Topological Constraints in Ship Design

Lluís Solano[*], Iñigo Gurrea[**] and Pere Brunet[*]

[*]*Computer Graphics Section. Software Dep. Polytechnical University of Catalunya. Barcelona.Spain. (solano@lsi.upc.es, pere@lsi.upc.es)*

[**]*SENER – Ingeniería y Sistemas. Barcelona. Spain. (inigo.gurrea@sener.es)*

Keywords: Topological constraints, ship design, object oriented methods, pattern-based design.

Abstract: In this paper we propose a generic and integrated data structure that is decoupled form the product model database in order to manage and update the geometric elements (parts) related with *topological constraints*. We present a practical application in the case of ship design. In ship design it is necessary to manage and to deal with a huge number of components and relationships. The data structure is based on a directed graph. The software design follows object oriented techniques and pattern-based design. Using class patterns to implement the graph and to manage the topological constraints performs high reusability of the proposed solution.

1. Introduction

During last years, solid modeling techniques and 3D visualization methods have been incorporated in CAD/CAM Systems used in Shipbuilding (Aarnio 2000). This, in conjunction with the dramatic improvement of the performance of computers and graphics devices, has led to the intensive use of 3D design techniques in many shipyards (Alonso 1996).

FORAN (Garcia 1994) is a CAD/CAM system specifically oriented to shipbuilding, used in more than 100 shipyards all over the world. One of the key features of FORAN is to create and to manage a ship product model in a single database containing all ship information in a concurrent multi-user environment.

The product model is a complete representation of the ship as an entity to be designed, fabricated, mounted, operated and maintained. The product model contains 3D geometrical data, information about the structure of the product (eg. the organization of the ship components by systems, blocks and zones), element attributes (materials and other technological properties) as well as all the necessary information, documents and schedules for the ship fabrication and mounting process (Duran 1995). Geometric, engineering and manufacturing data are recorded and maintained by the system, as well as their relationships.

FORAN is now used in many shipyards for the 3D detail design of the hull structural and outfitting elements of ships (figure 1). The incorporation of flexible tools for the management of the relationships and dependencies among the parts of the model, increases the potential to maintain changes in the design along the life cycle of the product. It is useful to include in the product model topological data about the relationships and dependencies among components. This paper addresses the use of object oriented techniques and pattern-based design to manage relationships and dependencies among parts.

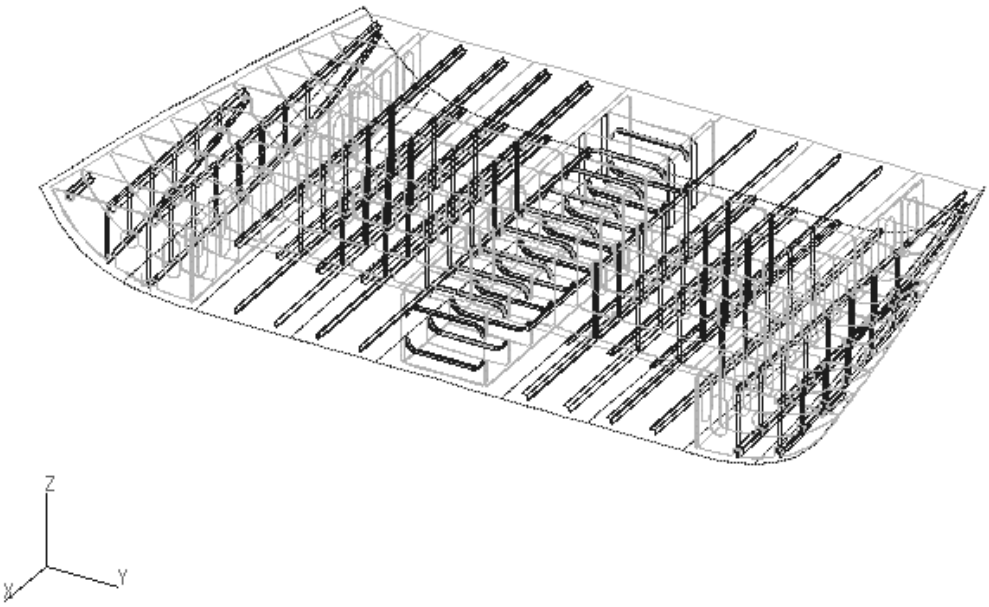

Figure 1: Example of subsection of 3D ship structure.

This paper is structured in the following way. First, the basic shipbuilding process is described and explained from the design cycle. In this process arises the problem of keeping track of the relationship between the hull ship and the structure and components parts of a ship. After that, some definitions are made and the problem is formalized. Finally, a solution is proposed and discussed using object oriented technologies.

2. PROBLEM DESCRIPTION

The starting point of shipbuilding engineering process is the design of the hull surface model. This task takes a considerable amount of time to be completed. The resulting product is an essential information for the following steps. Once a preliminary hull surface model is obtained, the following processes can start. Related with the hull there are all the ship structural elements. From the initial hull surface model starts the design of structural parts as decks, frames, panels and plates, and the components and devices (pipes, pumps, valves, etc.) that are located over the ship structure (figure 2).

Usually, design in shipbuilding environment is performed from a set of ship components that are instantiated with specific parameters. For each part, its location and position is defined together with its relationships with other parts. For instance, a pump is located related to the deck position. In this sense, the design can be understood as an assembly design with a very very large number of components.

In parallel to the design of structural parts based on the hull, the surface model is refined and faired, without preventing other tasks from going on (Rodriguez 2000). Once the surface model is finished, and using an integrated system the work performed on the preliminary model must be automatically updated.

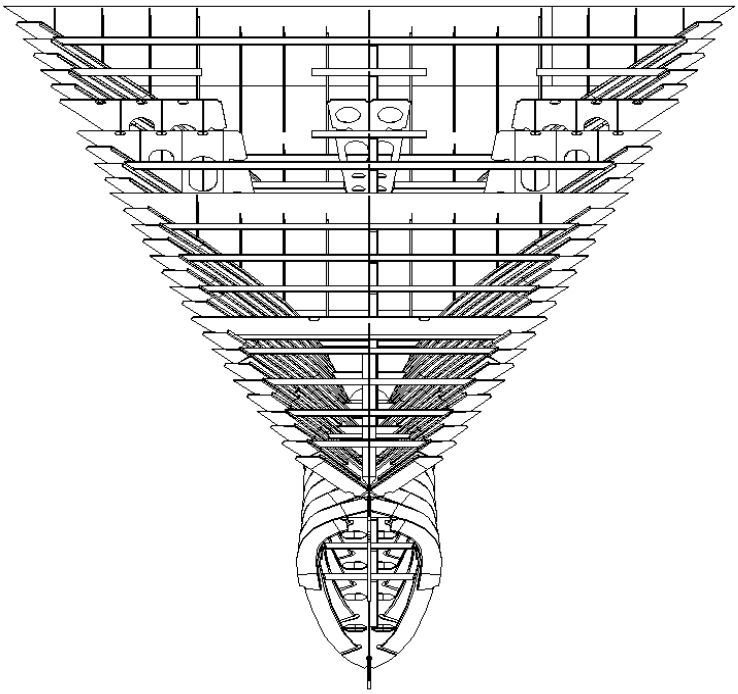

Figure 2: Example of ship structure.

The management of the relationships and dependencies among the parts involved and the automatic update of the geometry are necessary in order to decrease the engineering tasks. The relationships defined among different components allow establishing dependencies in changes. If an element A is modified, the elements depending on A and related with A, must be updated in position, dimension or both. We define these dependency relations among the involved elements as *topological constraints*.

Topological constraints have to be kept in the product model to maintain changes in the design and to update the location, shape and dimensions of the components and parts.

Some of the structural parts are directly related with the hull surface. In fact, there are components such as frames, which they are fit on the hull and follow a curve on the hull surface. This means that any change in the hull surface has to be transmitted to the related components. A modification or refinement in the hull of the ship implies to redesign the dimension and position of the structural related parts as deck surfaces, panels and plates. Using *topological constraints* the parts can be updated in an automatic way.

Modeling parts that are related with *topological constraints*, where the location and dimension of a part depend on the dimensions and locations of others parts is a form of *adaptive modeling*.

More specifically, let us consider the example in Figure 3 that shows a simplified transversal section of half ship.

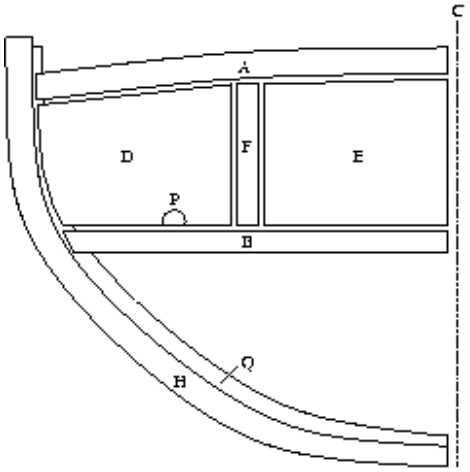

Figure 3: Simplified transversal section of a half ship.

Ship hull modifications must be propagated to the main deck, contiguous panels, lower decks and center plane reference, as they produce changes of their shape or location. The center plane reference must propagate its shape modifications to the different decks and plates. Changes in the shape or location of the lower decks can modify the position of the ship equipment; this forces specific modifications in the associated pipes which end up in changes in the panels that must be crossed by the pipes.

Figure 4 shows the corresponding topological constraints of Figure 3 components.

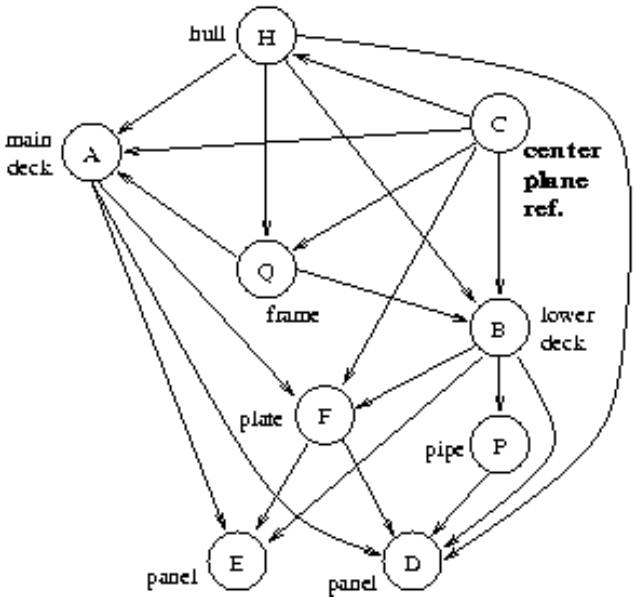

Figure 4: Example of topological constraints among ship components.

The case explained above is a simplified case in order to state the problem. In fact, in real ship design, a huge number of components in each one of ship sections are involved. In average, there can be from fifteen to a hundred sections and from a dozen to several hundreds of structural components as frames, plates, decks, and panels.

3. SOLUTION DESCRIPTION

A ship is designed hierarchically, by composition of simple parts into more complex ones. The natural way of representing the dependencies and relationships is using a directed acyclic graph (Rappoport 1993). From a directed acyclic graph that represents the topological constraints among ship components, it is possible to compute the sequence to perform the automatic update of the components.

We define the *topological constraint graph* as a directed acyclic graph where nodes are components of a design and edges represent the topological constraints.

The update process of the components related with *topological constraints* has to be performed in a specific order. In Figure 2, the evaluation of node P has as a prerequisite the evaluation of nodes Q, B, C, and H. To update the components after a change, it is necessary to process the nodes in such an order that no node is processed before any node which points to it.

Using a *topological sorting* (Aho 1983) it is possible to compute a linear ordering of the graph nodes, such that if there is an edge from node i to node j, then in the linear ordering, node i appears before node j.

Topological sorting is based on a graph depth-first search. The cost of a *topological sorting* is linear (O(n) where n= number of nodes). In the case of Figure 4, the *topological sorting* produces the sequence: C H Q A B F P E D.

A generic and integrated data structure, decoupled from the product model database, is proposed (figure 5). This data structure allows managing and updating the geometric elements (parts) related with *topological constraints*. The data structure is based on a directed implicit graph where the nodes are ship structural components that keep themselves the list of their related components (*topological constraints*). The graph is explicitly traversed when the update process is needed. The ship structure update is performed by a set of class objects that have access to the ship structure components.

The present software development is based in object oriented techniques and pattern-based design (Gamma 1994) in order to have a high level of software reusability and data structure decoupled.

All the parts in the ship structure that are represented as graph nodes can depend on other parts or they may have parts depending on them. To implement this behaviour we use a variation of the Observer pattern. Generically, Observer pattern defines a one-to-many dependency so that when one object changes its state, all its dependent objects are notified and are updated automatically. In the Observer pattern an item, the *Subject*, has a set of items, the *Observers*, that depend on him. When the *Subject* is modified, it notifies to all its *Observers* the modification and they update their state, if it is necessary. In our variation each ship part is both *Observer* and *Subject*.

To unify the interfaces of *Observer* and *Subject*, a Facade pattern is used. Facade provides a unified interface to a set of interfaces in a subsystem. Defining a higher level

interface makes the subsystem easier to use. In this way, the ship part, through the Facade is both an *Observer* and a *Subject*.

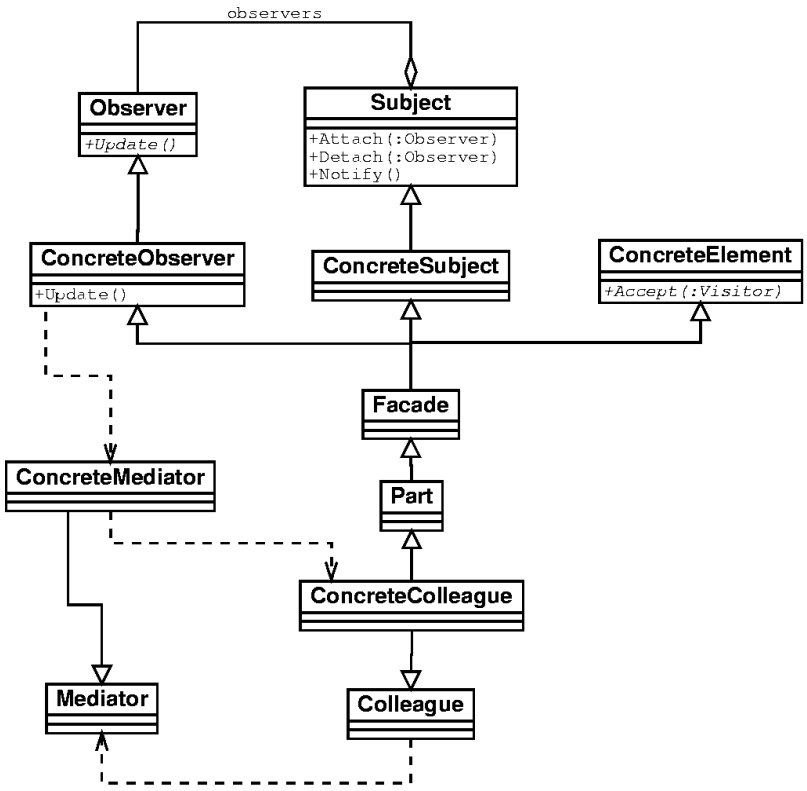

Figure 5: Class diagram defined to deal with topological constraints.

Using the generic Observer pattern, when a part is modified, the related parts are updated in depth-first search through the implicit graph. As we have explained above, due to part update prerequisites, it is necessary to update the parts following the sequence generated from a *topological sorting*. To compute the *topological sorting* and to begin the parts updating, the Subject *notify* method is redefined.

With this approach we have modified the Observer pattern in order to represent an efficient relation and evaluation scheme for problems that can be represented with acyclic graphs and need to be evaluated using *topological sorting*. The *topological sorting* is not always computed. Each time a part is modified the *topological sorting* of the graph is locally evaluated from the modified part. This implies traversals only of the related parts in the implicit structure. The worst case is if there are changes in a part at the top level of *topological constraint* graph.

Modifications of a part are not the only situation in which the parts structure must be traversed. A traversal is also necessary to prevent cyclic relationships when a new *topological constraint* is added or to check the parts status in a concurrent multi-user environment with a unified product model.

In order to have flexible graph traversal strategies the Visitor pattern is used, adding the Visitors *acceptor* to the Facade. A Visitor represents an operation to be performed on the elements of an object structure. Visitor lets define an operation without any changes in

the classes of the elements on which it operates. Using a single common structure, the Visitor pattern allows computing the *topological sorting*, testing the acyclic graph and easily opening a door to define future operations without changing the overall data structure.

The part updating methods are managed using the Mediator pattern. Mediator defines an object that encapsulates how a set of objects interacts. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets them modify their interactions independently. In the Mediator pattern the *Mediator* interacts with the *Colleagues.* The *Mediator* knows the concrete relations between the *Colleagues* and provokes their appropriated updates.

When the *Subject* updates the *Observer*, it activates the *Mediator* that is in charge of the relation managing between the parts. The *Mediator* interacts with the *Colleagues* that are ship parts therefore the Colleagues have access to the necessary methods on the ship parts to update them.

In figure 5, the class diagram used in the data structure described above is shown. The main advantage of this data structure is that it defines a framework to plug *topological constraint* management in a product model with minor changes. The persistent product model information is in the *Part* object. Using inheritance mechanism the topological constraints are added to the product model elements.

4. CONCLUSIONS

In this work a method to deal with *topological constraints* in ship design environment has been presented. *Topological constraints* play an important role to keep dependencies in the product model of a ship where a huge number of parts and information are involved. A directed and acyclic graph structure is used to manage the *topological constraints*. Based on a *topological sorting*, a data structure using object oriented technologies and pattern-based design has been described. The used patterns have been designed in order to achieve the required functionality and behaviour. Pattern-based design performs a high level of reusability. The proposed solution is decoupled from the product model and it is useful to implement *adaptive modeling* schemes. Future work includes the study of parallelization techniques to speed up the update process in the product model.

REFERENCES

Aarnio, M.,(2000). Early 3-D Ship Models Enables New Design Principles and Simulations. In *1st International Euroconference on Computer Applications and Information Technology in the Maritime Industries (COMPIT'2000)*, pp. 5-17.

Aho, J., Hopcroft, J., and Ullman, J. (1983). *Data Structures and Algorithms*, Addison-Wesley.

Alonso, F., Andujar, C., Brunet, P., Garcia, L., Navazo, I., and Vinacua, A. (1996). Virtual Reality Tools In Shipbuilding Design. In *TeamCAD'97*, pp.39-43.

Duran, J., Puzas, M.J., Alonso, F., and Garcia,L. (1995). The use of a 3D product model orientated CAD/CAM System in a small/medium size shipyard. In *CADAP'95*.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Garcia, L., Fernandez, V., and Torroja, J. (1994). The Role of CAD/CAE/CAM in engineering for production. In *ICCAS'94*.

Rappoport, A. (1993). A Schema for Single Instance Representation in Hierarchical Assembly Graphs. In Falcidieno and T.Kunii (eds): *Modeling in Computer Graphics*. Springer Verlag, pp. 213-223.

Rodriguez, A., Vivo, M. and Vinacua, A. (2000). New Tools for Hull Surface Modelling, In *1st International Euroconference on Computer Applications and Information Technology in the Maritime Industries (COMPIT'2000)*, pp. 400-411.