focus on the simplification of a visual representation of complex object structures, which allows users to directly navigate and manipulate instance level objects.

In the visual representation of complex object structures, it is even worse that the same icon node to represent a specific object may be repeatedly displayed in the screen because the object may be referenced by any number of objects. This makes the screen complicated, because the content browsing of the referenced object is repeated under the existing synchronized browsing scheme[1, 8, 9]. In addition, in the case that a slot of a given object may have a set of values, denoting a set of values of a slot as a set of icon nodes becomes unmanageable.

To solve this problem, we propose a **new browsing method** by which all the objects belonging to the same class must be always denoted by a single icon node even if they are distinct objects. As a result, each icon node corresponds to a class. At a glance, this approach might be similar to the existing schema browsing method. However, our work on browsing of object-level structures differs from the schema browsing method because it shows the structural relationships among instance-level objects. When there are multiple relationship instances between two objects, we introduce a **new relationship browsing method** by which one relationship is denoted as an **active link** while the other relationships are denoted as **inactive links**. The active link directly shows the values of a given slot. In the case of inactive links, the values of the corresponding slots are not shown in the icon node connected by the inactive link. In addition, we use a new icon, called **instance-set-icon**, to simplify the visual representation of a set of objects.

Our visual representation scheme of complex object structures doesn't directly correspond to the stored object structures. So, it needs to derive a window view to show icon-based display of complex object structures from the

internal object structures. To realize the simplification of the visual representation of object structures, we present a two-layer architecture which consists of a *window object layer* and a *stored object layer*, and then develop an algorithm for mapping between two layer objects. The window object layer maintains the widget objects to show icon-based display of complex object structures. The stored object layer maintains the internal data structures which are created and updated.

In section 2, we describe related works. In section 3, we briefly summarize complex object structures handled in this paper. An incomplete visualization scheme designed for browsing complex object structures will be described in section 4. In section 5, we discuss the issues of implementing the simplified visualization of complex relationships. Finally, we summarize key points in section 6.

2 Related Works

The icon-based GUI was first implemented in Xerox's STAR computer[13]. Objects visualized by graphical symbols, called icons, may be activated by direct manipulation[12]. ZOO[11] was directly influenced by [13]. ZOO provides a direct-manipulation interface to object-level knowledge similar to KL-ONE or Loops. Our work is most similar to the ZOO system which provides the visual representation of object-level knowledge, but not identical. The ZOO system has the complexity of large data visualization when the amount of data displayed grows. Our work on the visual representation of complex object structures differs from the ZOO's work because we revise the existing icon-based graphic interface for managing the complexity of instance-level display.

We also have been influenced by the work on the visual representation for navigating knowledge structures by using nested boxes rather than a network of nodes linked by arcs in MUE(Museum Unit Editor)[14]. In the MUE display, the visual representation of knowledge is a set of nested boxes, each box representing an object. Overlapping objects and nested objects are represented by overlapping boxes and nested boxes respectively. Our work was inspired by the MUE's point in which a node-and-link graphical display becomes impossibly tangled as the knowledge base grows. However, when we represent densely interrelated objects as nested boxes, the visual representation of complex object structures in terms of nested boxes is not natural and also less understandable than node-and-link form. So, we adopt the method of node-and-link graphical display for the visually representing of object-level relationships. Instead, we focus on simplifying the complexity of the visual representation of object-level relationships.

There have been many research works on graphics-based interface to databases. Most of them have addressed the problems related to the schema-based browsing[2, 8]. To our knowledge, there are few works to manage the complexity of node-and-link display of instance-level relationships in databases. To deal with large data visualizations, the filtering method has been used for selectively reducing the amount of information displayed[3]. However the filtering method is difficult to navigate complex relationships without querying. In Addition, the G+/GraphLog Visual Query System[3] allows for varying levels of abstraction in the display of hierarchical data for managing complexity. The abstraction display in [3] was not concerned with the display of non-hierarchical relationships.

The table-based interface to object-oriented databases [1, 6] provided facilities for synchronized browsing. The synchronized browsing has been employed for browsing active/inactive links in this paper. KIVIEW[9], SIG[8], and OdeView[1] support browsing of objects, navigation of related objects and synchronized browsing. However, those works differ from our work in that we use the icon-based node-and-link display method instead of the table-based interface like [1, 6]. For browsing structured objects, the method of node-and-link graphical display has an advantage over the table-based browsing method because the tables for displaying structured objects are difficult to navigate.

3 Complex Object Structures

We describe complex object structures in terms of an object-oriented frame-based representation scheme[10]. The object-oriented frame-based representation scheme has three kinds of is-a hierarchies for defining the relationships 1) between a super class and its subclasses, 2) between a class and its instances, and 3) between an instance object and its child instance objects. In general, is-a hierarchies are relationships between a generic concept and its specialized concepts. Fig. 1 shows three kinds of is-a hierarchies. DEVELOP is a subclass of PROJ. An object d_1 is an instance of the class DEVELOP. The instance object d_4 can inherit the properties of its parent instance d_2 where both d_2 and d_4 belong to the class DEVELOP.

Relationships between instance objects are referred as relationship instances. Relationships between classes are referred as class relationships or relationship types. Graphs with many nodes and edges for presenting relationship instances are difficult to display and navigate.

An example of an association relationship is shown in Fig. 2. The relationship S_1 is one-to-many where each instance of A can be related to N instances of B in Fig. 2. At the same time, the relationship S_2 is many-to-many.

The relationship type as shown in Fig. 2(a) is simple but relationship instances between instance objects are very complicated as shown in Fig.2(c). In the object-oriented paradigms, relationships are represented as the slot values as shown in Fig. 2(b). When directly browsing relationship instances, we encounter two problems. First, there are too many links for representing relationship instances. Second, the same node representing an object may be repeatedly displayed. In Fig. 2(c), the object B₁ related to A₁ appears two times. We address the problem of reducing the complexity of graphical display of relationship instances.

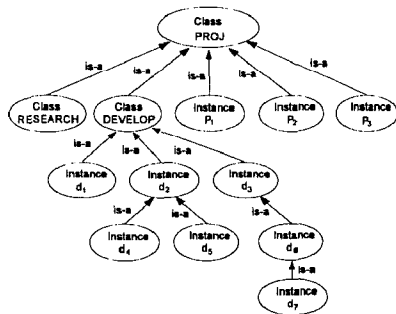


Figure 1. Three Kinds of is-a Hierarchies

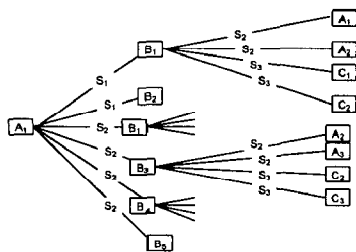
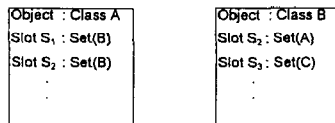
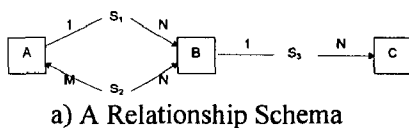


Figure 2. Non-Hierarchical Relationships

4 Incomplete Visualization Scheme

An icon-based graphical representation scheme is employed for displaying complex object structures. We use two kinds of graphical primitives: nodes and links. The graphical symbols used to represent classes, instances, and relationships are shown in Fig. 3. A

doubled rectangle denotes a class(Fig. 3(a)). A single rectangle denotes an instance object(Fig. 3(b)). The dashed rectangle denotes an inactive icon (Fig. 3(c)). In the case of inactive icons, browsing of a specific inactive icon is suspended until the inactive icon is relicked. That is, inactive icons correspond to unexpanded directories for viewing file systems. A tripled rectangle, called instance-set-icon, denotes a set of objects(Fig. 3(d)). The pseudo instance-set-icon shown in Fig. 3(e) is used for representing the fact that the front rectangle represents a real value, but the two dashed rectangles represent the slot values of inactive links. The surface of the pseudo instance-set-icon represents the value of the current active icon.

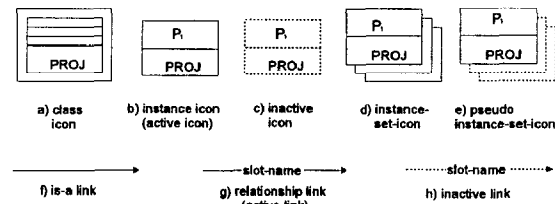


Figure 3. Graphical Representation Primitives

Hierarchical and non-hierarchical relationships are represented as shown in Fig. 3(f,g,h). The is-a relationship is always represented by a single arrow. Non-hierarchical relationships are denoted by solid arrows with a slot name(Fig. 3(g)) which is called an active link. The dashed arrow denotes an inactive link which doesn't show the value of a specified slot(Fig. 3(h)). To represent object structures as both icons and links are not a new idea presented for the first time in this paper. However, to use instance-set-icon, inactive icon, pseudo instance-set-icon, and inactive link are a first attempt to simplify the visual representation of complex object structures. The motivation is to employ incomplete visualization for dealing with visual complexity.

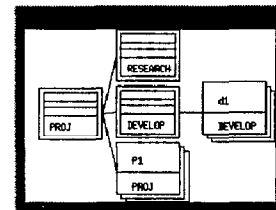


Figure 4. A Node-and-Link Representation of is-a Hierarchies

Many instance-of relationships can be simplified by using an instance-set-icon as shown in Fig. 4. Thus, class hierarchies are represented by arrows between two doubled rectangles, while instance-of relationships are represented by arrows between a doubled rectangle and a tripled rectangle(or a single rectangle). The three

instances P_1 , P_2 and P_3 of the class PROJ are represented by only one instance-set-icon P_1 . The seven instances of the class DEVELOP are also denoted by one node d_1 .

The content of an object can be displayed in a node-and-link form like Fig. 5. A project P_1 has four slots such as "name", "pno", "budget", and "mgr-prj". The slot names are designated by named links while the values associated with these slots are designated by icons. In the case that a slot has a set of values, the values of the slot are represented by an instance-set-icon.

In the case that there are multiple relationships between two classes, the direct representation of multiple relationships instances in a node-and-link form is difficult to display. When all of slot values of an object are displayed in the screen by using only icons and links, there are too many nodes and links like Fig. 2(c). To solve this problem, we use an instance-set-icon for representing a set of objects which are the values of a slot. As shown in Fig. 2(c), the object A_1 has two slots S_1 and S_2 for representing two relationships. Because the slot S_1 has the same domain as S_2 does, we represent the values of two slots as a single instance-set-icon as shown in Fig. 6(a). In Fig. 2(c), since the object B_1 is the common value of two slots S_1 and S_2 of A_1 , the content browsing of B_1 also was repeated two times.

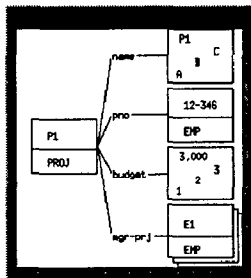


Figure 5. The Content Browsing of P_1

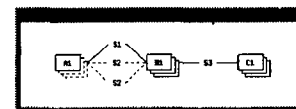
In general, if two slots have the same domain, we define the relation between the two slots as a **sibling relation**. In this approach, we represent all the slots belonging to a sibling relation as a single icon. If two slots belong to a sibling relation, it is possible to represent the two slot values as a single node by employing a pseudo instance-set-icon and/or an inactive link. In other words, if two slots have the same domain, we always represent the values of two slots as a single icon although the values of two slots are different from each other. When there are many relationship instances between objects, this approach can reduce the number of nodes of links displayed. Of course, the number of links displayed can't be reduced. This is a new browsing method presented in this paper.

In Fig. 6(a), the solid arrow denotes an active link which means that the surface value of the current instance-set-icon B_1 is the value of the slot S_1 of the object A_1 . The dashed arrows denote inactive links which

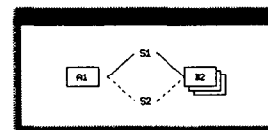
mean that the current instance-set-icon doesn't directly show the values of inactive links.

When the instance-set-icon B_1 is displayed, it is necessary to display the content of B_1 to show the other relationships connected to B_1 . This is inspired by the existing synchronized browsing. Because the domain of the slot S_2 of B_1 is the class of object A_1 , we denote the slot S_2 of B_1 as an inactive link. At that time, the pseudo instance-set-icon is used for representing the values of an inactive link S_2 of B_1 , because the slot S_2 of B_1 has a set of values. Fig. 6(a) shows a cycle consisting of two links, S_1 and S_2 . To display a self-loop or a cycle, we use an inactive link and a pseudo instance-set-icon.

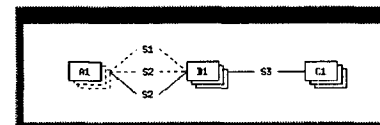
When users want to see the next object of the instance-set-icon B_1 in Fig. 6(a), users may click the instance-set-icon B_1 , which results in Fig. 6(b). At this time, the two slots S_2 and S_3 of B_1 disappear in the screen because B_2 doesn't have any other relations to other objects. When the inactive link S_2 from B_1 to A_1 disappears, the pseudo instance-set-icon A_1 is changed into a simple instance icon A_1 .



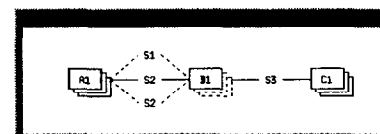
a) S_1 is an active link



b) When the instance-set-icon B_1 in Fig. 6(a) is clicked



c) When the inactive link S_2 of A_1 in Fig. 6(b) is clicked



d) When the inactive link S_2 of B_1 in Fig. 6(c) is clicked

Figure 6. Relationship Browsing

When users want to see the values of the slot S_2 of the object A_1 in Fig. 6(b), users may click the inactive link S_2 of A_1 . If users do that, Fig. 6(b) is changed into Fig. 6(c). If users click an inactive link S_2 of A_1 , the inactive link is replaced by a new active link and then the values of the slot S_2 of A_1 are represented as an instance-set-icon B_1 . The pseudo instance-set-icon A_1 is used again for representing a set of values of the inactive link S_2 of B_1 . In Fig. 6(c), users may click the inactive link S_2 of

B_1 for displaying the values of the slot S_2 of B_1 . As shown in Fig. 6(d), the pseudo instance-set-icon A_1 is replaced by the instance-set-icon A_1 which represents the values of the slot S_2 of B_1 .

There have been few attempts to browse recursive relationships at the instance level. The direct display of recursive relationships in a node-and-link form becomes unmanageable. By using inactive links and pseudo instance-set-icons, it is possible to visualize recursive relationships.

When users click an inactive link displayed for representing a recursive relationship type, a growing and shrinking view method is employed for showing all the set values of the pseudo instance-set-icon. The growing and shrinking view method allows users to see the values of the pseudo-instance-set-icon one-by-one. The growing view phase is to show all the set values of the pseudo instance-set-icon together. In the shrinking view phase, a set of displayed nodes is replaced by the original pseudo instance-set-icon.

Under this visualization scheme, users can see and browse aggregation and general relationships together. It is possible to display two or more relationships in a synchronized form at a time. However, the window for browsing is-a relationships is separated from the window for general relationships.

5 Implementation

We have built the icon-based GUI for browsing complex object structures on top of UNIX operating systems and the X11 environment[16]. In this section, we discuss the issues of implementing a new icon-based browsing method for navigating instance-level relationships.

5.1 A Two-Layer Architecture for Icon-Based Browsing

In this paper, we insulate a window view from an object view as Fig. 7. The window view means the visual display while the object view means the internal object structures as shown in Fig. 2(c). Fig. 7 shows a two-layered architecture which consists of an object view and a window view. The internal object structures can be created and updated by using API (or access) functions to an object-oriented database. On the other hand, the window view maintains window widgets for showing the icon-based display like Fig. 6. Therefore, the data structures for the object view differ from those of the window view.

To realize the two-layered architecture, we need to translate the object view into the window view. At this time, it is necessary to maintain **Browsing Control Information(BCI)** which is used for controlling window objects called widgets. That is, BCI is used for controlling

active/inactive links, inactive icons, instance-set-icon, and pseudo instance-set-icon. It is possible to change the screen by clicking inactive links, instance-set-icons, etc., without updating the object view. However, it is necessary to update BCI which has an effect on the move of widgets in the screen. This is a major contribution proposed in this paper.

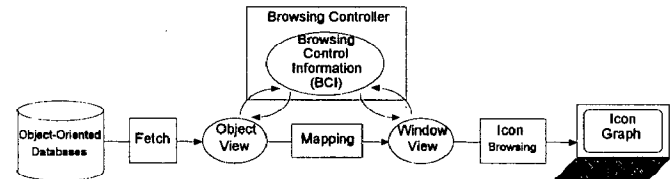


Figure 7. A Two-Layered Architecture for Icon-Based Browsing

In Fig. 7, the Fetch module is to access an object-oriented databases for creating the object view. The Icon Browsing module can be easily implemented by using X11 widgets. This paper doesn't deal with the Fetch module and the Icon Browsing module in detail. In this section, we focus on the two issues:

- how to map the object view into the window view.
- how to maintain Browsing Control Information for mapping between the object view and the window view.

To implement a DAG browsing algorithm, we used the Weighted Barycenters[4,5] which minimizes crossing and overlapping of arcs. Relationship browsing in this work was also implemented by using the STT algorithm[7] which supports automatic layout of large hierarchical graphs.

The internal object structures for representing the object view consist of linked list structures which are used for representing the accessed objects. We define a linked list structure SET_VALUE_LINK for representing a set of values as shown in Fig. 8. We also define a data structure SET_NODE as a head node of the linked list SET_VALUE_LINK. The SET_NODE corresponds to an instance-set-icon in the window view. When a SET_NODE is attached to a SLOT_NODE, the SET_NODE represents the values of a slot. Since an object may have one or more slots, a number of SLOT_NODES belonging to an object are represented as a linked list headed by a INSTANCE_NODE.

Fig. 8(b) shows a stored object view for describing object-level structures which represent the instance-level relationships of Fig. 8(a). The reasons why we define the linked list structure for the object view are as follows: First, we are able to easily represent a set of values in the object view. Second, it is simple to represent overlapping objects by using pointers. Lastly, linked lists are appropriate to realize instance-set-icons, for it allows users to view a set of values as a sequential order.

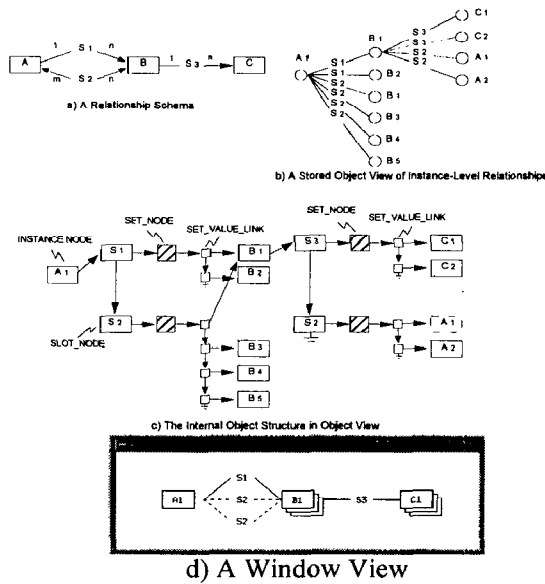


Figure 8. Mapping Example Between the Object View and the Window View

According to user's interaction, the window view can be changed while the internal object structures in the object view may not be changed. Translating Fig. 8(c) into Fig. 8(d) needs not only mapping algorithm but also BCI for controlling the window view. The window view consists of window objects which designate widgets displayed in the screen. In this paper, every icon is defined by individual widget. To realize the browsing of active links and inactive links, we implement links by Label Widgets[16].

5.2 Browsing Control Information

Because the window view differs from the object view as described in section 5.1, we need to map the object view into the window view. At this point, it is necessary to maintain information about window objects presently displayed in the screen as well as user's interactions by means of Browsing Control Information(BCI). In this paper, it is not possible to directly map the object view into the window view without using BCI. Using BCI is a major difference between the existing related works and this work.

As shown in Fig. 9, BCI represents the intermediate status information for mapping between the object view and the window view. When users click icons or links in the screen, BCI is changed first of all before window objects are updated.

To represent BCI, we use an adjacency linked list which consists of a *class-index-list* and a number of *slot-linked-lists*. The class-index-list describes information about the icon nodes. The reason why we describe the icon nodes as the header nodes of an adjacency linked list is for providing the fast access to a window object which is the source object of a specific link.

We define a *class index node* for describing each icon node. The class index node A_1 of Fig. 9(b) corresponds to the pseudo instance-set-icon A_1 of Fig. 9(a). If there are N icon nodes in the screen, we maintain N class index nodes in BCI. Because we show only one object among a set of objects which are the values of a specific slot, we need to record the currently displayed object. The field "currentobject" of a class index node represents the surface object currently displayed in the screen among a set of objects. The field "classname" of a class index node represents the domain name of the slot connected by the field "slotlink".

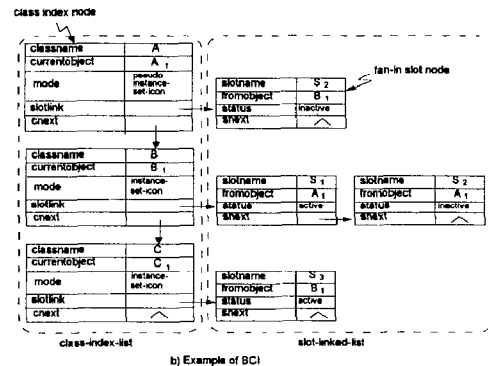
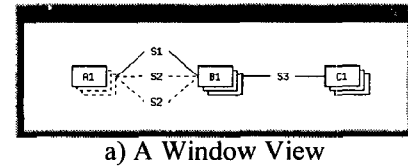


Figure 9. An adjacency Linked List for Representing BCI

In Fig. 9(b), we call a node of a slot-linked-list as a *fan-in slot node*. If a new slot has the same domain as the existing slot, we add a new fan-in slot node to the same class index node. The field "fromobject" of a fan-in slot node represents the source object of a given slot. The field "status" of a fan-in slot node represents whether a given slot is active or inactive link. When users click an inactive link, the status value of the selected fan-in slot node is changed. When the status of the fan-in slot node is changed, it is necessary to satisfy all the time the following constraints:

- 1) When there are multiple relationships between two objects or a given domain is shared by two or more slots of other objects, only one slot becomes an active link while the other slots become inactive links.
- 2) If there are cycles among icons, the recently created fan-in slot node of which domain corresponds to one of existing class index nodes becomes inactive.

5.3 Mapping Algorithm

By using information of BCI, we can decide whether a specific icon node for representing the values of a given

slot is displayed or not in the case of browsing the content of a given object. That is, if a given slot corresponds to an active link, both the active link and the related instance icons are displayed together. On the other side, if a given slot corresponds to an inactive link, the inactive link is displayed but the icon node for representing the slot value is not created

```

Algorithm Relationship_Mapping(
    (INSTANCE_NODE *node ), (integer node_mode))
SLOT_NODE *Slot;
class_index_node *ptr;
fanin_slot_node *rslot;
{
    if (IsDisplayedNode(node) or LongDistance(node) ==
        TRUE)
        return;
    else
        Create a Widget Object for the instance node with
            node_mode ;-- (1)
    for (Slot = node->slots; Slot != NULL; Slot =
        Slot->next) ----- (2)
    {
        if (IsUserDefinedClass(Slot) == TRUE) ----- (3)
        {
            ptr = FindClassIndex(Slot); ----- (5)
            if (ptr == NULL) ----- (7)
            {
                new(rslot);
                rslot->status := "active";
                rslot->fromobject := node;
                new(ptr);
                AddClassIndexNode(ptr);
                AddFaninSlotNode(ptr, rslot);
                Set_Node_mode(Slot->SET_NODE, node_mode)
                Create LabelWidget for an active slot link;
                AddQueue(Slot->SET_NODE->
                    SET_VALUE_LINK, node_mode);
            }
            else ----- (6)
            {
                new(rslot); /* create a new fan-in slot node */
                rslot->status := "inactive";
                rslot->fromobject := node;
                AddFaninSlotNode(ptr, rslot); /* add a new
                    fan-in slot node to slot_linked_list */
                Create LabelWidget for an inactive slot link;
            }
        }
        else ----- (4) /* in case of primitive types */
        {
            Create LabelWidget for making an active link;
            Create ICON Widget for representing the slot values;
        }
    }
}

```

```

While (queue is not empty) ----- (8)
{
    DeleteQueue(node, node_mode);
    Relationship_Mapping(node, node_mode);
}
}

```

Figure 10. Mapping Algorithm for Relationship Browsing

Fig. 10 shows a mapping algorithm for browsing the values of one or more slots of a given object. The argument "node" of the algorithm Relationship_Mapping represents an object to be browsed. The data structure INSTANCE_NODE is shown in Fig. 8(c). The argument "node_mode" represents a mode of a given instance node such as "instance_icon", "instance_set_icon", "pseudo_instance_set_icon", etc. The node_mode is determined by the function Set_Node_Mode().

Since we don't display the same icon two or more times in the screen, we need to check whether a given instance_node was already displayed or not. The function IsDisplayedNode() checks whether the current node was already displayed or not. If a given instance node is neither the currently displayed node nor the long distance node from the center of the screen, we create the widget object for displaying a given instance node(1).

Since one object may have one or more slots, it is necessary to repeat the creation of label widgets over each slot(2). First of all, it is necessary to check what is the domain of a slot because the icon for representing the value of a slot is unconditionally displayed in the case of primitive types such as integer, real, character, the etc(3). If the domains of a given slot are a primitive type, we create the corresponding active link and primitive icon widgets all the time(4). Instead, we don't create the corresponding fan-in slot node in the Slot_Index_list defined in BCI in the case that slot domains are primitive types. In addition, the class index node for representing the primitive type of a given slot is not created.

If there exists already the domain of a given slot in the class_index_list(5), the slot always becomes an inactive link(6). Therefore, it is easy to determine whether a given slot becomes an active link or an inactive link. The data structure fanin_slot_node defines the fan-in slot node of the slot-linked-list in BCI as shown in Fig. 9(b). The function AddFaninSlotNode() adds a new fan-in slot node to the slot-linked-list for recording a new inactive link in BCI. In the case of inactive links, we need to make only a labelwidget without making a corresponding icon node.

If there is no the class index node corresponding to the domain of a given slot in BCI, we need to make a new class index node and also to attach a new fan-in slot node to the class index node(7). The function AddClassIndexNode() adds a new class index node to the class-index-list. A new fan-in slot node is also attached to the class index node by calling the function

AddFaninSlotNode(). After making a label widget, we need to add the currently displayed node to a queue for continuously browsing the other objects connected to the current node. The currently displayed node, which is the value of a slot of a given object, is accessed by addressing such as Slot->SET_NODE->SET_VALUE_LINK as shown in Fig. 8(c). We use breadth-first browsing instead of depth-first browsing(8).

6 Conclusions

This paper describes the issues of designing and implementing an icon-based graphic user interface to manage the complexity of visual display for browsing complex object structures. We focus on simplifying the visual representation of complex relationships among instance-level objects by using a network of nodes connected by links. We present a new browsing method which uses instance-set-icon, active/inactive link, pseudo instance-set-icon, and inactive icon for simplifying the complexity of icon-based graphic user interface.

We also present a two-layered architecture for implementing the icon-based browsing of instance-level relationships. To realize the two-layered architecture, we maintain Browsing Control Information(BCI) for controlling window objects without changing stored objects according to user's interaction.

The main contribution of this paper is an incomplete node-and-link visualization scheme for simplifying the visual representation of complex object structures. Right now, the prototype system for browsing object-level relationships has been implemented by using the X11 Motif, but the facilities for direct manipulation have not been completely implemented yet.

References

- [1] Agrawal, R., et. al., "OdeView : The Graphical Interface to Ode", ACM SIGMOD, pp 34 - 43, 1990.
- [2] Byrce, D. and R. Hull, "SNAP: A Graphics-Based Schema Manager", Proc. IEEE 2nd Int'l Conf. Data Engineering, Los Angeles, California, Feb. 1986, 151-164.
- [3] Consens, M. P., I. F. Cruz, A. O. Mendelzon, "Visualizing Queries and Querying visualizations", ACM SIGMOD RECORD, Vol. 21, No. 1, March 1992, pp. 39-46
- [4] Davis, R. and Lenat, D. B. "Knowledge-Based Systems in Artificial Intelligence", New York: McGraw-Hill, 1982.
- [5] Davis, M. B., "A Layout Algorithm for a Graph Browser", Master's Project Report, Computer Science Division, EECS, UCB, Berkely, CA, May 1985.
- [6] Maier, D., P. Nordquist and M. Grossman, "Displaying Database Objects", Proc. 1st Int'l Conf. Expert Database Systems, Charleston, South Carolina, April 1986, 15-30.
- [7] Messinger, E., B., "Automatic Layout of Large Directed Graphs", University of Washington, Technical Report No. 88-07-08, July 1988.
- [8] Motro, A., "BAROQUE: A Browser for Relational Databases", ACM Trans. Office Information System. 4,2 (April 1986), 164-181.
- [9] Motro, A., A. D'Atri and L. Tarantino, "The Design of KIVIEW: An Object-Oriented Browser", Proc. 2nd Int'l Conf. Expert Database Systems, Tysons Corner, Virginia, April 1988, 17-31.
- [10] Parsaye, K., Chignell, M., Khoshafian, S., Wong, H., "INTELLIGENT DATA-BASES : Object-Oriented, Deductive Hypermedia Technologies", Wiley, 1989.
- [11] Riekert, W.-F., "The ZOO Metasystem: A Direct-Management Interface to O-O KB", ECOOP'87, pp 131 - 139
- [12] Shneiderman, B., "Direct Manipulation : A Step Beyond Programming Language", IEEE Computer 16(8), pp. 57 - 69, August, 1983.
- [13] Smith, D., C., et. al., "Designing the Star User Interface", BYTE, April, 1982.
- [14] Travers, M. "A Visual Representation for Knowledge Structures", Hypertext'89 Proceedings, pp 147 -158, November 1989.
- [15] Wu, C. T., "Development of a Visual Database Interface : An Object-Oriented Approach", Application of Object-Oriented programming, Addison-Wesley, pp 101 - 138, 1990.
- [16] Young, Douglas A., "The X Window System: Programming and Applications with Xt OSF/Motif Edition", Prentice-Hall, 1990.

Acknowledgment: This research was supported by the Korea Science Engineering Foundation under Grant No. 89010312033.