A CASE Tool for Object-Oriented Database Design

Damla Turgut, Nevin Aydin, Ramez Elmasri and Begumhan Turgut

Department of Computer Science and Engineering
The University of Texas at Arlington
P.O. Box 19015
Arlington, TX 76019-0015
E-mail: {turgut,aydin,elmasri,bturgut}@cse.uta.edu

Abstract. There have been many CASE tools developed for designing Relational Database applications. However, not many CASE tools exist for designing Object-Oriented Database (OODB) applications. A key difference is that OODBs have methods or operations in addition to the data structures; hence, an Automatic Code Generation module is needed for such a CASE tool. This paper describes algorithms that embed the integrity constraints into the methods so that automatic constraint checking is done when any of the basic methods are invoked. The basic methods we consider are the constructor (for creating new objects of a class), destructor (for removing objects), modifier (for modifying attribute or instance variables), relator (for relating objects), and unrelator (for removing object relationships). The CASE tool has a Graphical User Interface (GUI) which takes an Enhanced Entity-Relationship (EER) schema as an input and creates a textual representation of EER schema objects. This is then transformed and stored in the CASE tool directory and an Automatic Code Generation program generates code skeletons for generic methods (operations) for the object classes. Automatic Code Generation program generates the code for the ODE object-oriented DBMS classes after mapping from the EER to an OO schema.

Keywords: Object-Oriented Databases, CASE tools.

1 Introduction

In recent years, there has been an upswing in the availability and use of automated tools for software development of all kinds. First-generation computer-aided software engineering (CASE) tools were oriented toward capturing and displaying the analysis and design information needed to support the traditional "waterfall" model of software development: requirements analysis, design, code development, testing, and maintenance [3, 11]. Second-generation CASE tools tend to be integrated into workbenches for greater life-cycle coverage [11]. Their data-driven approaches, coupled with automatic code and schema generation, help the user to avoid entering specifications at multiple levels.

Although architectures differ across CASE systems, in general such systems include the following [3]:

- A graphic and textual user interface, which allows the user to enter, browse, and modify design objects and to invoke tools and transformations,
- A central data and process dictionary (sometimes called an encyclopedia, or just a data dictionary), which tracks and controls design objects,
- A set of design tools, including diagramming aids, design analyzers, and prototyping and code-generation routines,
- A library of designs, both generic and previously developed, to serve as a starting point and source of inspiration for developing new designs.

This paper presents a prototype CASE tool [12] for Object-Oriented Database Design. The three components of the CASE tool are Graphical User Interface (GUI), EER/OO Catalogs, and Automatic Code Generation. The Graphical User Interface is implemented in Tcl/Tk [14, 10], a GUI builder language. The GUI was originally built as described in [8]. It has been modified by adding new features in order to make it more complete in terms of functionality of the EER schemas that it can handle. After the output of the GUI is finalized, two programs were written to convert the output of the GUI to the input of the Catalog and Automatic Code Generation programs [6], thus creating an integrated CASE tool. The Catalog program [13] takes the translated output of the GUI program and stores an EER schema description. The program then translates the EER schema into an OO schema and the OO schema is also stored in the catalog. The catalog uses ODE [1] as its storage system. The Automatic Code Generation program takes the EER schema and generates a list of corresponding OO classes and some of their basic methods, such as constructor/destructor for each OO class.

2 CASE Tool Architecture

The general architecture of the CASE tool [12] is shown in Figure 1. The GUI supports functionality in creating graphical EER objects, editing the objects, moving drawn objects around the canvas, deleting objects from the canvas, saving and loading a schema into and from a text file, and finally printing EER schemas in postscript format. The user can either load an existing EER schema, and modify it or create a new EER schema by utilizing the GUI.

The Catalog program takes the output of the GUI, which is a textual representation of an EER schema, and converts it into an OO schema. An EER schema description is also stored in the EER catalog. Once the EER schema is transformed to an OO schema, similarly the definitions for an OO schema are stored in the OO catalog. The EER catalog will consist of entities, attributes, relationships, categories, classification, and domain type concepts of the EER schema. Similarly, the corresponding OO catalog will contain the OODB-Class, attributes and methods concepts of the OO schema.

Automatic Code Generation program also does mapping from EER schema to OO schema before generating the code for the methods of the ODE classes. Both implicit and explicit constraints defined in the EER schema are incorporated in

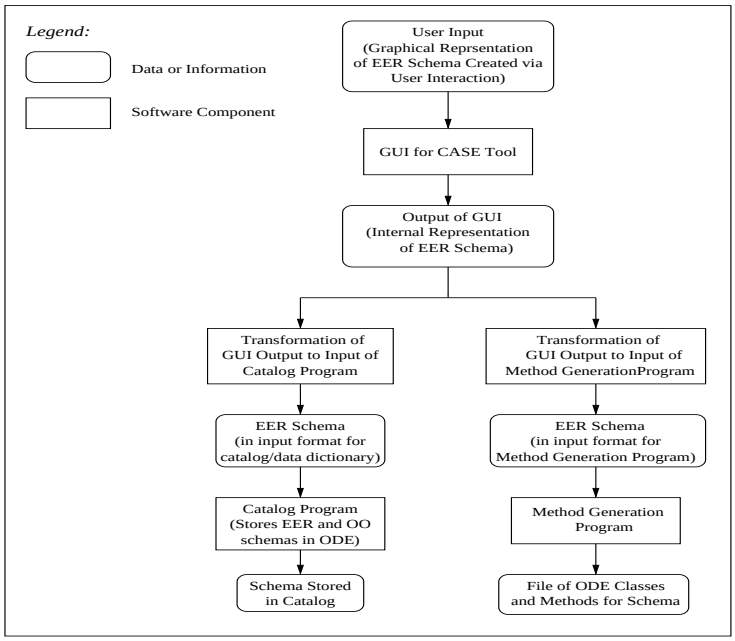

Fig. 1. Architecture Overview of CASE Tool

the code generation process. Some of the constraints checked include unique key constraint, structural constraint, and attribute cardinality constraint.

3 The Catalog Program

The system catalog is considered a "mini-database" [5] and its function is to store the schemas, or descriptions, of the databases that the DBMS maintains. The CASE tool uses ODE [9] to store catalogs for both EER and OO schemas.

3.1 ODE Database System Overview

The ODE [1, 7, 9] database is based on client-server architecture. Each application runs as a client of the ODE database. Multiple ODE applications, running as clients of the database server, can concurrently access the database. ODE also supports single user applications that can run without separate server and it consists of the O++ [2] compiler and the object manager library. Database applications are written in O++. The O++ compiler produces C++ [4] code, which is then compiled with a C++ compiler.

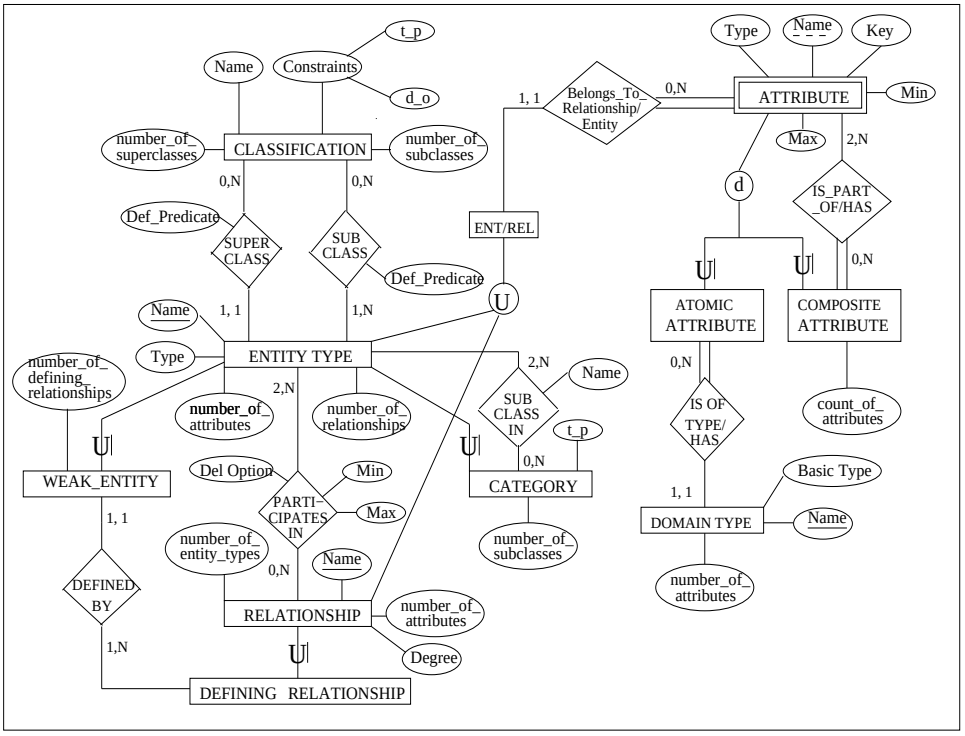

Fig. 2. EER Schema for EER Meta-data Repository

3.2 The EER and OO Catalogs

The EER catalog will store information about the EER schema as illustrated in Figure 2. The OO catalog for an OO schema will contain the information as shown in Figure 3. These follow the EER and OO models described in [5].

4 Graphical User Interface (GUI)

The Graphical User Interface is written in the Tcl/Tk [14, 10] programming language. Tcl/Tk is a scripting language specifically used for graphical user interface implementation. The original GUI program [8] has been modified and more functionality has been added to make it more complete in terms of the complexity of an EER schema that can be designed as input to the CASE tool. Some of the added functionality includes multi-valued attributes, *n-ary* relationships, min and max constraints for relationships, attribute value checking for null/not-null, and inclusion of the number of tokens in an attribute, relationship, and specialization/generalization list. The *n-ary* relationship is accomplished by

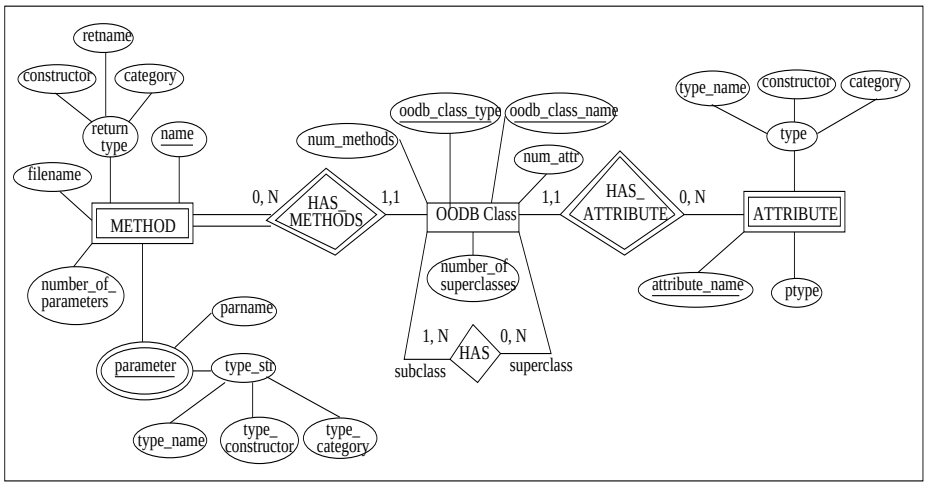

Fig. 3. OO Schema for OO Meta-data Repository

adding one relation at a time to an already existing binary relationship. The degree of the relationship will be called ternary for three participating entities in the relationship, and *n-ary* for more participating entities in the relationship, where $n > 3$. The degree increases as the number of participating entities are added to the relationship. Min and max constraints are required to be entered for the first and the second relationship, respectively, before the relationship is drawn between them. The attribute values are checked for null/not-null values every time an attribute is added. It is also checked when either the attribute value or the domain type has been modified. Null/not-null value checking can be reversed by modifying the attribute.

4.1 State Diagram for GUI

The state diagram for GUI is shown in Figure 4. The user can type "main.tcl" from the directory where the GUI files are currently located to start the GUI. The user can either open an existing schema or create a new schema. If the user chooses to open an existing schema, then the schema will be loaded and the user will be entered to the main state. On the other hand, if the user wants to create a new schema, then the user will either choose the new option from the "file" menu or simply start creating a new schema by selecting the necessary items from the icons palette located in the left side of the screen.

Once the user enters the main state of the program, the user can create a new schema or edit an existing schema by visiting many states. States includes

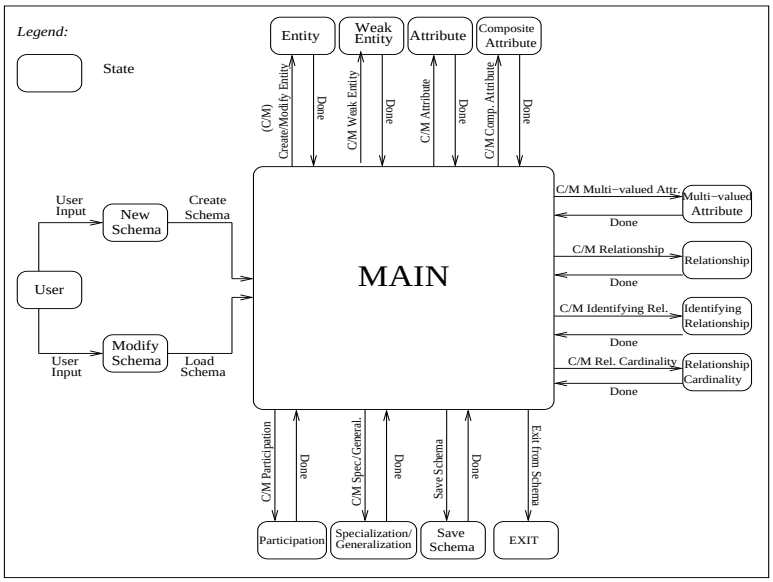

Fig. 4. State Diagram of GUI

entity, weak entity, attribute, composite attribute, multi-valued attribute, relationship, identifying relationship, relationship cardinality, participation, specialization/generalization, save schema, and exit from the GUI.

5 Mapping Tool and Automatic Code Generation

The Automatic Code Generation program [6] generates the complete code for the ODE methods. Prior to generation of the code, the EER schema needs to be mapped [13] to the OO schema. Thus, the output of the GUI, a textual representation of an EER schema, is converted into an OO schema. The O++ programming language is used under ODE database system.

5.1 Basic Methods for Object Model Classes

The basic methods that are automatically created are the following:

Constructor : Creates a new instance of an object
Destructor : Deletes or destroys a given object
Modifier : Allow changes to be made to the attribute values of objects
Relater : Relates objects in a relationship
Unrelater : Unrelates two objects in a relationship

Each of the basic methods have integrity constraint checking embedded in them so as to maintain the various constraints of the model among objects of classes

without relying on application programs to maintain them. This program defines algorithms that embeds the constraints in the methods so as to offer automatic constraint checking when any of these basic methods are invoked. The input specification [13] for the Automatic Code Generation program is given in [13]. The transformation program translates the output of the GUI into this input specification as part of the integration process.

5.2 The Basic Methods Constraints

The Unique Key Constraint specifies that the values of all keys of objects in a class extension are unique. Structural Constraint are specified using the (min, max) format for each participation of a class in a relationship. Attribute Cardinality Constraint specifies the minimum and maximum number of values multivalued attribute of each entity can have. The following constraints are common to all types of classes:

- The relationship participation and cardinality constraints specified via (min, max) may be violated in the insert, relating and deletion methods,
- The insert may violate a $min \geq 0$ constraint on relationship if the object does not have the minimum number of references required,
- In the relating operation, a class may violate the max cardinality constraint of participation in a relationship,
- In the deletion operation, a class could violate the min cardinality constraint if the number of objects fall below the minimum value specified,
- Unrelating an object could violate the min cardinality constraint.

The constraints checked in each basic methods are as follows.

Constructor: The implementations of unique key constraint, structural constraint, and attribute cardinality constraint are checked in constructor method.

Destructor: The implementations of structural constraint and referential integrity constraint are checked in destructor method.

Modifier: The implementations of successful modification of an attribute, violation of referential integrity constraints, and key constraints violated during modification are checked in modifier method.

Relater: The implementations of successful relating, and max constraint violation are checked in Relater method.

Unrelater: The implementations of successful unrelating, and min constraint violation are checked in Unrelater method.

5.3 Mapping of EER Model to Object Model

This section briefly describes how the EER model components are mapped to the Object model.

Attribute Mapping

Atomic and single-valued	atomic
Atomic and multi-valued	set
Composite and single-valued	tuple
Composite and multi-valued	set of tuples

Relationship Mapping

- Reference attributes of the related classes,
- Reference attributes can be single-valued or multi-valued depending on the cardinality ratio of the relationship,
- If single-valued, it is mapped to atomic reference attributes,
- If multi-valued, it is mapped to set reference attribute,
- $1:1$ Relationship: an atomic reference attribute is added to both classes of the relationship,
- $1:N$ Relationship: a set of reference attribute is added to the class on the '1' side of the relationship while the class on the 'N' side receives an atomic reference attribute,
- $M:N$ relationship: both classes receive a set reference attribute, each referencing objects of the other class.

Mapping Other Structures

Structures such as specialization's were modeled by using C++ type inheritance.

5.4 Process of Code Generation

The main steps followed in the process of code generation are now shown in order below:

1. The input file was parsed and persistent objects instantiated.
2. For each persistent Entity_Type object, class definitions were generated.
3. To generate the class definitions, the attribute objects stored for that entity_type were used.
4. Composite_Attribute class was checked to see if there are any composite_attribute objects for the entity_type, and corresponding attributes were added.
5. Relationship class was checked if there are any relationship objects for the entity_type, and the corresponding reference attributes were added.
6. Next, the constructor for each entity_type was written to the output incorporating all the constraints.
7. The destructor for each entity_type was written to the output incorporating all the constraints.
8. Modifier for each attribute was written.
9. Relater and Unrelater for each relationship of the entity_type was written.

5.5 Control Flow

This section shows the general control flow and the object model of the Automatic Code Generation program.

Start
    Read Input File
    Instantiate Objects
  Loop
    Generate for each "entity_type"
        Class Definition including method prototypes & attribute definitions
        Constructor with all the constraints
        Destructor with all the constraints
        Generate for each "attribute"
            Modifier with required constraints
        Generate for each relationship
            Relater with "max" constraint checking
            Unrelater with "min" constraint checking
  End of loop if next entity_type = NULL
Stop

6 Conclusions

This work describes the development of a CASE tool [12] for Object-Oriented Database Design by integrating three major program modules: Graphical User Interface (GUI), Catalog program, and Automatic Code Generation program.

The GUI component of the CASE tool has been enhanced by adding new features to the existing interface [8]. Added features includes multi-valued attribute, *n-ary* relationships, attribute value checking for null/not-null, min and max constraints for relationships, and inclusion of the number of tokens in an attribute, relationship, and specialization and generalization list.

Some of this added new functionality, such as min and max constraints for relationships and multi-valued attributes, has already been implemented in Automatic Code Generation program. The attribute value checking was the constraint needed by the Automatic Code Generation program. *N-ary* relationships were implemented to make the GUI more complete in terms of its advanced functionality. The transformation programs were written to convert the output of the GUI into the inputs of both Catalog and Automatic Code Generation programs. The programs generate two files that serve as the input for the Catalog program and Automatic Code Generation program. The Automatic Code Generation program creates skeleton methods for each class, with constraint checking. The methods include constructor, destructor, modifier (for each attribute), and relater and unrelater (for each relationship).

Possible extensions include compiling the GUI in a PC environment, since Tcl/Tk is a portable language. In addition, the code generators for other Object-Oriented Database systems can be added to the CASE tool.

References

1. R. Agrawal and N.H. Gehani, "ODE (Object Database and Environment): The Language and the Data Model," *Proceedings of ACM-SIGMOD 1989 International Conference Management of Data*, Portland, Oregon, May-June 1989, 36-45.
2. R. Agrawal and N.H. Gehani, "Rationale for the Design of Persistence and Query Processing Facilities in the Database Programming Language O++," *Proceedings of 2nd International Workshop on Database Programming Languages*, Portland, Oregon, June 1989.
3. C. Batini, S. Ceri, and S. Navathe, *Conceptual Database Design: An Entity-Relationship Approach*, The Benjamin-Cummings Publishing Company, Inc., Redwood City, 1992.
4. H.M. Deitel, P.J. Deitel, *C++ How to Program*, 3^{rd} Ed., Prentice Hall, 2001.
5. R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 3^{rd} Ed., Benjamin-Cummings, 2000.
6. R. Elmasri, S. James and V. Kouramajian, "Automatic Class and Method Generation for Object-Oriented Databases," *Proceedings of Third International Conference, DOOD'93*, Phoenix, Arizona, USA, December 1993, 395-414.
7. N. Geghani and H.V. Jagdish, "Ode as an Active Database: Constraints and Triggers," *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, September 1991, 327-336.
8. J. Juswaldy, R. Wardhana, "Project Report on Graphical User Interface for Object-Oriented Database Design", *University of Texas at Arlington*, 1995.
9. R. Arlein, J. Gava, N. Gehani, and D. Lieuwen, "Ode 4.0(Ode <EOS>) User Manual," AT&T Bell Laboratories, Murray Hill, New Jersey, 1995.
10. J.K. Ousterhout, *Tcl & the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994.
11. R.S. Pressman, *Software Engineering: A practitioners Approach*, 2^{nd} Ed., McGraw-Hill, 1992.
12. D. Turgut. "A CASE Tool for Object-Oriented Database Design". MS Thesis, *The University of Texas at Arlington*, December 1996.
13. D. Turgut, B. Ratakonda, K. Dawda, P. Desai, "Project Report on Mapping from EER Schema to OO Schema", *University of Texas at Arlington*, 1995.
14. Brent Welch, *Practical Programming in Tcl and Tk*, 3^{rd} Ed., Prentice Hall, 2000.

This article was processed using the LaTeX macro package with LLNCS style