COMPASS SMART PRODUCT MODELING ENVIRONMENT FOR
SHIPBUILDING

Dominique Payannet, Intergraph Corporation, Huntsville, Alabama, U.S.

Introduction

The Problem

The DARPA/MARITECH Commercial Object Model of Products/Processes for an Advanced Shipbuilding System (COMPASS) Program completed research and performed work in developing a Smart Product Modeling (SPM) Environment for shipbuilding. Some of the problems faced in this program included:

- **Scalability** -- Applications need to be same, whether used by one or one hundred designers.

- **Reusable design components** -- From a structural bracket to a complete bulkhead, the user needs the ability to “cut and paste” elements of the design, similar to a Word document.

- **Managing change over the design process** -- Ships can have up to 50 million components that need to be tracked throughout the entire life cycle.

- **Multiple orthogonal views from a single logical object model** -- A component can belong to a system, design, zone, construction unit, and a sub-assembly at the same time. The user must be able to “see” that component in any view, change the view on the fly, and redefine what views the component belongs to as the ship design matures.

- **Open information model** -- Shipyards have multiple systems that must interact with ship design data on a daily basis as the design evolves. It is no longer acceptable to get data “downloads” or translations to business, planning, and production systems. There must be an open, extensible repository of ship information.

- **Concurrent engineering** -- Since shipbuilding is low-volume manufacturing of highly complex products, the industry needs a concurrent, multi-user engineering design and production environment where engineering and manufacturing activities happen simultaneously.

The Solution

The COMPASS SPM Environment provides support to build applications without worrying about the underlying technology. This platform provides a three-tier Windows NT Microsoft Distributed interNet Architecture (DNA)-based architecture. Each task is developed independently from other tasks, which takes advantage of:

- Business rules supported by binary relations

- Relations described in UML and editable by UML visual modeler

- A transaction context allowing short and long transactions

- Concurrency and locking

- A versioning capability

- Independence from the data store

- A set of CAD objects (Active Xs, controls, scriptlets, beans)

- All commands are just COM objects that can be written in any language and plugged in at run time
This paper will review the research and show how the problems were solved in developing this COMPASS platform for the shipbuilding industry.

Current CAD Systems Limitations

Generally, traditional CAD systems are not designed to handle large projects such as those found in the shipbuilding industry. This evolution stems from mechanical-based file systems. Those systems that were applied to shipbuilding projects were hampered by lack of specific shipbuilding functionality, limited in regard to external database structures and management, and lacked the necessary technology that allowed concurrency to be applied to the process. The ones that did meet the functional requirements were monolithic entities that made usage cumbersome.

Data Problems

Mechanical applications are "model oriented." Geometry is the critical data and other data associated with the project (production, maintenance, price, and history) are added when required. The workflow is driven by the modeling capability of the CAD system, while data management is an externalized workflow. Consequently, the data management product serving production and maintenance duplicates some modeling data in order to provide query and versioning capabilities. Traditional CAD files are used as objects referenced by the data management system.

Initially, CAD systems did not provide complex relationships; the data models were static geometry. Model changes invariably meant a remodeling of the complete part. Many mechanical systems eventually added parametric relationships that allowed individual parts to be modified through 'simple parameter edits. Nevertheless, since these systems are file based, the boundaries of the files imposed severe limitations within the relationship tree. Although it is possible to manage relationships within a single file for a specific part or a set of parts, the management of relationships across thousands of files is nothing short of a pipe dream.

Mechanica systems were first designed to create and edit parts. It is only during the past few years that CAD vendors have become interested in assemblies. Progress in this field was again hampered by file boundaries limiting the size of the assemblies. A ship requiring millions of parts is not suited to file-based assembly processes.

Traditional CAD systems are evolutionary. This evolution turned these systems into monolithic applications because of the demand for more specific requirements and "how-to" interjections. In order to provide a complete solution amid an evolutionary system, the reprogramming of existing standard components (printing, text, GUI) had to be considered within each framework. The result of these changes produced complex applications that were difficult to learn, use, and customize.

Through the evolution of geometric data, vendors required their own data format in order to enhance their tools. Too much freedom in the evolutionary process of a CAD system can generate integration problems at the data extraction level when dealing with external processes such as data management and reporting. To alleviate these problems, vendors and customers agreed to formalize initiatives for data standards like STEP1. Although this may resolve the fundamental data exchange problems, the complete schema remains very much hidden in proprietary data formats of individual CAD systems. These systems, having been handicapped over the years by proprietary formats, can only expose public domain schemes through translation. This equates to loss of information and duplication of data.

---

Concurrency Problem

Working in a concurrent environment is restricted with file-based applications, since this allows only long transaction processes into the database. (i.e., only one user working on one file). The ensuing dilemma involves two problems. First, if the CAD system is able to handle assemblies, then only a handful of users will access and manipulate the "higher level assemblies." Second, if the files have "part" granularity, the relationship across parts is very limited and re-computation of other files becomes necessary, thus returning to the old problem. With file-based systems, access control cannot be managed at smaller granularities than a single file, again inducing the problem of relationships across parts and concurrent access.

Evolution

CAD systems have evolved for many years and have been ported to different systems (mainframes, Unix, NT). Backward compatibility was achieved only by providing a proprietary framework that shielded all the compatibility problems. Guarantees for upward compatibility meant that the learning curve for developers, as well as for end users, became prohibitive. The development of applications on these proprietary systems necessitated costly training and high consultancy fees.

Scalability also became an issue. Customers requiring a single workstation for a small specific project would be offered a lower-level CAD system that would be cost effective, but not necessarily technically productive. At the same time, the vendor would offer that customer who required a much larger number of workstations a higher performance system at a higher premium. These disparate systems could possibly work by exchanging data via translation and data duplication. However, the scalability of a system is a key factor in its successful acceptance in the market. More importantly, it addresses directly the concerns that the industry has for many years had to contend with.

COMPASS Architecture

Notwithstanding the problems described above, the DARPA/MARITECH COMPASS research program was launched to develop an SPM environment for shipbuilding. The thrust of this research project was to ultimately develop a set of tools that encompasses past lessons learned, incorporates the latest technologies, and is built on a Windows® framework and tool set. The bottom line is to create major productivity inducements within the shipbuilding engineering design.

The first issue was to remove proprietary frameworks in order to limit the learning curve of application developers. Scalability and component re-use prompted the move towards the standard DNA² from Microsoft.

DNA Architecture

The DNA architecture is based on scalability and binary object re-use. The design of applications using DNA provides seamless scalability from a single workstation to a fully distributed network of computers by making use of all the binary components available on the market. All applications are split into three logical tiers (See Figure 1):

- Server tier that handles the data access.

² Windows DNA “Building Windows Applications for the Internet Age,” Stephen Rauch, Platform Strategy and Architecture, PDC, September '98, Denver, USA.
- Middle tier that handles all the business objects and rules, and ensures data consistency before committing any transaction to the server tier.
- Client tier that contains the Graphics User Interface (GUI) and all the commands accessing the middle tier.

![Three-tier Architecture Diagram]

Figure 1. Three-tier Architecture

Each of these tiers can be distributed over several machines depending on the configuration. It can run on a single machine as well as with X clients driving M middle tier machines, accessing P servers. Microsoft tools provide load-balancing software as well as object caching and resource pooling for the middle tier machines. In this way, the development framework becomes the standard DNA model and the development tool is Microsoft Visual Studio. Within this framework, the COMPASS SPM toolkit becomes just a toolbox containing a set of binary re-usable components (as well as external components) that can be plugged in to build applications that handle large data sets.

Moreover, as the DNA model is Component Object Model (COM3)-based, the components offer another degree of scalability. The interface granularity provides a way to downsize applications. For example, a single application not requiring any access control does not contain components that do not call on the access control interfaces in the middle tier, hence skipping the relevant algorithms. One other major asset of this technology is the variety of client types ('rich' or 'thin') that can be plugged in without impacting any existing application.

Component Model

In order to avoid the 'monolithic' application problems (where all the environments have to be released at the same time by the same provider, deciding the eventual schema evolution), the COMPASS Program selected the more successful component model. With this schema, complex shipbuilding applications are made of numerous environments such as Catalogs, Structural, Outfitting, Planning, Preliminary Design, Drawing, or Analysis, and can be acquired individually and

---

released separately. For the first time, customers can easily create and integrate their own applications into this single environment.

In this manner, a shipbuilding application is made up of independent environments that can be plugged in or removed at any time. Each environment can also be completely independent from the others and have its own GUI, schema, and data store. Each independent environment can reference other environments in a Direct Acyclic Graph (DAG) fashion. (See Figure 2.) Hence the COMPASS platform handles transactions across several data stores.

In order to provide independent design for each environment, there are no imposed constraints. This allows each environment to have: 1) a specific data store that corresponds to its requirements, and 2) its own schema and evolution. This enables users to write their own integrated environments without impacting the existing data model.

Server Tier

As stated earlier, the server tier is responsible for data access. COMPASS adds an insulation layer between the business objects and the persistence schema by traversing the layer. In this way, the data store can be anything as long as there is an available driver for it. The following data stores can be used in the COMPASS platform:

- Object databases: ODI, Versant, Objectivity.
- All OLEDB providers: SQL7.0, Oracle, flat files.

Each data store provides functionality for persistence, query, transaction support, and locking mechanism. Since it is a COM model, some data stores might not support all of this functionality. However, the system will downgrade seamlessly. For example, a data store that does not support query will prevent this functionality from being exposed.

Middle Tier

The middle tier is the heart of the environment as it exposes the object model (logical schema) to all the possible clients. (See Figure 3.) The middle tier's role is to:

- Provide a transacted model for editing data.
- Insure data consistency of business objects particularly relevant with models containing large numbers of business rules.
- Be responsible for keeping object or server connections cache.

The data stores running on the server tier are connected as standard resources for MSDTC (Microsoft Distributed Transaction Coordinator).
They all expose a set of interfaces insulating the data access from the implementation of the data stores. Each business object in the middle tier only accesses this set of interfaces, keeping the business object’s logic completely independent from the data stores.

**Data Consistency**

As the environments use separate data stores, the middle tier has an active short transaction in which each data store part of the data edition is automatically enrolled. This is performed by the standard MSDTC, which implicitly enrolls all the resources (data stores) in the current transaction. When there are multiple independent data stores involved in the application, there is automatically a function of multiple schemas. The "active" schema is the union of all the schemas exposed for a specific operation. Since the environment can only have DAG dependencies, the corresponding schema will also have DAG included.

**Data Models**

There are two types of schemas for applications: 1) the physical model corresponding to the physical implementation for the data store, and 2) the logical model corresponding to the COM model. Middle and client tier objects access the latter since the business rules are only enforced by the business rules stored in the middle tier. Nevertheless, for read-only operations (i.e. reporting) clients might be able to access the data store model directly. The logical model is a public UML model entered with a visual modeler and stored in Microsoft Repository. The business logic refers to the Repository's public cache to create and maintain the business logic.

**Business Rules**

Binary relationships enforce all business rules across objects. These can handle direct dependencies or non-linear constraints. Each relationship can have semantics for every complex operation such as evaluation or copying. Every time a business object is edited and an update is requested, the graph of dependencies described by the relationship is captured and ordered. Then the corresponding semantics (e.g., evaluation semantics) are triggered and modify the dependencies. If all the semantics succeed during the update process, then the current transaction can be committed. Otherwise the current transaction is aborted.

The relationship and its semantics are described in the UML model. Changing a semantic can be done by just editing the schema -- instead of requiring a new release of software, as is common for traditional CAD systems. The mechanism is also impacted by the access control delaying some tasks across access control boundaries.

**Transaction**

The CCMPASS platform supports two types of transactions: short and long. Long transactions are simple to handle since data is insulated from concurrency before editing is executed. Only short transactions make sense for concurrent environments, since the data is locked for very small periods (milliseconds). Short transactions are driven by commands, but their duration is not dependent on the GUI. An end user can take several minutes to perform a complex command while the data might be locked only for a few milliseconds. Combining optimistic locking with the GUI operations with pessimistic locking (during the last data consistency check) provides this capability.

---

No objects remain between transactions; the only possible data left are object caches or connection pools.

**Access Control**

Workflow and security must be controlled on large projects. Each business object is subject to access control rules before finding its way into the middle tier. This is done by a two rule-based system that can be altered at run time. The first rule set assigns the objects according to logical groups; these rules can be as simple as the notion of an ‘active group’ to rules based on any information exposed in the UML schema (i.e. type, interface, attribute value). In the latter case, the rules are re-evaluated before committing the edited data.

The second set of rules grants users, groups, or roles certain operations such as Create, Read, Update, or Delete on certain logical groups. These pre-compiled, optimized rules are run every time a business object is instantiated in the middle tier (creating or loading business objects). The business logic takes into account the access control to delay or prevent the editing of data. This provides full flexibility to organize a dynamic data partition (static in the case of files).

**Query**

Business objects do not have geometric constraints. Geometry is treated as any other attribute for the data management such as provider name, creation date, and price. Query becomes the main selection mechanism to work with data (i.e., "locate" in traditional CAD systems). Query is based on the Active Data Object (ADO) model. It is performed by SQL statements agains the logical schema exposed through the UML model and returns a list of business objects. (See Figure 5.)

**COMPASS Business Objects**

Typically, application writers build their business objects, but the COMPASS toolbox provides several generic objects that can be re-used as is. These include:

- Relations (allow the creation of any business rule).
- References (to external 3rd party objects supporting OLE for Design & Modeling7).
- Symbols (allow sharing of parametric definitions of parts).
- Proxies (allow creation of relationships across data stores).
- Basic 3D geometry (usually aggregated by applications business objects).

**Versioning**

There are a number of reasons why a shipbuilding CAD system must efficiently handle versioning of objects. Two of the most notable include supporting what-if trade-off studies in the early stages of design, and allowing multiple ships to be built from a single class design without having to duplicate and separately maintain copies of the database for each ship.

Traditional CAD systems do not deal with multiple versions of many objects. They depend on an external PDM system to perform this function. This requires copying portions of the product model into working files used by the CAD system, and then copying the data back into the PDM

7 OLE for Design and Modeling Applications, Microsoft, January 24, 1995, Redmond, USA.
database when the work was completed. This approach works for items such as automobiles, with a moderate number of objects, but does not scale well to the ship design problem where the database may house millions of objects and a user will reference tens of thousands of parts in a typical design session.

The COMPASS design has a very clean solution to this problem that takes advantage of the N-tier distributed object architecture. In this approach, the physical model on the disk stores multiple versions of objects, while the logical business object model is kept simple by ignoring versions. A thin layer of software "filters" through the physical versions and delivers the single appropriate version of each object to the in-memory business object tier.

Figure 6 shows how this approach provides each user with immediate access to the shared product model while presenting a logical view of a particular configuration.

The COMPASS design also supports many relationships between objects in the logical and physical models. These relationships may also have multiple versions in the physical model. The software layer that filters the objects also filters the relationships. This ensures that the appropriate version of each object presented to the in-memory business objects and their interrelationships are consistent with the view of the physical model selected by the user.

**Logical Model**

**In-Memory Workspace**

**Business Object**

**Version Selection Filter**

**Physical Model**

<table>
<thead>
<tr>
<th>Version Selection Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Version 3)</td>
</tr>
<tr>
<td>(Version 2)</td>
</tr>
<tr>
<td>(Version 1) Business Object</td>
</tr>
<tr>
<td>(Version 2)</td>
</tr>
<tr>
<td>(Version 3)</td>
</tr>
</tbody>
</table>

**Figure 6. Logical Model is Consistent with the Physical Model**

**Client Tier**

The client tier contains all the GUI and commands specific to applications and environments. It accesses the business objects. For read-only operations, the connections can be accessed through the standard ADO model. Applications can support many types of different clients, from thin clients (that might be Web-centric reporting tools) to rich clients providing real time 3D-model manipulation. Thin clients can be written with market-available tools such as DHTML, controls, ActiveX™, applets, scripts, and through the eventual re-use of the components that form part of the COMPASS toolbox. Although the market provides many standard components (tree controls, GRIDs), there are still few re-usable binary objects for building rich 3D clients (3D views, command control).

The COMPASS toolbox provides the most useful components for a 3D CAD system: view sets, 3D views, locators, highlighters, menus, working sets, transaction manager, command manager, units of measure, session manager, trader, view printers, element lists, select sets, project management, status bar, style manager, and MDI control. From these components, only the working set and the transaction manager access the middle tier.

With the traditional framework, there is a rigid model made of hardwired subsystems like views and documents. Developers override as much as they can of the standard presentation model

8
in order to customize their specific applications. This requires a thorough understanding of the full model as well as all the hard-coded behaviors.

The COMPASS platform provides a unique way to design environments without this drawback. Instead of trying to re-shape a fixed framework, each environment describes the set of components that it requires, including the layout with an XML page. Once the environment is selected, its XML page is parsed, loading into memory the objects described in the page. The layout information is used for GUI initialization. All the loaded components are accessed through a trader object that becomes the root of a flat automation model, that is, the list of components described in the XML page. If the components have interdependencies, they are loosely connected once the framework is fully compiled. (See Figure 7.)

This unique technique allows people to dynamically create environments from any COMPASS component, as well as re-usable components from third-party vendors (ActiveX, controls, applets). (See Figure 8.) The environments load only the necessary components and do not require detailed knowledge of any proprietary framework. Note that each environment can be persisted in sessions files (i.e., saved in the client space) retaining the status of the components when a change in the environment is executed if the session is saved.

In this schema, commands are just standard ActiveX DLLs running against the flat automation model described by the XML page. Commands can be written easily with any language supporting COM objects (applications example delivered with the toolbox shows command examples in VB, J++, and C++). In the same way, all the GUI components, toolbars, ribbon bars, and forms, are just standard controls that can be created with Visual Studio.

Large applications such as shipbuilding can be split into separate environments. The screen is cluttered with full sets of GUI tools, showing only the specific environment commands. There is also full re-use of components since commands (being components themselves) can be shared across environments. When the end user switches between environments, the corresponding XML pages are compared and the common elements are kept in memory, allowing a smooth change over of the dynamic framework. The COMPASS toolbox also contains 2D controls that are able to parametrically edit 2D profiles. In this way, any 3D application allows the editing of 2D shapes (i.e.,

---


---

**Figure 7.** Dynamic Component Framework

**Figure 8.** Re-use of Components
profiles for extrusions, revolutions, drawing generation).

**Conclusion**

Having exposed the problems that traditional CAD systems have in handling complex applications such as shipbuilding, the COMPASS Program demonstrated the following research conclusions:

- Moving from 3D geometry file-based applications to general applications storing data in databases is key to handling large projects efficiently.
- The DNA open architecture is key to controlling and reducing development costs, learning curve, and scalability.
- An open, logical schema is critical for third-party vendors in order to interface their existing applications into this architecture.
- Formalizing the relationships for business rules ensures a full data consistency without burden for the application writers.
- The notion of independent environments simplifies the GUI and facilitates the integration of multiple modules as building blocks.

**References**

7. OLE for Design and Modeling Applications, Microsoft, January 24, 1995, Redmond, USA.

OVERVIEW OF ADVANCED CIM FOR SHIPBUILDING PROJECT

- VISION AND REFERENCE ARCHITECTURE

Ken Ito, Masahiro Sonda, Atsushi Fujita, Satoru Endo,
Ship & Ocean Foundation, Tokyo, Japan

Abstract

The Advanced Computer Integrated Manufacturing (ACIM) for shipbuilding project sponsored by the Nippon Foundation and participated by seven major shipbuilders in Japan, was launched in 1997 with a planned duration of three years. The goal of ACIM project is to enable highly effective, efficient and flexible collaborative engineering for shipbuilding based on knowledge sharing among multiple disciplines by developing practical product/process model and promoting network based software integration environments. To achieve the goal, we have developed a CORBA-based reference architecture that enables integration and management of distributed objects and systems including newly developed applications and existing legacy systems. The aspect of the construction of the future virtual enterprises has also been considered in the development of the reference architecture. In this paper, we provide an overview of the ACIM project and present the ACIM reference architecture.

Introduction

Since 1989, a series of research and development projects on the CIM for shipbuilding has been carried out by Ship and Ocean Foundation together with seven major shipyards in Japan, aiming at drastic increase of productivity in shipbuilding industries. Figure 1 shows the history of CIM projects. The concept of CIM for shipbuilding, which is rather different from mass productive industries such as automobile industries, was established in the first project. The importance of ship product model was recognized and has since been emphasized throughout the CIM related projects. The specification of product model was established in the second project. Object oriented technologies have been widely applied for the system analyses as well as system implementation since the first project. In the GPME project, the preceding project of ACIM, the general product model environment was developed for the efficient development of product model. The results generated from these R&D activities have been transferred to the Japanese major shipyards. All seven shipyards have established their own CIM systems based on the results of the CIM research projects.

The ACIM project was launched in the spring of 1997 with a planned duration of three years. The vision of ACIM project is to enable highly effective, efficient and flexible collaborative engineering environment for shipbuilding based on knowledge sharing among engineers from multiple disciplines. Developing product and process models is one of the major R&D tasks in this project since they are the basis for knowledge sharing. Another important task is to establish a distributed object and system environment based on CORBA (Common Object Request Broker Architecture). Other emerging information technologies such as agent technology is also introduced for better engineering work support and information exchange through network.
Figure 1. History of SOF CIM Project

Goal and Objectives

As stated above, the overall goal of ACIM project is to develop a system framework to support the collaboration among engineers in design and production in shipbuilding.

The specific goals of the project include:
1. To develop a CORBA-based ACIM reference architecture that enables integration and management of distributed objects/models in the network and supports concurrent engineering of shipbuilding
2. To construct a practical product model through the extension of the Frame Library developed in the preceding GPME project
3. To define a process-driven collaborative engineering work paradigm
4. To develop a process model for engineering support that can adapt dynamically changing actual engineering process
5. To develop intelligent agent based collaboration support mechanism
6. To test the system infrastructure using real shipbuilding case scenarios

The deliverables generated by this project, such as product model and applications, are not prototype systems but are practical application systems and will be used in actual ship design and production. "Practical"—i.e., real—is one of the key words in this project. Therefore, the issues related to deployment of the deliverables to the shipyard are also included in the project scope.

Needs for Knowledge Sharing

Shipbuilding is a typical example of concurrent engineering where the information is exchanged intensively between engineers at different stages and continuously updated by multiple disciplinary engineers. The main goal of ACIM project is to develop a system infrastructure to support this
collaboration by using advanced information technologies and to achieve information and knowledge sharing for high efficiency.

To share the knowledge and information about shipbuilding among multiple engineers and furthermore between engineers and computers, one needs both product and process models. Sharing engineering information through a common product model has been proven to be an effective way for engineering team members to identify inconsistencies and generate timely information flows. By accessing a logically centralized product model database, an engineer can retrieve the latest engineering information generated by other engineers and pass his or her task results to others through the database.

While shared product model can be used to facilitate information flows among engineers, it contains only the results generated by engineers. The information of the process through which the results were generated is not part of a product model. Another important aspect of engineering knowledge is about engineering processes. Engineering tasks are not carried out in a random way. Rather, they are well planned and managed based on the planned processes. An engineering process is composed of a set of interrelated activities that collectively realize certain engineering objectives.

Furthermore, there are many applications prepared for the engineering support for design and production. These applications are developed based on shipbuilding specific domain knowledge. It can be said that the specific knowledge of design and production of shipbuilding is expressed by product model, process model and applications that are distributed in the network.

The information technology about distributive objects is emerging nowadays. This technology secures interoperability and adaptability between software systems. Interoperability means how easily a software can be integrated into various other systems and adaptability denotes how one system can be upgraded following the latest technology. This information technology of distributed objects enables us to construct a network-based collaborative environment for ACIM.

**Common Object Request Broker Architecture**

OMG is a software standardization organization that promotes Object-Oriented technologies for integrating heterogeneous and distributed computer systems. OMG's system building concept is described as OMA (Object Management Architecture), the architecture aiming at global network oriented software linking environment. The standard defined by OMG is called CORBA (Common Object Request Broker Architecture). It is natural way to adopt CORBA standard to realize above mentioned ACIM collaboration support environment. The specification of CORBA defines how to make interface which is compliant to OMA's architecture. CORBA enables OMA concept in the following four mechanisms:

- OMG IDL (Interface Definition Language) is used to define software interface grammatically.
- ORB (Object Request Broker) is incorporated to simplify the transaction in heterogeneous and distributed computer system environment.
- OMG IDL written interface facilities called Basic Object Service and Common Facility are defined.
- CORBA Object-Oriented approach enables interoperability and adaptability.

CORBA basic object services are implemented by software vendors as CORBA product. Software components with interoperability and adaptability can be developed by use of CORBA product. CORBA can mask the discrepancy of computer systems (i.e. hardware platforms, software language, data access mechanisms, compiler versions, component module interfaces, network protocol etc.) and there are several wrapping methods to accomplish it.
ACIM Reference Architecture

As described above, CORBA, the distributed object technology, is an important information technology to build the system infrastructure for ACIM since we want to achieve knowledge and information sharing through the integration of applications and data without worrying about hardware or software platforms.

This CORBA-based system environment enables to continuous extension of ACIM through the addition of the new software components and the integration with existing systems. In order to construct the ACIM based on distributed objects environment, it is required to define reference architecture as the system development framework. The architecture is basically designed by referring to the OMA proposed by OMG. The specific requirements that comes from the concept of the advanced CIM for shipbuilding system has been taken into consideration for construction of reference architecture.

1. Product model that represents knowledge and information about ship product and process model that represents knowledge and information about ship engineering process are two core models for knowledge description.

2. The effective services/facilities to support collaboration of engineers, which can be commonly used in ACIM environment, have to be provided.

3. It should be described in the architecture that the ACIM can be enhanced according to the preparation for the domain specific services concerning ship structural design and pipe design.

4. From the practical view point, special consideration must be paid to the integration with legacy systems used in the shipyards.

![ACIM Reference Architecture Diagram](image-url)

**Figure 2.** ACIM Reference Architecture
The ACIM reference architecture is shown in Figure 2. In this architecture, eight (8) interface categories are defined according to the functionality of software components which provide necessary facilities and services to support concurrent engineering of engineers. ORB plays an important role to associate these software components included in these categories.

In the following, we describe the functionality of each category and implementation of these categories.

**CORBA Object Services : COS**

This category is corresponds to the Basic Object Services, one of architectural components of OMA described in previous section. These services are the basic services to construct distributed object system. The specifications on functionality of each service are defined by OMG and software vendors implements its functionality based on these specifications. There are several commercial CORBA products available nowadays. The specification of object services has been already defined by OMG and published. At the moment, there are sixteen (16) services defined as specification, such as “Naming service” for linking object's name and real object, “Event service” for asynchronous communication. In the ACIM project, we adopted “Orbix” from IONA Technology as the CORBA product.

**GPME Facilities : GF**

GPME facilities provides the access facilities to the product model which is implemented in the previous GPME project and extended to the practical level in the ACIM project. The versatile access to the product model is available through open natured IDL interfaces. The instantiation and retrieval of instances from product model are performed through this services. The important point is that we have adopted the late binding method for implementation, so all the class methods which have been already implemented in Frame Library and also new methods which may be added in future are available as a CORBA objects in ACIM environment through this access facilities. The late binding enables to specify dynamically the functions or method when application is executed. This means it is not necessary to define the IDL interfaces and implement it one by one. Even in the case that extension or modification of certain method is occurred, only the recompilation of source program has to be done.

**Workflow Management Facilities : WfMF**

As was described above, we regard the process model as one of core model to express knowledge about shipbuilding process and to realize collaborative engineering environment. In ACIM project, we propose a process-driven work paradigm for collaboration support. The paradigm will be briefly described in the succeeding section and the details are presented in another paper submitted to ICCAS.

This category provides the necessary functions to realize process-driven work paradigm. Two important servers have been identified as system components of work support system. The components are Process Server and Enterprise Resource Server. The functions of Process Servers include capturing process models and managing process execution based on the process models. For process execution management, Process Server generates activation signals, control the real process by reasoning based on the given rules. On the other hand, Enterprise Resource Server centralizes resource information logically. To manage engineering processes, resource information is very important. It has also process templates and journal information about the results of engineer’s operation or communication. The services corresponding these functions are provided by WfMF facilities.
Domain Function Facilities: DFF

DFF is the category which includes shipbuilding specific functions necessary to develop applications. The DFF facilities provide the services based on the shipbuilding ontology. The applications are developed mainly on the basis of these DFF facilities. The functions are dependent on the kinds and functions of applications to be developed. Theses DFF facilities are sometimes used for several applications, and furthermore one DFF function may be used by other DFF facilities as the primitive services. The DFF facilities provide functions for application development by combination of GF services such as assessor and methods which GF provides.

As mentioned before, the services provided by GF are implemented by late binding method. This means all the methods provided by product model are available as the CORBA facilities. The methods included in the GF category contains rather higher level functions such as calculating the weld length and weight calculation about intermediate assembly products. Due to the late binding method, these functions are provided by the GF facilities. The examples of DFF are shown in the another paper submitted to ICCAS.

Applications: APF

In this reference architecture, applications are defined as the new applications used in shipbuilding design and production which has been developed on the basis of the ACIM reference architecture and adapted to the CORBA-based ACIM open environment, i.e. distributed object environment. In ACIM environment, applications are developed efficiently combining the services and facilities provided by software components included in other categories of ACIM reference architecture. In this project, two (2) applications for production planning are developed in a practical level for the purpose of the verification of ACIM product model.

Legacy Applications: LA

This category contains all the applications, such as CAD systems and other domain specific applications to support design and production work, currently used in the shipyard. These applications do not have the adaptability to the CORBA environment. So it is impossible to communicate with other applications through CORBA. However, from the practical view point, it is very important to exchange information between these legacy applications and software object developed under the ACIM open environment. So these applications have to be adapted to the ACIM environment. CORBA enables to make these legacy applications adapt to the ACIM open environment. The software added to the legacy applications in order to make it adaptable to CORBA environment is called wrapper. Through the wrapper, legacy applications can communicate with new applications or COTS products adaptable to CORBA environment. On the contrary, it is also possible to make the functional components included in the legacy applications as a software component that can be used by other applications. COREA provided the great advantage that makes the legacy application adaptable to distributed object environment. This means that the a part of legacy application can survive as a software component for a long time.

ACIM Common Facilities: ACF

Software components in this category provides the general services commonly used in other software components. These components are independent from the domain specific ontology, so these are widely used from many applications. Commercial software components not compliant with CORBA environment can be adapted to the ACIM environment by adding a wrapper and added as the new service. In this way, the new software components can be added easily to ACIM services.

On the other hand, the native services are developed and prepared as the ACF services in this
project. These are agent facilities and product model management facilities. As these facilities are independent from shipbuilding domain ontology, services are categorized as ACF.

Example of services provided are schedule management, agent supported functions that may be used in the process-driven ACIM collaborative environment, product model management facilities that is indispensable for operational phase in a shipyard. Furthermore, ACIM desktop, work environment for ACIM users, is developed in this project. Services necessary to develop ACIM desktop are provided as ACF.

COTS (Commercial-Off-The-Shelf) Products

This category includes the COTS products which provide CORBA compliant services. The concept of integration of distributed systems based on CORBA is widely accepted. Software vendors such as PDM(Product Data Management) and ERP(Enterprise Resource Management) are now moving to adapt their software products to CORBA. Furthermore, the consortium of PDM vendors are trying to define the standardized IDL interfaces aiming at the information exchange between different PDM products. Also it is expected that the software vendors which are specialized in development of software component will emerge in near future according to the progress of the distributed object technology progress. By using best-breed COTS and/or combining effectively commercial software components, the application development environment will be much improved.

On the other hand, in the world of Microsoft Windows, distributed object linking mechanism DCOM(Distributed Common Object Model) has been formalized. Many software component products are available nowadays. For example, Microsoft Transaction Server can be easily customized by selecting and combining software components implemented as the DCOM objects. As will be explained later, it has already been confirmed through the prototype test that the information exchange between CORBA and DCOM can be realized with ease by use of CORBA/DCOM bridge provided by CORBA commercial products. This means that the libraries compliant for DCOM can be used from CORBA world. Therefore it is expected that increase of commercial software components compliant with DCOM enable efficient application development even in CORBA environment.

As was described above, the Advanced CIM system which enables highly efficient collaborative engineering can be realized on the basis of CORBA-based ACIM reference architecture. The flexible exchange of information and knowledge through network can be realized by CORBA. Services physically distributed in the network can be used regardless of the hardware/software platform. The information exchange between applications become possible over the enterprise.

Verification of CORBA Based System Through Prototyping

In order to clarify the technical issues prior to the actual development we have carried out the verification of CORBA-based system by prototyping. The scenario of prototype system is as follows;

- Firstly, a test application access to the product model to get all parts from specified assembly block
- Secondly, the calculation of weight and welding length of specified assembly block is carried out by using services included
- Display the results to a screen in graphical way by Java applet

The prototype scenario is very simple one, and it includes services of several categories such as CORBA Object Services(COS), Domain Functional Facilities(DFF), GPME Facilities(GF) and
Applications (APP), and it is enough to verify the possibility about the CORBA based systems. Figure 3 shows the prototype test environment.

We have confirmed through this prototype system that there is no specific technical issue for developing the CORBA-based system. The performance overhead is not so big in case that the granularity of service is not so small. We have also carried out another prototype test concerning about CORBA/DCOM linkage. The test scenario is to get information from product model on the EWS and to transfer it to the spreadsheet on the PC. The functions provided by commercial CORBA products as a CORBA/DCOM bridge was used in the test. We have confirmed that the data exchange between CORBA object and DCOM object can be realized without any problem. This means that the services implemented as DCOM object can be utilized in ACIM environment.

![Diagram of Prototype Test Environment](image)

**Figure 3.** Prototype Test Environment

### Development of Practical Product Model for Shipbuilding

Now, we will explain another important issue in the ACIM project. In the ACIM project, it is recognized that product model which expresses knowledge and information concerning ship as a product plays key role for the knowledge sharing. Thus, the product model should have enough ability to express information and knowledge on actual ships and should be practical. One of the important R&D tasks in the project is to establish the practical product model for shipbuilding, which express ontology of shipbuilding. When developing the workable product model, maximum usage is made of GPME (General Product Model environment) which has been developed in the preceding CIM project. In the GPME project, CFL (Common Frame Library), the information model to express common ontology for assembly industries has been extended to EFL/S (Extended Frame Library for Shipbuilding) which includes the ontology for shipbuilding. However, it was a relatively small extension only for verifying the usefulness of GPME as a product model development software tools. In the ACIM project, extension of EFL/S is being carried out to much more practical level. The product model is realized by combined use of CFL and EFL/S and becomes the base for the engineer's collaboration. EFL/S is now being extended in two directions as shown in Figure 4. One is to enhance
the product model so as to be rich in the modeling flexibility; this lead to the easy design alteration and simulation. Another is to extend the expression capability so as to express most of the hull structures including details and almost all outfitting of actual ships. The details of the practical product model construction is referred to paper submitted by other authors.

![Diagram of EFL/S with annotations](image)

**Figure 4.** Extension of EFL/S in Two Directions

**Development of Process Model To support Collaborative Engineering**

While knowledge about engineering products captured by product models is important for engineering support and needs to be shared among distributed engineers and systems, another important aspect of engineering knowledge is about engineering processes. In ACIM project, the process model that represents the knowledge and information about engineering process is one of the core models for collaboration through knowledge sharing. Engineering tasks are not carried out in a random way. Rather, they are well planned and managed based on the planned processes. An engineering process is composed of a set of interrelated activities that collectively realize certain engineering objectives. A process model is a representation of an engineering process in a form which supports automated manipulation or enactment by a process management system. Process model includes the control information that defines how engineering activities should be carried out and how they relate to each other.

In ACIM project, we propose a framework based on a process-driven and agent-supported approach to collaborative engineering support called Active Process. The process model developed in Active Process has a capability to describe 'how engineering tasks should be done' in order of precedence. It is powerful but still not sufficient because engineering task procedures will be often changed according to situation. To meet this problem, agent-supported approach is introduced to Active Process. Each participant in the system has its own agent who is interested in negotiation with others to acquire information.

Figure 5 illustrates how process information represented as process models can be used to support engineering work and collaboration. The details of the proposed framework is presented in another paper submitted by other authors.
Verification of Product Model and Collaborative Engineering Environment

Verification of Enhanced and Extended Product Model

As was stated in the previous section, product model (mainly the EFL/S) is now being extended in two directions, i.e., one for enhancement on the functions such as modeling flexibility and the other for the extension of expression capability. We take different verification approach for these two extensions.

As for the verification of enhancement of functions such as modeling flexibility, we take an approach to verify it through the development of applications and their execution for the enhanced product model. The application is being developed by using the class methods provided by enhanced product model. The application must be a practical one, not a prototype, in order to evaluate functionality of product model more correctly. Two applications are now being developed for this purpose. One is an application to support initial production design so-called block breakdown (ACIM IPD : Initial Production Design), while another is an application to support process planning (ACIM CAPP : Computer Aided Process Planning). These applications are developed on the basis of the ACIM reference architecture. The services provided by GF, DDF and COS of ACIM reference architecture are used to develop these applications. Details of the applications are described in the paper submitted by other authors.

As for the verification of the extension of model expression capability, we take an approach to verify it through the exchange of model information for actual ships generated by CAD system being used in the shipyard. As the model data of shipbuilding CAD system now in practical use can express most of the hull structures including details and almost all outfitting of actual ships. Therefore, if these information can be transferred to the ACIM product model and successfully stored, it is confirmed that the extended product model has enough coverage to express actual ship structures and outfitting items.
In order to exchange data between product model and CAD system in shipyards, we are now developing the dual-direction data transmission application. This application is also developed on the basis of the ACIM reference architecture. The data exchange test will be carried out using this software. As the software is so developed as to be able to exchange in dual direction, the results obtained from ACIM IPD and CAP? described above is transferred to the existing CAD system in shipyards and continuously used in the practical design. This means that this data exchange software can be used in an actual ship design process.

Verification of Collaborative Engineering Environment

The goal of ACIM project is to establish the system infrastructure using advancing information technologies which enable efficient and flexible collaborative engineering environment for shipbuilding on the basis of the knowledge sharing. We plan to verify it based on a set of test scenarios. In the test scenarios, all the deliverables obtained from the project such as practical product model, applications of ACIM IPD and CAPP and process model are taken into consideration. In the project, we are planning to propose the ACIM desktop which is newly developed user’s work environment with enhanced graphical interface. The process-driven and agent-supported mechanism for collaborative engineering support mentioned in the previously section will work on this ACIM desktop. The verification of collaborative engineering environment will be carried out through this desktop according to the test scenario.

Conclusions

In this paper we presented an overview of our on-going ACIM project, focusing on the ACIM reference architecture. The vision of ACIM project is to enable highly effective, efficient and flexible collaborative engineering environment for shipbuilding based on knowledge sharing between multiple disciplines by developing practical product/process model and promoting network based software integration environments.

In order to achieve our goals, we have developed an OMG CORBA-based reference architecture that enables integration and management of the distributed objects, models, and systems including newly developed applications and existing legacy systems currently used in shipyards.

Our proposed ACIM reference architecture represents a framework of the advanced CIM systems. This architecture is open to the outer world and adaptable for the emerging information technologies. The product model and process model are defined as the core components of knowledge sharing ACIM environment. The practical product model is developed in this project. This product model is used in the shipyard after this project is completed.

We propose a framework based on a process-driven and agent-supported approach to collaborative engineering support called Active Process.

The verification of the enhanced product model and collaborative environment is carried out through the newly developed applications.

The progress of information technology is being accelerated nowadays. The ACIM reference architecture based on the distributed object oriented paradigm is very flexible enough to incorporated the new technologies. We believe that the ACIM reference architecture provides a platform for continuous improvement of CIM for shipbuilding.
Acknowledgements

This project has been carried out under the sponsorship of the Nippon Foundation, and the authors would like to express their cordial thanks to them. They would also like to express their gratitude for the guidance and encouragement received from Prof. Takeo Koyama, University of Tokyo and Prof. Yan Jir, University of Southern California.

References

1. Koga, K., CIM System and Product Model for Shipbuilding, ICCAS 94
2. Nagase, Y. et al., GPME Modeling Methodology and Applications, ICCAS 97
3. Kawazoe, H. et al., GPME: An Information Modeling and System Integration Environment for Shipbuilding, ICCAS 97
HIGH LEVEL LANGUAGE FOR SHIP DESIGN

F. Alonso. Sener. Ingeniería y Sistemas. S.A., Tres Cantos, Madrid, Spain
J. Subias. Sener. Ingeniería y Sistemas. S.A., Tres Cantos, Madrid, Spain

INTRODUCTION

The CAD/CAM Systems currently used in shipbuilding are powerful tools, essential nowadays to perform in an efficient and precise way all the design and manufacturing processes of a ship.

The evolution of these Systems over the last 25 years led from an initial situation, covering in a limited way the design and manufacturing processes (basically naval architecture and N.C cutting), up to the present situation, covering most of ship design and production tasks.

There has been also a big change, from the initial applications, basically in batch mode or with limited interaction (mainly alphanumeric), to the present graphic interactive applications, working with 3D models and with a high degree of interaction between the user and the application.

As a result, the applications have become very complex, requiring a high degree of customisation and flexibility in their operation.

All this has generated new user requirements to achieve a greater efficiency and flexibility in the use of the CAD/CAM Systems.

These new user requirements can be summarised as the need for a macro language allowing the users to perform repetitive tasks, to define their own standard constructive solutions, and in general to create "intelligent" programs able to access the functions and commands of the System.

As an answer to this important requirement of the users of its FORAN CAD/CAM System, SENER decided to develop the new FORAN Macro Programming Language (FMPL) as a general tool to be added to all the modules of the System.

The new FORAN Macro Programming Language has been developed using the Tcl/Tk scripting language, extended with the capability of executing commands and other utilities of the System.

OBJECTIVES OF THE PAPER

This paper intends to describe the main characteristics of the new FORAN Macro Programming Language, making special emphasis on the characteristics relevant to a user of ship design applications.

With this objective in view, the paper has been structured in the following parts:

- A short introduction to Tcl/Tk, the basic language used for the development of the FMPL.
- A description of the objectives, main characteristics and structure of the FMPL.
- A presentation of some practical cases showing the application of the FMPL to different areas of ship design.
- An analysis of the future developments of the Tcl/Tk language and the FMPL.
- A summary of the results obtained up to this moment, with an estimation of the savings and advantages that the users can expect from the normal use of this Macro Programming Language in their daily design tasks.
A SHORT INTRODUCTION TO Tcl/Tk

Tcl is one of the most widely known scripting languages freely available through the network. A scripting language, as opposed to a system programming language (such as C or C++), is designed to integrate, connect or "glue" basic components, commands or applications, [JO'98]. Tk is an extension to Tcl, a toolkit to build Graphical User Interfaces.

Tcl/Tk is an interpreted language with many features available in standard procedural languages (variable assignment, procedure calls, flow control structures, etc.). The basic data types are strings, lists and arrays. To speed up the process, a compiler is also available with the latest Tcl/Tk version.

Among the many advantages that Tcl/Tk offers, we can note the following:

- It is easy to learn.
- It allows rapid development of applications.
- It is available in many platforms (Windows/NT and UNIX, including LINUX).
- It strongly facilitates the embedding in the language of new functions or commands developed in other languages like C or C++.
- It is free, as well as many of its extensions.

All these advantages, as well as the availability of tools to build Graphic User Interfaces (the toolkit Tk), induced SENER to select Tcl/Tk among other scripting languages (Perl, Visual Basic or Python) as the basic language to develop the FORAN Macro Programming Language.

On the other hand, although Tcl/Tk provides many useful tools and capacities, the power and flexibility of a Macro Programming Language built on top of a scripting language like Tcl will mainly depend on the components, functions or commands of the basic application (in this case FORAN) made available to the users through the scripting language.

THE FORAN MACRO PROGRAMMING LANGUAGE

The FORAN Macro Programming Language is a new macro language build on top of the Tcl/Tk scripting language, to extend and complement the interactive command capabilities of FORAN.

The Macro Language uses all the power of Tcl/Tk extended with the possibility of executing FORAN commands and other utility functions.

The Macro Programming Language has been developed to fulfil different objectives:

- To allow the users to repeat sequences of commands. to facilitate the execution of repetitive tasks during the design activities.
- To generate geometry macros for the 3D definition of geometric models of equipment units, and outfitting and accommodation components.
- To develop more powerful macros accessing all the capabilities of the FORAN command language and a set of specific FORAN functions for each design discipline.

To achieve the first objective, normal users of the System do not need to know anything about the Macro Programming Language or about programming in general. More advanced users trying to reach the two last objectives need some more knowledge of Tcl/Tk and of the FMPL.

A schema showing the integration of FMPL inside the FORAN modules can be seen in Figure 1. Macros can be executed in the same way as other interactive commands of FORAN, and also can
be added to the standard menus of FORAN as commands or icons.

**Macro recorder utility to generate sequences of commands**

Using this utility, users can generate macros containing sequences of commands. The utility opens a console to add interactivity and control to the automatically generated macro.

The recorder console permits the user to decide which actions should be interactive when reprocessing the macro. The console has also functions for displaying user messages, adding pauses, storing the current viewing attributes or storing the scene.

For more advanced users, this utility can be used for the creation of skeletons of new macros to be completed or modified later on using the text editor.

**Generation of geometry macros**

A geometry macro is a user defined program which performs a 3D object layout based on the rules described in the macro.

Geometry macros can be used to define 3D representations of pieces of equipment, piping and ducting components, accommodation elements and outfitting structures (supports, hangers, ladders, etc) making it easy to adopt different shapes and dimensions.

The parameters of the macros can be standard values stored in the database or can be given by the user or directly measured from the ship 3D model.

**Macro programming for advanced users**

A complete use of the capabilities of the FMPL allows advanced users to combine all the power of Tcl/Tk with all the existing FORAN capabilities.

By extending the script interpreter with new FORAN functions (libraries), the macro language will not only be able to process FORAN commands but also to call library functions that will allow to implement new capabilities not covered by the existing commands.

To facilitate user interaction, specific functions for text prompting and graphical input have been added to the Macro Language.

**APPLICATION FIELDS**

The Macro Programming Language has not been conceived as a macro language for a specific ship design area, but as an open and complete language for all the design activities covered by the FORAN CAD/CAM System.

As a consequence, the language incorporates general functions that can be used in all disciplines and specific functions for each design discipline.

To illustrate the capabilities of the Macro Programming Language, we have selected some examples of macros in the different areas covered by the System. These macros have been created using a first prototype of the Macro Programming Language.

Two of the examples (Hull form import and Definition of standard loading conditions) use application functions developed in C, called from the Tcl scripts. The other macros call directly different FORAN interactive commands from the Tcl script.

The prototype will evolve in the near future, incorporating more functions and commands that will
allow adding more “intelligence” or “expertise” to the macros.

Definition of the working environment

As we have already indicated one of the simplest uses of the Macro Programming Language can be the repetition of sequences of commands (with or without interaction).

One example of this use can be the definition of the working environment when a user starts a new working session.

To establish this environment, the user needs to enter a set of interactive commands to read the whole information of the working area (including hull structure, equipment, ducts, pipes, foundations and cable ways), to change the visual parameters and to define the default technological attributes.

All these commands and other common initial tasks can be replaced by a simple macro performing all these tasks (See the result in Figure 2).

As was mentioned before, an important advantage is that the macros or scripts with sequences of commands can be automatically generated during the interactive sessions of the modules of the System.

Hull Form Import

This example shows the results of a macro to import a complete hull form, defined by means of DXF lines, into the System.

In this case the macro is a Tcl script using the following basic functions:

- Opening a file
- Reading a set of DXF lines
- Creation of a boundary curve
- Creation of a knuckle
- Creation of a grid of waterlines
- Creation of a grid of frames
- Closing a file

The macro can be used to import any hull form defined by means of any set of DXF lines. The results of the use of this macro to import the hull lines of a catamaran can be seen in Figure 3.

Naval Architecture

To create the macro shown in this example the following basic functions have been used:

- Definition of a new loading condition.
- Obtaining compartments with some specific contents.
- Definition of a load in a compartment.
- Storing a loading condition.

Combining these basic functions, a macro for the creation of standard loading conditions can be easily created. The macro and some loading conditions created with the macro are shown in Figure 4. This example shows how easy the preparation of a macro can be. The complexity of the macro (and its power) can be increased according to user requirements.
Hull Structure

This case consists of a macro for the definition of all the shell plates of a central zone of a ship. The macro makes use of the following individual commands of the FORAN Shell and Deck Plate Development module:

- Butt definition.
- Topological point definition.
- Seam definition.
- Panel definition.
- Plate definition.
- Some measurement commands, like distances in development.

The input parameters for the macro are the following:

- Limits of the zone (they can be given interactively).
- Keel width.
- Plate width.
- Plate length.
- Height of seam above double bottom deck.

The macro also establishes some default parameters required for the plate generation, like the material quality and thickness of the plate (they also could be given interactively) and takes some "decisions" depending on the geometry of the shell.

The result of the application of the macro to a bulkcarrier of 182 metres of length can be seen in Figure 5.

The macro allows the creation of 11 butts, 15 seams, 30 panels and 150 plates, and the corresponding symmetrical elements, with a single command, producing a considerable saving of time. This macro can be considered of medium/high complexity, requiring not only a good knowledge of Tcl/Tk but also of the FORAN command language.

The result of the application of the same macro to a different ship (in this case a chemical tanker of 133 metres of length) can be seen in Figure 6.

Piping

To show the possibilities of the Macro Language in piping, a macro for the automatic generation of different layout alternatives, following the main ship directions between two points, has been created.

The macro makes use of some standard available piping layout commands, including clash detection between the new pipe and the other elements. The macro takes into account some constructive constraints (e.g. bending constraints). The macro also inserts flanges at the ends of the pipe. The macro builds consecutive layouts in a sequential way and the user may select interactively the most adequate layout.

Starting with the working environment shown in Figure 1, a first piping layout alternative built by the macro can be shown in Figure 7. This alternative presents a clash that is also visualised in the same figure. Another alternative piping layout built by the macro and without any clash is shown in Figure 8.
Piping supports

Piping supports can be generated using geometry macros with standardized values of the parameters that can be hard coded in the same macro.

The piping support generated by one of these macros is shown in Figure 9, including the parameters of the macro.

Macros can also be used to position the piping supports in piping lines, as can be seen in Figure 10. The nominal diameter will be taken directly from the piping line.

Accommodation

Accommodation is another example of the use of macros. The result of a very simple macro for the generation of sets of accommodation components is shown in Figure 11. The only parameter in this case is the number of chairs.

FUTURE EVOLUTION OF THE FORAN MACRO PROGRAMMING LANGUAGE

The future of the FMPL will depend, firstly, on the evolution of the scripting language on top of which it has been built.

The future of Tcl/Tk

Tcl/Tk is becoming one of the most popular scripting languages available through the web. According to the latest estimations there are about 50,000 new users each month.

One key factor in this dramatic increase of Tcl users has been the constitution of a new company by the Tcl creator, J. Ousterhout. The aims of this new company are to develop the free Tcl core and to give commercial support to Tcl/Tk, creating new development tools, like debuggers, compilers and source code checkers, and promoting the development of new Tcl extensions [Tcl'99].

Among these extensions, because of their interest for future developments of the FMPL, we can point out the following:

- [incr Tcl], [incr Tk] and [incr Widgets], which are object oriented programming extensions.
- OraTel, to provide access to ORACLE databases.
- Tcl Browser Plugin, to run Tcl/Tk scripts inside web Browser pages.
- TclDP, to provide distributed programming features, including socket access.

New capabilities of the FORAN Macro Programming Language

The FMPL has to follow a continuous process of development, incorporating new functions as a result of the requirements imposed by the users of the System.

In addition to this, two important developments are foreseen for the near future:

- The extension of the language with a set of functions to generate tailor made reports, including text, tabular information and graphics. Functions to extract information from the product model,
including SQL calls will be available.

A new Programming Environment to create new modules that will use all the functionality of the FMPL.

CONCLUSIONS

As has been shown in the practical cases presented in this paper, the use of a Macro Programming Language integrated in a shipbuilding CAD/CAM System can produce significant savings in many ship design tasks, not only for detail design, but also for ship conceptual and basic design.

The main advantages offered by a Macro Programming Language of this type built on top of Tcl/Tk can be summarised in two points:

♦ Simplicity of use.
♦ Easy access to the commands and functions of a powerful ship design and production system like FORAN.

However, the savings and advantages that can be expected from this new FORAN Macro Programming Language will be determined to a large extent by the degree of compliance of the following goals:

- High adaptability of the Macro Programming Language to the real requirements of the ship design users
- Availability and accessibility from the Macro Programming Language to most of the basic functions and commands of the CAD/CAM System

To achieve these goals, a joint project between ASTILLEROS ESPAÑOLES and SENER has been established. The aims of this project are to identify the design tasks performed with the CAD/CAM System that can be improved by means of the introduction of macros and to specify the corresponding additional functions to be included in the language. The project started the fourth quarter of 1998 and it is foreseen to finish at the end of 1999.

REFERENCES


Figure 1 - FORAN Macro Programming Language Schema

Figure 2 - Working environment defined with a single macro
set ballast 1
set fuel 3
set diesel 4
set luboil 5
set water 6
set load 10

# Assign filling percentage (pfill)
proc fillspaces (content pfill) {
set 1spaces
set laux
set laux [ReadListSpaces -en Sccontent]
set Lspaces [Sort Labels]
for [set i 0] {Si < [length $Lspaces]} [incr i] {
    set Idsp [string range [index $Lspaces Si 0 3]
    # Set filling percentage
    LoadInSP -name $Idsp -pf $pfill
}
}

# Main program Definition of Loading Conditions
DclLoadInd -name BLDE -desc "Ballast departure"
# Selection of spaces with content
fillspaces Ballast 100
fillspaces Sfuel 100
fillspaces Sdiesel 100
fillspaces Swater 100

# Transfer the information
StLoadInSP
# Store the loading condition
StLoadInd
DclLoadInd -name BLAR -desc "Ballast arrival"
fillspaces Ballast 100
fillspaces Sfuel 100
fillspaces Sdiesel 100
fillspaces Swater 100
fillspaces Sload 10

# Macro for definition of standard loading conditions

Fig. 3. DXF/3D imported Hull Form

Fig. 4. Macro for definition of standard loading conditions
Figure 5. Panel and plate definition. Bulkcarrier

Figure 6. Panel and plate definition. Chemical carrier
Fig. 7 - First piping layout alternative. Detail of interference

Figure 8 - A different piping layout alternative
Figure 9 - Piping support defined with a macro

Figure 10 - Positioning of supports
Figure 11 - Definition of accommodation elements by means of macros