ELSEVIER

Computer-Aided Design 35 (2003) 1161–1170

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Encapsulation of geometric functions for ship structural CAD using a STEP database as native storage

Junhwan Kim*, Soonhung Han

Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology, 373-1, Gusong-Dong, Yusong-Gu, Daejeon 305-701, South Korea

Received 2 September 2002; received in revised form 17 December 2002; accepted 7 January 2003

Abstract

It is difficult to support collaborative design with a conventional ship CAD system that manages design information using files. The file storage, however, can be replaced with a commercial database management system. This paper describes OpenDIS, which is an interface between the geometric modeling kernel and the DBMS. The main purpose of OpenDIS is to implement a CAD system that has the STEP database as the native storage. A prototype CAD system has been implemented using the OpenCascade geometric modeling kernel and ObjectStore, a commercial object-oriented DBMS. The STEP methodology is used for the database schema. This CAD system has been applied to the hull design process of a ship in order to verify the usefulness of the interface.

Keywords: STEP; Databases; Geometric modeling; Kernel modeling; Structures; Ship hulls

1. Introduction

In manufacturing industries, geometric model plays a crucial role among all the information generated and processed during a product lifecycle. However, it is difficult to manage the model throughout the lifecycle because design requirements and storage structures are different for each design stage or design department. Most conventional CAD systems have file-based storage structures. To shorten the time-to-market, the concurrent engineering concept has been adopted in the manufacturing industry.

Concurrent access of design information is necessary among heterogeneous CAD/CAM/CAE systems. Initial graphics exchange specifications (IGES), standard for the exchange of product model data (STEP) standard, and direct translation between systems have been used as the exchange mechanism. To support sharing of geometric information, CAD vendors provide interfaces between product data management (PDM) and CAD systems, or provide extra modules for collaborative design. Because they are based on a file system, the external reference between files or the relation between a part and an assembly should be handled by managing relations among files.

If the information of a product is distributed in several files, it is difficult to access only the required workspace and to re-store the data after design changes. A database management system is necessary to have the capabilities of concurrent access, data management at logical level, external reference, and change propagation. To construct a neutral database, standard schema such as STEP can be used. Also the technology to overcome the impedance mismatch [17] with the main memory structure is necessary.

Many CAD systems have been developed using geometric modeling kernels, which are collections of geometric algorithms and serve as engines of CAD systems. In this paper, a CAD system based on a geometric modeling kernel and a commercial database management system (DBMS) has been implemented. An interface has been proposed between the geometric kernel and the database. The data interface is between the persistent data of the database and the transient data of the geometric kernel. Fig. 1 shows the conceptual structure of OpenCascade database interface for STEP storage (*OpenDIS*), the proposed interface. OpenDIS plays the role of application program interface (API) or the connector between

* Corresponding author. Tel.: +82-42-869-3080; fax: +82-42-869-3210.
E-mail addresses: everwind@icad.kaist.ac.kr (J. Kim), shhan@kaist.ac.kr (S. Han).

0010-4485/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0010-4485(03)00020-4

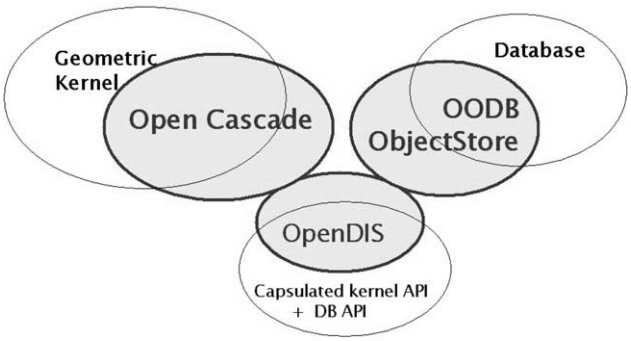


Fig. 1. Conceptual frame of OpenDIS.

the geometric kernel and the DBMS, which allows the development of a database-driven CAD system.

In Section 2, existing studies on the database-driven CAD and the STEP database are reviewed. Section 3 describes the OpenDIS internal mechanism, implementation process, and a test application.

2. CAD system which has a database as the native storage

2.1. Needs for database-driven shipbuilding CAD

To design the hull structure of a ship using a general-purpose CAD system, more than 100,000 files should be handled by the operating system. It is difficult to manage the names of all the part files, but the part names can be managed at the logical level in a database-driven CAD system. We want to store the part-level modeling information into a DBMS rather than as operating system files. The external references can be managed easily because the parts structuring is managed at the logical level. In a conventional CAD system, model files cannot be accessed concurrently. A database-driven CAD system has a database as its native storage and it allows concurrent accesses to the model data.

A strategy for data management has been applied to a complex design environment. This strategy enables the following: (1) implementation of a high level functional interface on top of the geometric modeling kernel to manage the database objects; (2) retrieval of only the needed parts from the database of the hull CAD; and (3) change propagation after modifying the CAD database.

In the mechanical engineering domain, an assembly structure is constructed by collecting sub-assemblies and a sub-assembly is constructed by collecting parts. A part is a unit that has an independent function. A working unit of the initial design stage is a *part* and a working unit of the detail design stage is also a *part*. The working unit of the initial design is the same as that of the detail design. However, in the shipbuilding domain, the working unit of the initial design stage is not the same as that of the detail design stage.

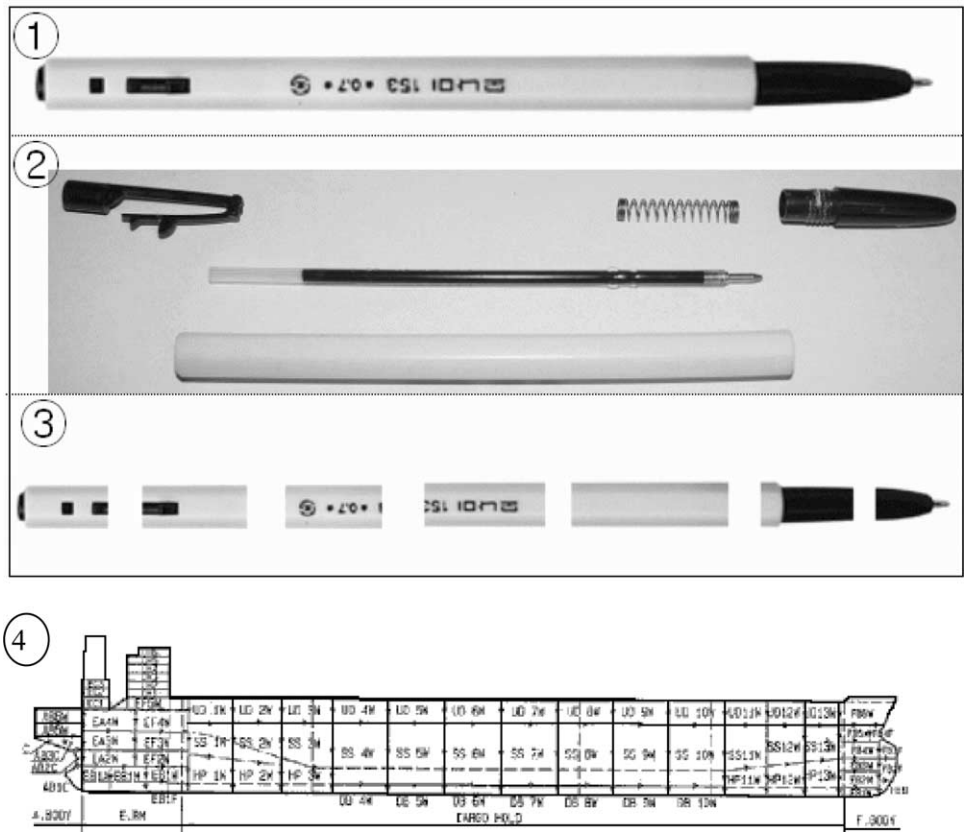


Fig. 2. Mechanical assembly (ball-point pen) and ship structure.

Table 1
Difference of working units between MCAD and ship CAD

	Initial design	Detail design
Mechanical	Part	Part
Shipbuilding	Functional system	Manufacturing block

The ship must be divided into blocks that are small enough to be loaded by cranes of the dry dock or building berth. The working unit during the detail design is a building block while the initial design stage focuses on functional systems such as piping, ventilation, longitudinal structural strength and propulsion, the system of which spans the whole ship length.

A ballpoint pen is shown in Fig. 2 (1) as an example of a mechanical product. It consists of parts such as lead, a cover and body. The assembly structure is shown in Fig. 2 (2). The cover comprises one part and does not need to be divided into several pieces for manufacturing. The ballpoint pen is made of functional units and each functional unit is a manufacturing unit. The pen does not need to be manufactured, as shown in Fig. 2 (3). However, in the shipbuilding domain, a functional unit is not a manufacturing unit, and splitting units into blocks is necessary to build a ship, as shown in Fig. 2 (3) or Fig. 2 (4).

One block of a ship is so huge that it cannot be produced at one time. It should be assembled from pieces and welded into a block. The result of the initial design is made of functional parts. The functional part cannot be manufactured as one piece because of its size; for example a pipe is several hundred meters in length. Such components should be assembled from several short pieces.

Table 1 shows differences between the working unit of a mechanical design and ship design. The unique issues related to the working unit of a ship design create the data management problem. Some mechanical CAD systems use DBMSs to interface with PDM systems and they provide database management modules. However, the geometric information is not handled by the DBMS and is stored in a file.

2.2. Related research

Research applying DBMSs to CAD data management began in the 1980s. In the late 1980s, Meier [1] and Harder [2] applied a relational database to solid modeling. They described the limitations of storing solid models into a relational database and presented methods for efficient retrieval. Kim [3] showed that an object-oriented database is efficient for CAD environment. But for GSCAD [4,23] of the GRAD consortium a relational database is used for better performance.

The methods used in previous studies are compared in Table 2. GNOME [4], Graph Support [5], and OsconCAD [6] are systems that link an object-oriented database with a geometric modeling kernel. The GNOME geometric engine has been implemented using object-oriented modeling methodology. Database functions such as query and version control are supported for CAD actions. OsconCAD has been developed using the AutoCAD API. It does not use a file system but instead an OODB is used for the storage of CAD models.

Autokon, a shipbuilding CAD, employs a network type database called Tornado [7]. It is designed up to the physical level database. Its API is in the system level and it does not support concurrent control. GSCAD, a new shipbuilding CAD system [13,23], is based on a MS-SQL relational database. It uses the ACIS geometric kernel and provides a short transaction concept.

There have been several studies about STEP database implementations [8–11] and utilization of STEP in the shipbuilding industry [12,14,19,21]. Table 3 compares methods to implement the interfaces between (1) a modeling kernel and database; (2) a modeling kernel and STEP; (3) STEP and database. High-level data access interface (HLDAI) [16] does not directly deal with the application interpreted model (AIM) of an STEP application protocol (AP); instead it develops an application program model (APM), which is an application-oriented model and a higher level interface than standard data access interface (SDAI). In this paper, STEP is used for the database schema, and the database manipulates geometric information, features, domain specific entities, and their properties. The required entities are selected for the application interface.

Table 2
Related research on geometric modeling with DBMS

	This research	GNOME [4]	Graph support [5]	OsconCAD [6]	GSCAD [13,23]
Target data	Ship structure data in AP218	Detail geometry	Feature part	Architecture, all domains	All data
Database type DBMS system	OODB ObjectStore	OODB ObjectStore	OODB Objectivity	OODBObjectStore	Relational DB SQL Server
Native format	Database	Database	File and Database	DXF	Relational database
Geometric kernel interface	OpenCascade	Own kernel	Parasolid	None, AutoCAD API	ACIS
Commercial CAD interface	Unigraphics, solidworks	None	None	AutoCAD	Intergraph

Table 3
Rules for the OpenDIS implementation

Mapping target	Rules
STEP and DB	STEP schema maps to the database schema on a one to one basis
Geometric kernel and STEP	Attributes of the *capsulated geometric API* are the same as attributes of the DB schema
Geometric kernel and DB	By calling the *capsulated geometric API* together with the database function for the instance creation, they can be synchronized

2.3. Structure of DINA-CAD

Fig. 3 shows the process of system implementation. The database schema has been created by referencing the application protocol 218 (AP218) schema [15]. The OpenDIS interface has been created by referring to a portion of the AP218 schema. The prototype CAD system DINA-CAD has been developed based on the OpenDIS interface. DINA-CAD consists of a database server (left) and clients (right upper), as shown in Fig. 4. It also has a module for data bulk load using the *Rose Library* of STEPTools, and the database can be accessed from commercial CAD systems. DINA-CAD has basic functionalities such as creation, modification, deletion of geometry, and advanced functions. We have implemented multiple-view access according to the ship design needs. Multiple-view of the design information can be achieved by the filter module. The *Access Filter* in Fig. 4 supports queries by *panel property*, *region*, *object type*, *plate relation*, and Fig. 7 shows some results of the multiple-view.

As shown in Fig. 4, the following features are noted: (1) File storage of the CAD system has been substituted by a commercial DBMS. The database is not an auxiliary medium, but the main storage. (2) It is possible to query

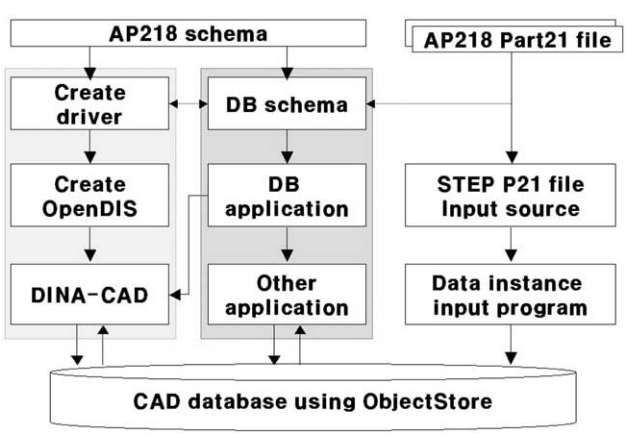


Fig. 3. Procedure of the system implementation.

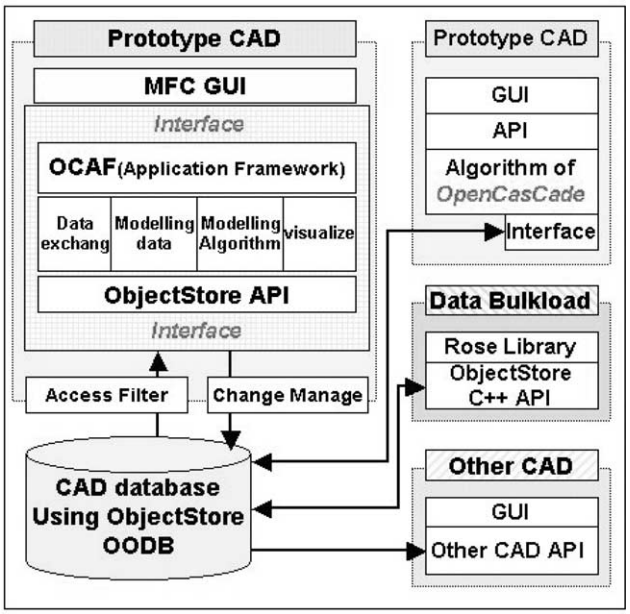


Fig. 4. System architecture of DINA-CAD.

by geometry or query by property. (3) The database has been implemented based on the STEP standard schema. (4) DINA-CAD can support collaborative design through the concurrent access of data. (5) It is possible to access the model data not only from the DINA-CAD system but also from commercial CAD systems. (6) It can support the initial functional design as well as the detail production design. (7) The database is used for manipulating relations between entities and also for change management.

Existing STEP databases [8–11] are not different from the file storage systems and their schemata are for file upload and download. Partial change of a file is difficult because the STEP files are generated from CAD translators. In this paper, STEP data can be generated and modified through the interface. The STEP database is used as the CAD system's main storage, which allows partial creation and modification of models. Databases of commercial CAD systems have own internal data storages which are designed for the internal data structures. Databases made with the STEP schema are external databases only for data sharing. The database proposed in this paper has both aspects, because it is not only the native storage of the CAD system but also a medium of data sharing.

In this research, the functions of the geometric kernel are fully utilized, as in the work of Jacobson [14]. The prototype system uses the commercial geometric kernel OpenCascade [20] and the commercial object-oriented DBMS ObjectStore. STEP is used as the database schema. This study concentrates on an interface to implement a database-driven ship CAD system that combines existing technologies [18].

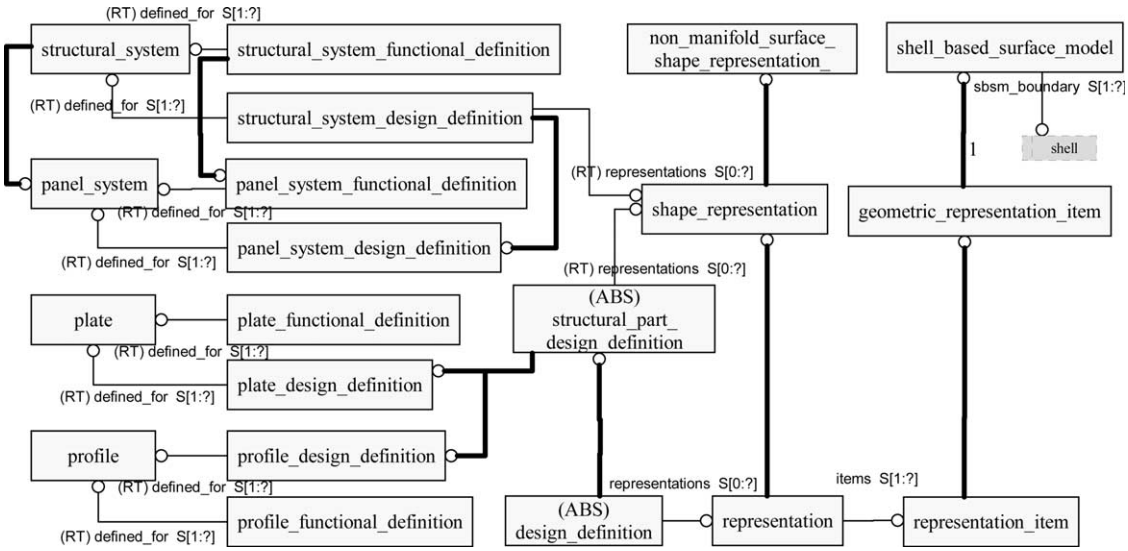


Fig. 5. Partial schema of AP218 (generated using EDM visual Express of EPM [24]).

3. OpenDIS—interface between geometric modeling kernel and DBMS

3.1. Linking modeling kernel with STEP database

OpenDIS is the interface between a database and a geometric kernel; it allows synchronization between the persistent data in the database and the transient data in the main memory. OpenDIS offers an interface for STEP database storage. We have adopted STEP as the database schema because it supports several design stages and reflects many necessary constructs of the industrial domain. AP218 is an AP to define and exchange the structural design information of a ship. The application reference model (ARM) of AP218 is used for the implementation [21].

Fig. 5 shows part of the AP218 schema. The entity *structural_system* consists of *panel_systems* and the entity *panel_system* consists of the entity *plates* and *profiles*. Each *system* (or *plate*) refers to the entity *functional_definition*, which defines functionality, and also the entity *design_definition*, which defines the geometric shape. *Plate_design_definition* contains the 3D geometric plates information. The 3D geometric model of AP218 refers to STEP Part 42. AP218 uses the following four *shape_representation* types of the entity *representation*: *edge_based_wireframe_shape_representation*, *geometrically_bounded_wireframe_*, *non_manifold_surface_*, and *advanced_brep_*. We have used the entity *non_manifold_surface_shape_representation*. The entity *geometric_representation_item* of AP218 employs a *shell_based_surface_model*, *face_based_surface_model* and an *edge_based_surface_model*. Generally, the *shell_based_surface_model* and the *face_based_surface_model* are used to represent the ship model. The *shell_based_surface_model* is used in this paper. In summary, the *plate_design_definition* entity has a *non_manifold_surface_shape_representation* entity as its *representation* and the *non_manifold_surface_shape_representation* entity has a *shell_based_representation* as its *representation_item*.

Three data structures used in this study are the STEP schema, the database schema of ObjectStore and the data structure of the geometric kernel. OpenDIS links the database schema to the data structure of the geometric kernel. OpenDIS implements the application program, which has the functions of creation and modification of geometric entities, and storing and retrieving them into the database. OpenDIS is defined as the 'encapsulated modeling functions that map and translate geometric entities'. Fig. 6 shows the relations among the APIs which organize the OpenDIS. This figure shows the API of the OpenCascade geometric modeling kernel (GMKA), the driver, the *capsulated geometric API* and OpenDIS.

To link the three elements of STEP, database and the GMKA, first the *driver* is implemented to link STEP and the modeling kernel where GMKA is used. Second, to link the STEP and database, the STEP schema of AP218 is used. Since both the geometric kernel and the database are linked by STEP AP218, all three elements are linked. The *STEP compatible layer* contains the *driver*, STEP AP218 and database. The *capsulated geometric API* is implemented using the *driver*. If an entity is created by the *capsulated geometric API* then that entity is STEP AP218 compatible. OpenDIS can concurrently call the *capsulated geometric API* and the database API, which link STEP, the modeling kernel and the database.

To link the database with STEP an ObjectStore class having the STEP class as its member has been defined. The database entity has a one-to-one relation with the STEP entity. For the design of the database schema, existing mapping methodologies are used [10,22]. The naming rule

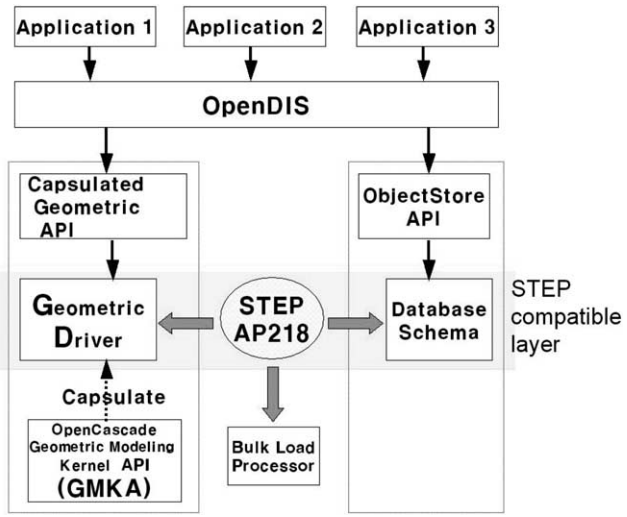


Fig. 6. Relationship between APIs.

for mapping is as follows. The entity that is the geometric entity and at the same time an ObjectStore entity has the prefix DB_ in its name in order to remove confusion with the name of the STEP entity (Table 4). Non-geometric STEP entities use the same names as the schema. For instance, there are some geometric entities such as *DB_Line*, *DB_BsplineSurface*, and non-geometric entities such as *Plate_design_definition*. As shown in Table 3, attributes of the capsulated API are equal to attributes of the database schema.

As STEP is used as the database schema, the database can be called a STEP database, and the CAD database of this paper is the STEP database. Table 3 shows the basic rules of implementing OpenDIS. There are two interfaces between data entities. Internally OpenDIS calls both the OpenCascade API functions and the ObjectStore API functions. A single OpenDIS function calls the *capsulated geometric API* to create and modify geometry and also calls the capsulated ObjectStore API to store the geometry. Arguments of the geometric kernel function, *TS_Commands::Create_Line* and the constructor of the database class, *DB_Line* are the same. Since the function that creates the data structure of the GMKA and the function that creates the database instances are called at the same time, they can be synchronized.

Commercial CAD systems have their own APIs. Unigraphics has OpenAPI and ProEngineer has Pro/Toolkit. OpenDIS is the API of DINA-CAD, which is the prototype CAD system implemented through this research. Whereas the API of a commercial MCAD system provides a data file access module, OpenDIS offers a database access module.

3.2. Implementation of OpenDIS

The implementation procedure is as follows.

1. Select an entity that can be instantiated and find the corresponding geometric kernel function.
2. Create the *driver*.
3. The *capsulated geometric API* is implemented by using the *driver*.
4. Create *OpenDIS* by combining the *capsulated geometric API* and DB functions.
5. DINA-CAD is implemented by using OpenDIS and adding the user interface program.

Using an OpenCascade geometric function such as *BrepBuilderAPI_Make_Line*, a *driver class* such as *DinaCAD_LineDriver* is implemented. Using the *driver* the *capsulated geometric API* is implemented. For example, when *capsulated geometric API* such as *Create_Line*, *Modify_Line*, *Delete_Line* are implemented, *DinaCAD_LineDriver* is used. By linking the *capsulated geometric API* with the ObjectStore database API, the OpenDIS can be implemented.

At the first stage, entities that can be instantiated are selected. They can be selected from the STEP entities to support domain specific applications. For the prototype CAD system, the target domain is the shipbuilding industry, and the sample model is the midship section of a ship. To implement the system the basic entities for the prototype system are selected as shown in Table 5. Table 5 shows the range of entities that can deal with the midship section.

Table 6 shows examples of *persistent* data and *transient* data. OpenCascade class entities such as *TopoDS_Line* and

Table 4
Relationships among attributes of OpenDIS function, database entity, and STEP entity

OpenDIS function	DB entity	STEP entity

Void ODIS_Create_Line (int ID,
DB_cartesian_point pnt, DB_direction vector)
{
TS_Commands::Create_Line (ID, pnt, vector);
new DB_Line((ID, pnt, vector);
}
	Class DB_Line {
	int ID;
	
	DB_cartesian_point pnt;
	DB_direction vector;
	}
		ENTITY line
		SUBTYPE OF (curve);
		pnt: cartesian_point;
		dir: vector;
		END_ENTITY;

Table 5
Entities which have corresponding *geometric drivers*

Entity type	Entity name
2D entity	PolyLine*Driver*, Circle*Driver*, Line*Driver*
Surface entity	Bspline_Surface*Driver*, Bezier_Surface*Driver*, Plane*Driver*
Solid entity	Box*Driver*, Cylinder*Driver*, Cone*Driver*
Non-geometric STEP entity	Plate_design_definition*Driver*, Non_manifold_surface_shape_representation*Driver*

gp_line are present in the geometric kernel, but they are not managed as *persistent* data. If needed, *TopoDS_Line* and *gp_line* entities can be stored using the *DB_Line* entity of OpenDIS; otherwise they are managed in memory and then abandoned. *Plate_design_definition* is an example of an entity that is both *transient* and *persistent*.

At the second stage, to create and modify the geometric entities, *drivers* are created; they are the capsulated *geometric kernel functions* of geometric entities. OpenCascade's OpenCascade application framework (OCAF) module is used to make the capsulated *driver*. There are two rules to implement a *driver*. One is that the external attributes of the driver should be equal to the attributes of database instances. The application programmers who use the *driver* do not need to be concerned about the internal structure. They can make interfaces between the geometric kernel and the database using the same attributes. The other rule is that the *core geometric functions* must be selected among the low level geometric functions of the OpenCascade kernel. For example, the *LineDriver* uses the geometric modeling function *BrepBuilderAPI_Make_Line*.

At the third stage, the *capsulated geometric API* is created by using *drivers*. For example, *Create_Line*, *Modify_Line*, *Delete_Line* are created by using the *LineDriver*. At the fourth stage, OpenDIS functions such as *ODIS_Create_Line* in Table 4 are generated by joining the *capsulated geometric API* and corresponding database functions. At the final stage, the GUI is implemented and the DINA-CAD system is completed. Fig. 7 shows the class hierarchy of the DINA-CAD client program. This structure consists of several parts, the GMKA, the *driver*, the *capsulated geometric API* (*Capsulater* in Fig. 7), OpenDIS, database API, and Microsoft foundation class (MFC) for the GUI. Examples of *PlaneDriver* and *BSplineDriver* show the relationships among classes.

Table 6
Examples of persistent vs. transient entities

Persistent	Transient
Plate_design_definition	Plate_design_definition
DB_Line	TopoDS_Line, gp_line, Line

The *driver* is made by inheriting the *Tfunction_Driver* class of the OCAF module. The *driver* is the interface to create and modify the entity instances based on the AP218 schema and to allow undo, redo, and save. Fig. 7 shows that the *core geometric function* of *PlaneDriver* is derived from the OpenCascade API function *BRepBuilderAPI_MakeFace*. The GUI classes use the MFC and have basic functions such as zoom, pan and rotate. The dialog box for property query is a part of the GUI class.

The working mechanism of classes is as follows. When a user initiates a job through a client GUI, the *CDinaView* class deals with the GUI functions and the *CDinaDoc* class executes the user level modeling functions. At this time, the *TS_Command class*, which is in charge of all the *capsulated geometric APIs*, is instantiated and member functions related to geometry are executed. These *drivers* are derived from the *TFunction_Driver* class for every geometric entity. After the *driver* is instantiated, the *Modify_* member function is called to modify the geometric entity. The *Create_* member function creates geometric entities and stores the data into the database. The *Get_* member function

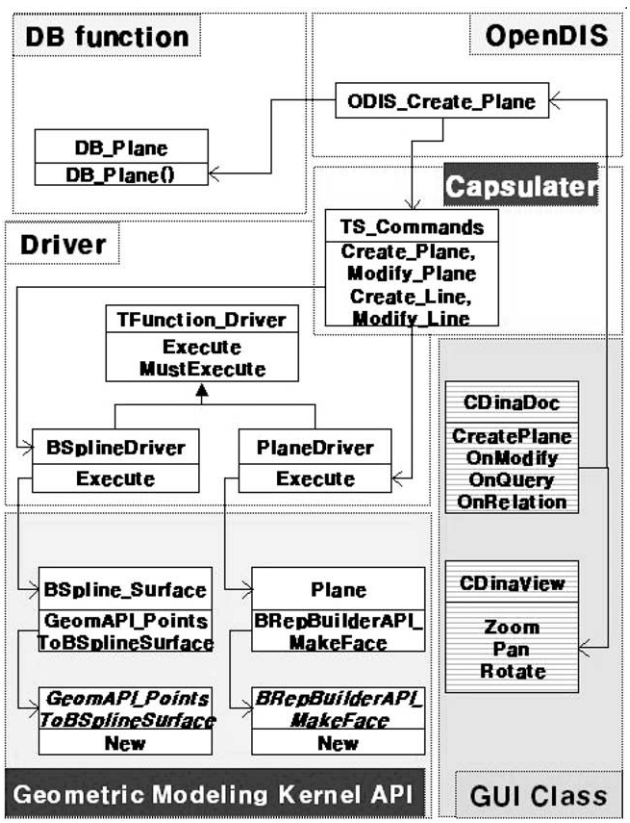


Fig. 7. Relationships among DINA-CAD classes.

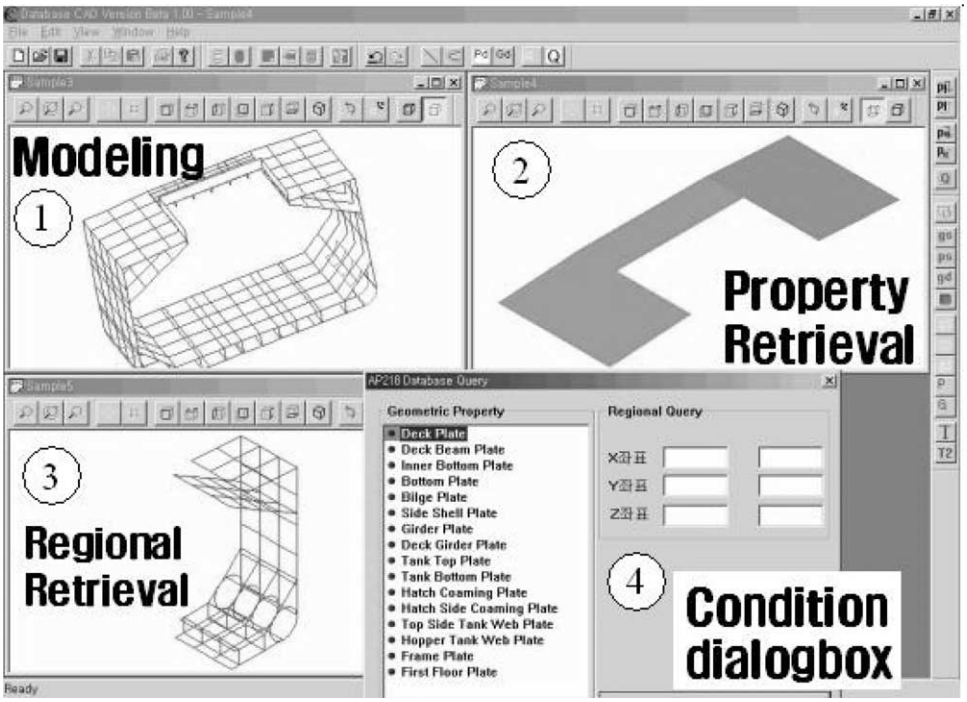


Fig. 8. GUI of the DINA-CAD client.

retrieves the model from the database and visualizes it on the GUI.

3.3. Experiments with the prototype CAD

Based on the *Capsulated API* the prototype CAD system, which adopts the STEP AP218 schema, has been implemented to show the feasibility of OpenDIS. Using this methodology, CAD users can make their own CAD systems. An application programmer does not need to know the details of the kernel, and by using only OpenDIS he can develop his own module of a DB-based CAD application. Fig. 8 shows the graphic user interface (GUI) of the DINA-CAD client. Modeling functions for primitive geometries are implemented and *undo* and *redo* are also implemented. The test model is the midship section of a ship. It consists of 25,000 entities from 400 parts. Fig. 8 (1) shows the midship section modeled by the DINA-CAD.

From the viewpoint of the database, the geometric modeling transaction can be divided into several processes as shown in Table 7. To start modeling the designer initiates the database, prepares the *TypeSpec* for type generations, opens the working database, and retrieve the workspace instead of opening part files of the operating system, which is the case of a conventional CAD system. When a transaction is started, geometric instances are stored as collections such as a list, set, or bag based on the database root. OpenDIS has *undo* and *redo* facilities at the *user command* level. A modeling transaction consists of executions of the OpenDIS functions. If the designer issues the *save* command from the GUI, one session of the database transaction is finished. After the modeling job is finished, the database is closed.

Conceptually, data in the database server is equal to one large file. The database makes it possible for a designer to access only the necessary portion and controls concurrent access from client applications.

The system is supported by the *plate* object; that is, a unit of information, and its related entities are shown in Fig. 5. Property of plate, geometry of plate, relationship between plates and functionality of plate are the main

Table 7
Process of a geometric modeling transaction

	Work	ObjectStore function
Initialize	DB initialization	ObjectStore::initialize();
	Collection initialization	os_collection::initialize();
	TypeSpec definition	new os_typespec('db__line');
	DB open	os_database::open('DB');
Modeling	Transaction start	OS_START_TXN(tx1)
	Finding root	db1- > find_root
	OpenDIS execution	ODIS_Create_Line or ODIS_Modify_Line or…
	Execution of visualization functions of the geometric kernel and setting properties based on the data structure of the geometric kernel	
	Transaction stops	OS_START_TXN(tx1)
Finish	DB close	os_database::close (DB);

items. *Plate_design_definition* and *Plate_functional_definition* refer to *plate*, and the *plate* entity has properties such as *deck plate* or *inner bottom plate*. Fig. 8 (2) shows an example of a *deck plate*.

Fig. 8 shows examples of concurrent access to the design model by different designers. Designers from different stages can access necessary parts. A detail designer can access a structural region when several initial-designers are processing their jobs. Fig. 8 shows the result of a test of *query by region* and *query by property*. The DINA-CAD opens the deck plate, which is the upper part of the midship section. Fig. 8 (3) shows the result of a *query by region*. The DINA-CAD retrieves the model inside the bounding box determined by the XYZ coordinates. Fig. 8 (4) shows the dialog box to input the query conditions.

In a large-scale STEP database, performance is an important issue. Loffredo [10] evaluated the access capacity of STEP databases based on Oracle, ObjectStore and OpenODB. In a DB based CAD system, regardless of the schema that is used, the type conversion problem remains. The loading capacity for the model of Fig. 8 (1) is about 600 objects/s in a PC with 500 MHz CPU and 256 MB memory. The NURBStone-benchmarking test of Loffredo reported 1000 objects/s in his ObjectStore environment [10]. In this research, the size of the test data is less than 30,000 entities. It is necessary to deal with data comprised of over 1,000,000 entities in the shipbuilding domain.

4. Conclusions

Conventional CAD systems with file-based storages cannot provide concurrent access, data management at the logical level, external reference, or change propagation. A commercial DBMS has been used for the data storage of the pilot shipbuilding CAD system. To solve the impedance mismatch problem between the persistent data of the database and the transient data of the geometric kernel, capsulated functions are used to interface the geometric kernel and the DBMS. This paper describes OpenDIS, an interface between a geometric kernel and a DBMS and its development procedure.

The storage structure of the prototype CAD system is not only the transient storage of the CAD models but also the persistent data repository for design information sharing. The STEP AP218 schema, the geometric kernel OpenCascade and the object-oriented DBMS ObjectStore are combined to develop the CAD system DINA-CAD. The geometric modeling functions are encapsulated and linked with the database API, and synchronization between the kernel and the database has been implemented. A pilot experiment tested the feasibility of the interface by applying the interface to the shipbuilding domain. The structural model of a midship section has been generated, stored, queried (*query by property*, *query by region*), modified, and stored back again into the DBMS.

In this research, the AP218 schema is used as the database schema, but other APs of STEP can also be used as the database schema. This interface can be used for other applications to develop a CAD system with a concurrent design environment, for selective access of design workspaces and data sharing with other commercial CAD systems. In the future, a comparative study is needed between RDB and OODB in terms of performance.

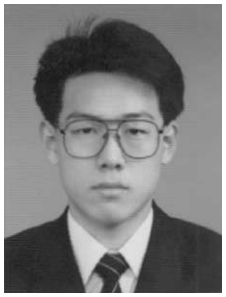
Acknowledgements

This research has been partially supported by the Korea Maritime STEP project (NRL).

References

[1] Meier A. Applying relational database techniques to solid modeling. Comput-Aid Des 1986;18(6).
[2] Hader Th, Hubel Ch, Mitschang B. Information structures and database support for solid modelling. Proceedings of International Workshop on Theory and Practice of Geometric Modelling, Blaubeuren, Germany; 1988.
[3] Kim W. Object-oriented database support for CAD. CAD 1990;22(8).
[4] Sriram RD, Wong A, He L-X. GNOMES: an object-oriented nonmanifold geometric engine. CAD 1995;27(11):853–68.
[5] Karacali O. An introduction to object design technology with computer graphics to support concurrent engineering. Modelling and graphics in science and technology, Berlin: Springer; 1995.
[6] Marir F, Aouad G, Cooper G. OsconCAD: a model-based CAD system integrated with computer applications. Itcon 1998;3.
[7] Ulfsby S, Meen S, Oian J. Tornado: a data-base management system for graphics applications. IEEE Comput Graphics Appl 1982;71–9.
[8] Urban SD, Ayyaswamy K, Ling FU, Shah JJ, Integrated product data environment: data sharing across diverse engineering applications. Int J CIM 1999;12(6):525–40.
[9] Krebs T, Luehrsen H. STEP databases as integration platform for concurrent engineering. Proceedings of Second International Conference on Concurrent Engineering (McLean, Virginia), Johnstown, PA: Concurrent Technologies Co. 1995. p. 131–42.
[10] Loffredo D. Efficient database implementation of EXPRESS information models, RPI, PhD Thesis; 1998.
[11] Sun J, Hardwick M. Building an integrated large scale STEP database for virtual enterprises. Proceedings of ASME DETC99/DFM, Las Vegas, USA; September 1999.
[12] Shin Y, Han S-H. Data enhancement for sharing of ship design models. Comput-Aid Des 1998;30(12):931–41.
[13] GRAD Inc. http://www.gradinc.com/.
[14] Jacobsen K. Development of a ship modeling application using CAS.CADE, an object oriented software development environment. ICCAS (International Conference on Computer Applications in Shipbuilding), Held in Yokohama, Japan; 1997.
[15] ISO TC184/SC4/WG3 N799. ISO/CD 10303-218 Product data representation and exchange—Application protocol—Part 218: Ship structures; September 1999.

[16] ISO TC184/SC4/WG11 N039. The EXPRESS-HLDAI mapping language; 1997.
[17] Morris K. Database management systems in engineering. NIST Technical Report (NISTIR 4987); 1992.
[18] Kim J, Han S. Interface between geometric kernel and database for a ship CAD which has a STEP database as the native storage. Trans Soc CAD/CAM Engng 2002;7(3). in Korean.
[19] Park K-P, Lee K-Y, Cho D-Y. A study on the ship cargo hold structure data model based on STEP. Trans Soc CAD/CAM Engng 1999;4(4). in Korean.
[20] OpenCascade Corp. http://www.OpenCascade.org.
[21] Hwang H-J, Han S, Kim Y-D. Mapping 2D midship drawings into 3D structural models based on STEP AP218. Geometric modeling and processing, Saitama, Japan: Riken; 2002.
[22] Kim C-Y, Kim N, Kim Y, Kang S-H, O'Grady P. Distributed concurrent engineering: Internet-based interactive 3-D dynamic browsing and markup of STEP data. Concurr Engng: Res Appl 1998;6(1):53–70.
[23] Tsuru H, Katayama F, Doi K, Nishiyama H. In: Johansson K, editor. A study of design process innovation with virtual production supported by next generation CAD. Proceedings of ICCAS (International Conference on Computer Applications in Shipbuilding), Held in Malmo, Sweden; 2002. p. 657–70.
[24] EPM Technology. http://www.epmtech.jotne.com.



Junhwan Kim received a BSc in 1995, a MSc in 1998 and a PhD in 2003 from the Department of Mechanical Engineering, KAIST (Korea Advanced Institute of Science and Technology), Korea. His research interests include the CAD database, Internet-based CAD & Collaborative design, STEP, data exchange and history based parametric, Intelligent CAD.

Soonhung Han is an Associate Professor in the Department of Mechanical Engineering at the KAIST (www.kaist.ac.kr). He is leading the Intelligent CAD laboratory (icad.kaist.ac.kr) at the KAIST and the STEP community of Korea (www.kstep.or.kr). He is the editor of the new web-based journal IJCC (www.ijcc.org). His research interests include STEP, geometric modeling kernel, VR application in design, and knowledge-based design system. He has a BS and a MS from the Seoul National University, Korea, and a PhD from the University of Michigan, USA.