DBDesigner: A Tool for Object-Oriented Database Applications

Shuguang Hong
Georgia State University

Joshua Duhl and Craig Harris
Ontos, Inc.

DBDesigner is a graphical front end tool for ONTOS, a commercial object-oriented database system. Through its graphical user interface, DBDesigner allows an application developer to design, browse, and modify a database schema, and to generate C++ header files from the schema. For the purpose of testing and prototyping, DBDesigner permits the user to create, modify, and delete objects in a database. DBDesigner can be a stand alone tool for the design of object-oriented databases or as a library module that can be linked into an application program. The unique approach of DBDesigner is to integrate these activities into a working environment for the development of object-oriented database applications.

Introduction
Because of the large scale and complex structures of applications built on the top of object-oriented databases (OODBs), the user, an application developer, not only have to manage the database schema and a large number of objects generated by application programs, but also need routinely to browse, modify, test, and debug the database. To help the user to manage the complexity and to improve the productivity, computer-aided tools for the development of OODB applications are required. The design and implementation of such a tool must consider the following major issues.

(a) Schema and object management. A mechanism must be developed to help the user store, retrieve, and modify a large number of classes, objects, and relationships.

(b) Data representation. An appropriate visual representation for the database schema and objects must be selected carefully to match the needs of the user.

(c) Prototyping. Interactively, the user should be able to design and modify a database schema and to test the schema by creating and modifying objects. In addition, programming code should be generated from the schema that can be directly incorporated into application programs.

(d) Database query. Query capability must be provided to the user for extracting information from a database in an easy and natural manner.

(e) Customization. The system can be tailored to the needs of different users with respect to methodology, environment, and design goals (Reiner, 1988). For example, the user should be able to customize the visual display of the

DBDesigner is a graphical tool for the development of OODB applications based on ONTOS, a commercial object-oriented database system (Andrews, Harris and Sinkel, 1989; ONTOS, 1989, 1990). DBDesigner provides the user with a visual representation of database schema and objects. A working environment is created that allows the user to design, browse, modify, and test database schema and objects and to generate C++ header files from the database. The tool also can be used as a library module and linked into an application program. Figure 1 shows a typical application development cycle using DBDesigner.

In the rest of the paper, the section Related Work provides a short survey about related work. Then, the graphical user interface of DBDesigner is introduced, and followed by a section devoted to the design and implementation issues. Lastly, the conclusion is given. To aid discussion, the terminology used in this paper is compared to other terms that are commonly referred to in other literature.

<table>
<thead>
<tr>
<th>Class:</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property:</td>
<td>Attribute, Instance Variable</td>
</tr>
<tr>
<td>Procedure:</td>
<td>Operation, Method</td>
</tr>
<tr>
<td>Object:</td>
<td>Instance, Instance Object</td>
</tr>
<tr>
<td>Class</td>
<td>Generalization/Specialization, IS-A, or Super-Hierarchy: type (-Class) / Sub-type (-Class ) Relationship</td>
</tr>
</tbody>
</table>

### Related Work

Research prototypes based on graphical user interface for database systems have been reported in recent years, such as ADAM (Ellis and Demurjian, 1991), OdeView (Agrawal et al., 1990), RIDL (Troger, 1989), SNAP (Bryce and Hull, 1986), SIG (Maier et al., 1986), Gambit (Braegger et al., 1985), ISIS (Goldman et al., 1985), DBDT (Maryanski et al., 1985), LID (Fogg, 1984), SKI (King and Melville, 1984), and DDEW (Reiner, 1984). Since these systems were built mainly for research purpose, each of these prototypes focused on one or a few of the major design issues listed in Section 1. For example, LID concentrated on a graphical navigation methodology for database browsing. Although SNAP provided utilities for database browsing, schema designing, and simple query specification, it did not support prototyping and code generation. By contrast, SIG and OdeView permitted the user to construct a graphical display for databases, but other issues have been paid a little attention. ADAM allowed the user to design classes and relationships, and to specify database update propagations for which C++ code can be generated. But, database browsing and prototyping was not supported by ADAM. As a commercial product for solving real world problems, DBDesigner has addressed all major issues, such as schema design and modification, database browsing and prototyping, code generation, and tool customization.

The assumption of the user type is other issue that differentiates DBDesigner from most of the prototypes discussed above. For example, tools such as (Agrawal et al., 1990; Troger, 1989; Bryce and Hull, 1986; Braegger et al., 1985; Goldman, 1985; Fogg, 1984) assumed that the users are naive users. Based on such an assumption, those tools have emphasized "ease of use." Because the users of DBDesigner are application developers, the design of DBDesigner must offer and maintain a balance between ease of use features and the degree of sophistication.

DBDesigner permits the user to customize some screen layouts. But, unlike those systems that are for constructing graphical user interfaces, such as ET++ (Weinand et al., 1988), InterViews (Linton et al., 1989), MERCY (Stary and Messner, 1991), Ingrid (Guimaraes et al., 1991), DBDesigner is a graphical database design tool. Hence, it has a different set of objectives and does not provide the similar capabilities as that of those systems.

Tools have been developed for aiding object-oriented programming, such as ObjectWorks (1990), GraphTrace (Kleyn and Gingrich, 1988), and OOPE (Borras et al., 1990). Those tools provided object-oriented programming
environments with utilities for navigating data structures, browsing function invocations, incremental compilation, and debugging. But, these systems did not provide database functionality. DBDesigner is a tool for database applications and allows the user to start, commit, or rollback transactions, and to construct database queries.

**User Interface**

Figure 2 shows a typical screen layout of DBDesigner. The start up command window is at the top left corner of the screen and contains commands for initiating a browsing session, saving database modification or restore database to its previous state, printing help message, etc. A schema diagram window below the start up window displays a class hierarchy. A type window next to the schema diagram window on the right lists a class definition. Below the type window, an object window contains the values of an object. A display options panel is shown next to the object window on the left.

A browsing session begins with a selected subschema presented in a schema diagram window. The user can explore the database by directly manipulating the contents in a window, so called object-oriented user interface style. The manipulation can be:

- Browsing class hierarchies, class definitions, objects of a class, or values of an object,
- Creating classes or modifying class definitions,
- Creating, modifying, or deleting objects, or
- Generating C++ header files for selected classes.

After browsing the class hierarchy as drawn in Figure 2, for example, the user selects four classes of interest (highlighted) from the hierarchy and wants to examine those classes. DBDesigner responds to this request by popping up four windows, retrieving those classes from the database, and displaying their definitions in those windows, respectively.

**Design and Implementation**

**Integrated Working Environment Approach**

The common activities of OODB programming involves schema design, browsing, editing, and database testing and debugging (prototyping). Instead of designing a
separate tool for each activity, we decided to integrate these activities in a working environment. As pointed out by (Zlodnik and Maier, 1990), there is little agreement on the best ways to address the issues in the development of graphical tools for databases. But compared to other approach, our approach has the following advantages.

1. The user can perform all activities within a single system. Although each of those activities has its own requirements, they are logically interrelated. For instance, the user modifies a class definition only after that class is browsed and examined. Also, when satisfied with the definition of a schema, the user may want to test the schema by creating instance objects. Furthermore, those activities do not conflict each other. However, if those activities are separated into different tools, the user would have to switch frequently among those tools, which would be inefficient and annoying.

2. Coherent and uniform user interface style can be enforced easily across all system components. That is, the same screen layout and user-computer interaction style is used to communicate with the user to perform various activities.

3. The learning and training time required of the user is reduced. The user could begin with learning simple functionality and gradually progress to advanced features. After knowing how to browse the definition of a class in a window, for instance, the user could advance to the class modification by learning a new command in the same window. That is, accumulated learning is encouraged.

Database Navigation Model

An OODB usually contains a large number of classes, relationships, and objects. A navigational guidance is required to help the user locate information of interest easily and quickly. Relationship navigation has been used extensively as the means to guide database browsing. But, there are several kinds of relationships commonly accepted, such as generalization/specialization (IS-A or class hierarchy), classification (is-instance-of), association, etc. Because the IS-A relationship is an important and popular concept of object-oriented paradigm, we have decided to build the database navigation model around the IS-A relationships, i.e., class hierarchies. In DBDesigner, a browsing session begins with the display of a class hierarchy. The user can then explore the database by expanding or shrinking the class hierarchy, displaying class definitions or object values, or navigating in the database via various relationships from a class to other class, from a class to objects, from an object

Figure 3: Class Hierarchy Query
to other object, or from an object to a class by a simple selection and a mouse click.

To help the user manage a large number of classes and objects, DBDesigner has used the Directory facility of ONTOS. With the help of DBDesigner, the user organizes classes and objects into a directory hierarchy. This facility also allows the user to change freely from a directory to other directory and to set the database browsing scope and range.

Database Query Construction

Direct screen manipulation has been adopted as a way to construct queries. As discussed in the previous subsection, the navigation via relationships is a simple database query. Those queries retrieve classes or objects based on predetermined conditions. For example, the query,

*Select the advisor of student John Smith*

is formed by highlighting the value of Advisor field of the student object whose name is John Smith, and clicking a mouse button twice.

A complicated query condition can be specified in terms of display options. Figure 3 shows an option panel and a schema diagram window. The class hierarchy in the window is specified by the conditions set in the option panel that requests to display the user-defined subclasses of OBJECT down to 5 inheritance levels\(^1\). DBDesigner interprets the options as the query,

*Select all user-defined subclasses of the class Object in depth of 5 levels.*

Internally, DBDesigner provides two major means to construct queries from the user’s actions. The first method is used mainly for browsing class hierarchies. Abstractly, a class hierarchy is a special kind of graph, lattice, in which nodes are classes and arcs are inheritance relationships. Queries on class hierarchies are specified in terms of manipulations that add, prune, and traverse the nodes and arcs of a graph as well as to partition a graph into multiple graphs or to merge graphs into a single graph\(^2\), see (Hong, 1990) for detailed discussion. For example, the query,

*Select all user-defined or built-in subclasses of the class Object in depth of 3 levels,*

is constructed as the union of two graphs; one contains all user-defined classes; and another includes built-in classes.

Followed the similar direction as OdeView (Agrawal, 1990), the second method to construct queries is to implement operations that is similar to the projection and restriction operators of the relational algebra (Codd, 1970). The projection operation is used to select a portion of class definition or object values to display. Figure 4 shows the definition of BOOK and an option panel. The display options request to retrieve public and private definition of BOOK. Currently, a set of predefined projections has been

![Figure 4: Projection on Class Definition](image)
implemented. Those projections are divided into two categories, content and scope projections; the content projection partitions the contents of a class or object into properties, constructors, and procedures; and the scope projection divides the contents into inherited, local, public, protected, and private groups. Those projections can be combined using the AND Boolean logic to form complex projection expressions. The example in Figure 4 is translated into the query:

*Projection Book on public and private properties, procedures, and constructors.*

The restriction operation is used to select which object(s) should be manipulated. In the current version of DBDesigner, the restriction operation is performed only within a single class. That is, objects of a same class are selected. For example, select a group of manager objects. We have realized that this is a limited form of restriction. A plan has been made to remove the limitation in the next version of DBDesigner.

**Schema Design and Modification**

New classes can be created interactively through DBDesigner. The user can select superclasses first, and then form a creation method (constructor in C++) for the new class. The newly created class is added to a displayed class hierarchy, automatically.
The properties of an existing class, except built-in classes, can be modified. By default, the definition of a displayed class is read only to prevent any accident modification. The user must explicitly switch to the write model in order to make any change. After a class is altered, the user can either save or discard the changes. A very useful feature of DBDesigner is to propagate the change made in a superclass to all its subclasses. After the user requests the transaction commit, all changes become permanent in the database.

**Database Prototyping**

DBDesigner allows the user to create, modify, and delete objects. To create an object, an object template is prompted to the user for entering the object values. After the given values are validated, a new object is created. Figure 5 shows the template for creating a MANAGER object. The `Execute` command in the window issues the object creation request. Note that the entered values can be reused, with modification if necessary, for creating the next new object. This is a very efficient way to reduce repeated data entry.

In a manner similar to the modification of classes, the values of an object can be modified as the user wants. Figure 6 displays an EMPLOYEE object, John Smith, in a window in which this object can be browsed and modified.

**Data Representation**

Because our users are application developers, we have combined the diagram approach and the forms approach (Bryce and Hull, 1986; Rowe and Shoens, 1982;, Shu, et al., 1983) as the interface metaphors. Diagrams are used to draw relationships and forms are used to display classes and objects in a C++ like syntax that matches with the programming language, C++, used to develop applications on the top of ONTOS database.

Foreseeing the needs of displaying information in different programming language syntaxes beside of C++, we have separated the text syntax seen by the user from that which is used to store the schema and objects in a database. The system is responsible for converting information into the desired syntactic format between the user interface and the database. For example, ONTOS stores the declaration of a procedure in a `Procedure` object that uses separate structures for storing the procedure name, returned type, and the names, types, default values of arguments, etc. DBDesigner converts the procedure information into C++ syntax back and forth depending on the user’s request that is either data

---

**Figure 7: Different Class Hierarchy Layout**
A property value of an object can be a reference to other object. To display the values of an object, all object references of that object are replaced by the names of the referenced objects. We also define display formats for properties that have complex structures, such as dictionary, set, list, and array. For instance, assume that in an Employee object the Previous-Employers property is of type Dictionary that contains ordered pairs of Company and Address objects. This property value is displayed as a set of ordered pairs of Company and Address names. By selecting an object name or object names, the user can browse the actual object(s).

Screen Layout Design

To be able to display as much information as possible in the limited screen space without resulting in a crowded screen layout, we adopted three techniques to achieve our goal. One is to allow the user to close a window as an icon when the information contained in that window is not currently needed.

The second technique is to following the object-oriented design style. Each window is associated with a set of operations that are applicable to the contents displayed in that window. The class window in Figure 4 is an example. Instead of separating various functionality into individual tools (windows), this window is associated with operations for browsing and modifying the class definition, creating and browsing objects, and navigating other classes or objects via different relationships. This technique has not only reduced the number of windows, but also enhanced the object-oriented design style.

The third technique is to divide windows into permanent and transient windows and maintain only a few permanent windows. A permanent window may remain on the screen until either the user closes it or a browsing session is terminated. The schema diagram windows and class definition windows are permanent windows. A transient window stays for a short period and is dependent on a permanent window. For example, the display option panel of a class definition window in Figure 4 is a transient window. It pops up when the user needs to form a query on the contents of a class, and pops down after the query is processed.

4.8 Customization

The design of DBDesigner has taken customization into consideration. The first form of customization is to permit the user to customize the graphical representations of class hierarchies. Figure 7 shows three different representation of a same class hierarchy in which the orientation of the hierarchy and the shapes of the classes can be selected.

The second form of customization is to provide a simple programming interface so that DBDesigner can be linked to an application program. This interface consists of functions for initializing DBDesigner, starting a DBDesigner session, closing a DBDesigner session, and terminating DBDesigner. The stand along version of DBDesigner in fact is a simple application program that starts DBDesigner only once. In general, the application program can control when, how, and how often to start the DBDesigner.

Conclusion

DBDesigner is a commercial tool to support OODB programming. It provides a visual representation for database schemas. It creates an integrated working environment that allows the user to design and modify a database schema and to browse, and test the database. It also generate C++ header files from the database.

Several limitations of DBDesigner have been noted. One is lack of the capability for allowing the user to specify queries that have complex conditions. Also, it has no utilities to support schema evolution, versioning, and database administration. Since the first release of this product, plans have been made to study those limitations and to improve the functionality of the product.

ACKNOWLEDGMENTS

We would like to thank the engineering staff of Ontos for their valuable comments and suggestions. Special thanks to Tim Andrews, Bob Martin, Paul Martel, Lisa Molesworth, Jerry Thomas, and Wes Wilson for their contributions to the design, implementation, and testing of the DBDesigner.

References


Endnotes
1 Note that in the class hierarchy, two small boxes have been drawn aside of OBJECT class. They indicate that some superclasses and subclasses of OBJECT have not be displayed on the screen. The user can follow the indicators to reveal the hidden classes. In reverse, the user can hide a displayed class.
2 DBDesigner provides other operations for moving classes around in a displayed class hierarchy. But, they are not regarded as query operations.
3 If an object does not have a name, ONTOS always generates a unique name for that object.

Shuguang Hong is an Assistant Professor of Computer Information Systems at Georgia State University. Prior to that he was a member of the technical staff at ONTOS, Inc. His research interests include the object-oriented approach to data modeling, database design, systems analysis and design, and graphical user interface tools.

Joshua Duhl is the Product Marketing Manager at ONTOS, Inc. Prior to that he was a senior member of the technical staff. His interests include making technology work for people, creating and building and coaching high performance teams.

Craig Harris is a founder and vice president of research at ONTOS, Inc. His interests include object-oriented approach to natural language understanding.
Related Content

INDUSTRY AND PRACTICE: Another Look at On Time and Within Budget: An Agency Theory Explanation
www.irma-international.org/article/industry-practice-another-look-time/51221/

Enhancing the ER Model with Integrity Methods
www.irma-international.org/article/enhancing-model-integrity-methods/51223/

Enterprise Application Integration (EAI)
www.irma-international.org/chapter/enterprise-application-integration-eai/20769/

Practical Case Study of a Web-Based Tutor Payment System
Tanguy Chateau, Cecile Leroy, Johanna W. Rahayu and David Taniar (2003). Web-Powered Databases (pp. 88-120).
www.irma-international.org/chapter/practical-case-study-web-based/31425/

WebFINDIT: Providing Data and Service-Centric Access through a Scalable Middleware
www.irma-international.org/chapter/webfindit-providing-data-service-centric/4301/