

CAD-Centric, Multi-Source Data and Information Management

Nick Danese, Nick Danese Applied Research, Antibes, France, ndar@ndar.com

The ability to share data during ship design, production and management is very limited, with undesirable consequences. Data that could be shared with benefit can be loosely categorized as CAD and non-CAD, as a function of source, nature, but also practical use and possible exploitation parameters during the various stages of the ship's life span. Existing off-the-shelf tools and technology can be used today to share, manage and exploit such data. Examples of the above, including software recently developed for the purpose will be illustrated, and a practical example presented.

1. Introduction

The effective use of orally transmitted data has been the foundation of mankind's evolution, *Danese (2010)*, but for practical and effective use in modern industry data needs to be re-usable exactly. At least three corollaries follow:

- data must be stored persistently.
- data must be retrievable.

Therefore

- data must be stored electronically.

Therefore, the creation of an electronic version of data is the first step. Electronic storage being a rather non-standard environment by definition, data must be formatted intelligibly, and formatting must be documented. While being obvious, this is the case far less often than desirable. The vast number of electronic data organizer tools developed in recent times is a loud symptom of the importance of data management, and the absence of one, or even a few, standard-setting product(s), an even more evident indication data-sharing proficiency has not been achieved.

The reason for data not being shared effectively is not fundamentally technical, plenty of tools exist today to help find, retrieve and process information. On the other hand, assuming that data has been made available in the first place, which is more and more the case, how it is made available will determine whether it can be found or not, and then exploited, or not.

With regards to the CAD environment, and more specifically the ship related industry, a majority of data does exist in electronic form, is readily available in industry standard file formats, but is not exploitable for a number of reasons, two immediately identifiable culprits being proprietary data structure, and absence of contextualization.

2. Common place data management tools.

Among the remarkable efforts undertaken so far, some are representative of the main avenues of development: MS-SharePoint, the web search engine (Google, Yahoo, etc.) and social networking systems (FaceBook, etc.). However:

- MS-SharePoint manages documents of various nature, but not necessarily the data contained in them.
- Search engines like Google, Yahoo, etc. use keywords, which can be loosely compared to a relational database structure, and successive searches, but are somewhat limited to formatted text and, for example, do not read inside databases, files, etc. and cannot interpret pictures and graphs.
- Social networking systems allow multiple sources to contribute to a given data set ("friends")

are allowed to post to an individual's page), but again data is exposed through text.

Perhaps cynically, it could be argued that the difference between the above and the eternal (and ubiquitous) card filing systems is "limited" to the global reach of the formers, and the filing cabinet bound nature of the latter.

3. CAD and non-CAD data

In a first, perhaps somewhat simplified approach, two generic data groups will be identified, *CAD* and *non-CAD*. *CAD* data is loosely intended to include all data that can and is explicitly generated and represented by "graphical" means, such as a line, a solid, etc. This data is generally created by CAD programs, or via graphical libraries (ex. Hoops, etc.). *non-CAD* data includes all data that is difficult to or cannot be represented by graphical means, such as relations between objects, number of objects, results of calculations, dates, etc. Nonetheless, this data can be and often is, an integral part of the message to be delivered in conjunction with or even by *CAD* data.

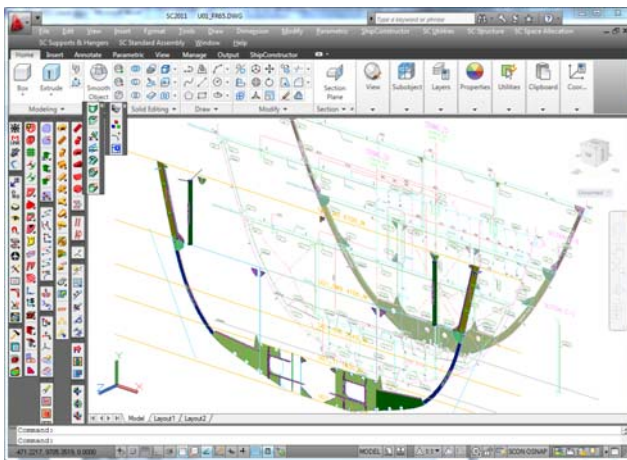


Fig.1: CAD data

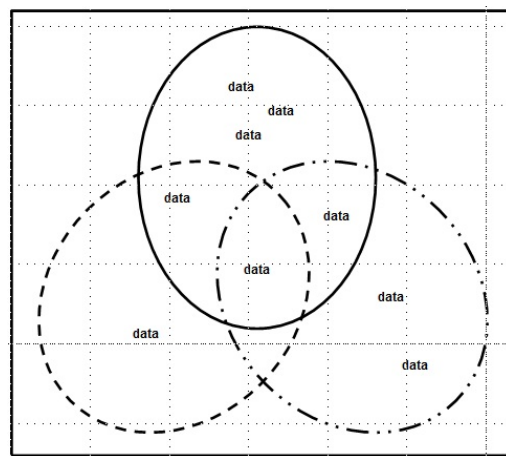


Fig.2: non-CAD data.

The word "graphical" undeniably lends itself to argument, so let us consider a debatable example: the concept of thickness. Thickness can be represented graphically, generally by hatching a section or of two parallel segments, but its value is generally expressed with a number, written as such.

4. Data structure, contextualization and inconsistency

Further to previous work on the nature of data and more specifically ship-related data, *Danese (2010)*, a high-level study of data structure and contextualization was undertaken, taking into account the implications of the intrinsic inconsistency of data.

4.1. Data structure.

"Data structure" refers to how the data is stored and, hence, made available. Data structure will depend on the nature of the data (somewhat universal), and on how the data will be interpreted in order to process and present it.

4.1.1. Data structure of CAD data.

Let us consider different ways to describe a line segment and a cylinder, while in each case using the same data set (one data set for the line and one for the cylinder).

These examples apply equally to lines and cylinders created via the graphical interface of a CAD program, via the programming of a graphical library, or via the use of neutral file formats, such as Auto-

desk's DXF. So, while the information needed to identify the line and the cylinder is always present in an explicit format, unless the structure of the format is known the data cannot be interpreted and used.

various ways to represent a line
<i>(0,0,0,1,1,1)</i>
<i>segment (0,0,0)(1,1,1)</i>
<i>x(0,1), y(0,1), z(0,1)</i>
<i>etc.</i>

various ways to represent a cylinder
<i>(r1,h3)</i>
<i>cylinder 1,3</i>
<i>cbr 1, ch 3</i>
<i>etc.</i>

4.1.2. Data structure of non-CAD data

Let us now consider a table of numbers, from which we wish to retrieve totals. Interestingly, such a table could be created using a CAD program or a graphical library, but in this case the numbers would not be treated as such unless the software is capable of managing and processing logical information, which is not always the case. The contents of the tables are the same (the individual numbers), but the totals will depend on the direction of summation (by row, by column, diagonally, etc.). Therefore, while containing the same data, the two tables shown hereafter will have to be interpreted differently. In the first case the summation will be conducted by column, in the second case the summation will be conducted by row:

1	2	3
1	2	3
1	2	3
3	6	9

1	1	1	3
2	2	2	6
3	3	3	9

For a slightly more abstract example, let us consider a common planning situation, such as the need to identify which parts of the ship are scheduled for assembly on a certain date, and to cross check the availability of a hired crane of sufficient lifting capacity to move the finished assembly to another location in the yard. While the ship parts can be represented graphically, and even the crane for that matter, cross-checking assembly completion date and crane availability probably cannot. To go one step further, let us consider the case of the crane becoming unavailable at the last minute, for example due to a mechanical failure, and the ensuing consequences due to the fact that the assembly hall remains unnecessarily occupied by the completed assembly.

Once again, CAD data (which parts of the ship will be affected) and non-CAD data (the planning corrective measures to be taken) are related and inter-dependent, and it will be beneficial for both data types to be documented in an exploitable fashion and be available together.

4.2. Data contextualization

For the purpose of this work, data contextualization is intended as the intrinsic meaning of the data itself. For example, a line could represent another object altogether, such as a stiffener, and a number could represent a weight or a frequency, or a person, etc..

The intrinsic meaning of data creates and controls the need for data to be available, found and interpreted in a given context. For example, weight data is relevant in the context of weight reporting but, more to the point, the context itself will drive the use of that data. For example, consider the need to either know the weight of an object or the total weight of a group of objects. In the first case we are dealing with direct data reporting, while in the latter we are dealing with data processing at first, and data reporting next. Going further, the total weight of one type of object might have to be multiplied by a unit cost, while the total weight of another group of object will be multiplied by a different unit cost. So, contextualization extends beyond the initial data itself, to the realm of data processing.

Data contextualization is generally achieved by the use of words, such as titles, descriptions, etc., which works well in the case of a printed document, but less so in the case of electronic data storage and transferring, as the method is limited and additional interpretation is required: for example, how many different meanings one same word can take depending on the context. Other, but simpler and more limited ways to contextualize data is to use colours, layers, file names, etc.

Data structuring and formatting is a pre-requisite to the contextualization of electronic data, and effective use of contextualization more often than not requires handling of the data itself, as opposed to simply accessing data storage. For example, the title of a drawing or of a file may indicate the presence of certain data, but will directly supply neither data itself nor its structure, which must be searched for, retrieved, interpreted and possibly processed.

4.2.1. Contextualization of CAD data

Going back to the example of the line and the cylinder, here are a few different possible meanings for these graphical entities:

possible meaning of a line
<i>distance</i> <i>trace of a stiffener</i> <i>centreline of a pipe</i> <i>edge of a plate)</i> <i>etc.</i>

possible meaning of a cylinder
<i>hollow pipe</i> <i>solid column</i> <i>tank</i> <i>hole in a solid</i> <i>etc.</i>

It can be argued that contextualization of CAD data is necessary for the data to be useful.

4.2.2. Contextualization of non-CAD data

non-CAD data is perhaps even more in need of contextualization. To use the tables of numbers example again, although the two tables contain the same numbers, each table may refer to different quantities, such as weight and thickness. Moreover, each number itself may represent something different. Hence, data structuring must be developed as required to correctly contextualize each piece of data.

4.3. Data inconsistency

While other words may be more fitting than "inconsistency", its general meaning accurately represents the practical problem presented by the need to associate data of the same type, but in different groups and to different extents. The theory of ensembles is a simple way to illustrate the above:

The practical task is to structure data in such a way that *CAD* and *non-CAD* data can be associated in any desired manner. For example, a line representing a stiffener may have a colour, the stiffener may have a different colour, and the planned date of assembly of that stiffener will not have a colour assigned to it at all. Yet, line, stiffener and date are associated. What is more, the colours themselves may be described differently, such as by name, by number, by pantone codes, by RGB percentages, etc. So, managing data inconsistency requires the documentation of relations and associations.

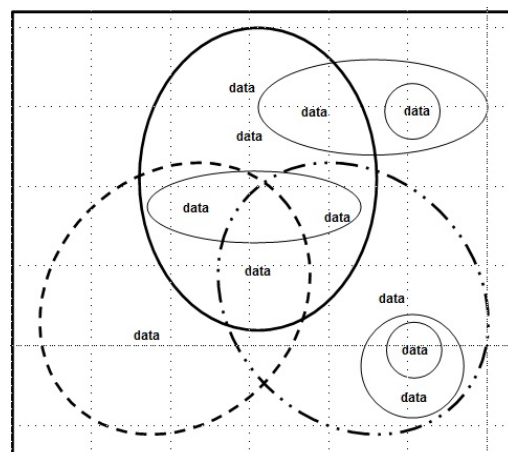


Fig. 3: Data inconsistency.

One effective way to stipulate associations is the use of relational data sets. Relational data sets are thereby contextualized to the deepest level, as required to associate the individual data elements. This

can be achieved by using potentially large numbers of files and an overall management software to link the data in the files, or by using relational databases.

5. Databases

Using files and corresponding management software does not offer unique, outstanding advantages and, another diminishing factor, carries the potentially disastrous disadvantage of the data not being collected in a single repository. This makes it very vulnerable to damage and loss, and difficult, if not impossible to share and to associate with data from other sources. Relational databases, such as SQL, can be as immediately human-readable as plain text files, and additionally offer several significant advantages:

- the data is collected in one container (the database file)
- scripts, queries and procedures provide a rational way to store, retrieve, pre- and post- process data, and insulate the data from the user thereof
- data integrity is protected
- the database engine automatically manages concurrent access to same data by multiple users
- association of data belonging to different databases is a feature by design
- the amount of data that can be contained in a single database and the number of associations are virtually unlimited

Moreover, reviewing ship data models, databases and associated CAD files, if any, tend to occupy 2.5 times less data storage space than file-only systems, on average. The author having direct experience mostly with the ShipConstructor database, this has been used in the research and development work presented hereafter.

6. The information model

A detailed review of the ship data model is presented in *Danese (2010)* and, in terms of data contents, it can be summarized by the following flowchart:

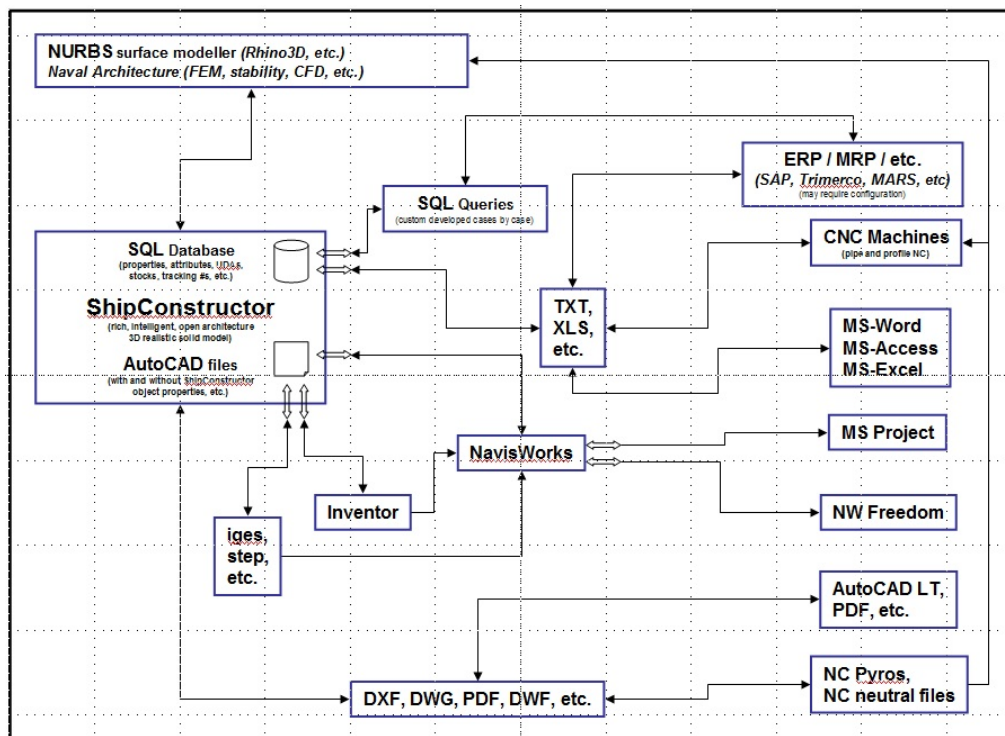


Fig.4: Ship Data Model flowchart

Differently than most if not all other ship data systems, it should be noted that the flowchart shows data flowing *back* into ShipConstructor's database. In other words, any *non-CAD* data directly related to any "object" in the ShipConstructor model can, and should be stored in the ShipConstructor database. Two crucial comments are in order here:

- "objects" in the ShipConstructor database are not limited to graphical and physical objects, such as lines or solids, but include logical entities, such as a branch in any of the user-defined hierarchical model structures, or a stock.
- the ability to collect and most importantly associate *CAD* and *non-CAD* data is the sine-qua-non criteria which must be satisfied in order to qualify a true data and information model as such. In fact, it becomes appropriate at this point to use the "information model" label in its full right, as the "data" connotation becomes, by definition, an underlying pre-requisite.

Moreover, it becomes evident that such an information model is no longer limited to the ship per-se, but involves and affects other, farther-ranging domains, such as company management, extra-ship PLM, etc. One final note, yet fundamental, note to the above considerations is that, of course, data can also flow from the ShipConstructor database to any other data repository, as the need may be. Therefore, the conditions to define an information model can be resumed to:

- it must be possible to associate *CAD* and *non-CAD* data, from different sources
- it must be possible to import data from and export data to other concerned software systems

7. The ShipConstructor information model: macroscopic structure and data flow

Macroscopically, the ShipConstructor information model is based on a single SQL database and on AutoCAD files. Data is exchanged and synchronised between the SQL database and the AutoCAD files via a direct, real-time, transparent interface. Moreover, data can be input from both inside and outside the ShipConstructor environment, via AutoCAD and via SQL.

7.1. Macroscopic structure

In short, the AutoCAD file contains a copy of every ShipConstructor object created in it (lines, solids, relations, associations, etc.), while the SQL database contains all ShipConstructor objects created either in AutoCAD files, and directly in or via SQL, for example using the graphical SQL database manager, as well as all the libraries of objects, relations, associations, etc. and the project's environment settings. This combination, and in some instances the managed duplication, of *CAD* and *non-CAD* data is fundamental to the achievement and exploitation of the multi-source input/output data flow described here after.

7.2. Data flow

While the words "import" and "export" describe the flow of data, these terms generally refer to the use of intermediate data exchange files. In the case of ShipConstructor, while classic import and export are supported, the tools available to exploit the data contained in ShipConstructor information model actually allow for the direct exchange of data, provided of course that other software involved in the process does, too.

Data flow is described in some detail in the following paragraphs, but one aspect thereof deserves foremost mentioning: the purposely built-in by-design ability to interact with the ShipConstructor information model using commercial off-the-shelf, shrink-wrapped tools, be they software in their own right (ex. Crystal Reports, Oracle/SQL interfacing software, MS-Office, etc.), or programming tools (ex. C#, VB, Lisp, etc.). This is possible thanks to several features offered by the ShipConstructor environment, such as:

- an increasing number of ShipConstructor commands are exposed in the AutoCAD environment and can be scripted.
- database interaction is made virtually transparent thanks to the ShipConstructor API, stored procedures, and exposed queries.
- access to raw data is permitted (but not encouraged).
- user-defined tables may be added to the ShipConstructor SQL database.
- the ShipConstructor SQL database may be linked to other databases.
- last and perhaps most powerful, the ability to interact with the ShipConstructor information model in a remote, IT neutral fashion, using standard communications technology.

Much work has already been carried out in this respect, and a few examples are presented hereafter.

7.2.1. Data input

In terms of data creation, several mechanisms are available in the ShipConstructor environment:

- *CAD* input, via the AutoCAD environment, using AutoCAD commands, ShipConstructor functions and non-ShipConstructor functions and programs.
- *non-CAD* input, via the dedicated ShipConstructor graphical SQL database interface, and via non-ShipConstructor functions and programs.

It is very important to note that reference is made here to "input", as opposed to "data". The distinction is fundamental, and it is made because the ShipConstructor user will create and input *CAD* and *non-CAD* data alike via both the AutoCAD and the SQL environments. So, graphical entities, modeling schemas, parameters, associations, etc. will be defined in some instances via the *CAD* AutoCAD environment, and in other instances via the *non-CAD* SQL database environment. A special note is in order at this point to underline the importance of *CAD* and *non-CAD* data associating mechanisms, specifically using input from multiple sources. One remarkable feature offered by ShipConstructor is the User Defined Attribute (UDA), a particularly powerful and versatile tool in several respects. For example:

- UDA data can be input from both inside and outside the ShipConstructor environment
- UDA data can be virtually anything: numbers, strings, hyperlinks, soon also formulas, etc.

7.2.2. Data output

Many output options are available, but a concise summary of output "types" will suffice here:

- graphical (ex. paper drawings, pictures, etc.)
- tabular (ex. reports)
- to neutral file formats (ex. dxf, pdf, dwf, rtf, etc.)
- to proprietary file formats (ex. MS-Excel, NavisWorks, etc.)
- via off-the-shelf software (ex. Crystal Reports, MS-Office, etc.)
- via custom applications (ex. targeted harvesting from the AutoCAD and SQL environments)

It is very important to note that data and information can be extracted from the ShipConstructor environment without need for AutoCAD and/or ShipConstructor, directly from project database. Therefore, the contents of the ShipConstructor information model become available to virtually anyone at any time, and more particularly they inherently become a dynamic component of company management, tactical and strategic decision making.

8. Development examples

Most of the few examples illustrated here have been contributed by several fellow researchers, who therefore deserve the credit, and have been chosen specifically to cover a range of different contexts:

- CAD environment: custom labelling of ShipConstructor parts.
- SQL environment: custom reports of ShipConstructor data.
- combination CAD / database environment: combining *CAD* and *non-CAD* ShipConstructor data into User Defined Attributes, in order to embed custom Product Hierarchy information into a standard ShipConstructor Bill of Materials table (BOM).
- SQL environment: external input from an IT neutral, non-ShipConstructor source.
- SQL environment: combining data from two ShipConstructor databases, then processing it.
- SQL environment: automating the acquisition of planning and scheduling information from third party suppliers using wireless technology such as WAN (2G : GSM / CDMAone, 3G : UMTS / CDMA2000, 4G : LTE / WiMax), LAN (Wi-Fi) or PAN (ZigBee).
- company scale example: exploitation of the open ShipConstructor Information environment in the early phases of estimation

8.1. CAD environment application: Custom labelling of ShipConstructor parts

Custom labelling was developed to accommodate specific drawing detailing and presentation requirements. Custom labelling is effectively an AutoCAD application, and uses data contained in the AutoCAD file. It directly harvests data from the ShipConstructor objects present in the current AutoCAD (as opposed to a X-ref file), and uses the harvested data to create formatted, persistent text strings (the parts' labels). The labels are stored in the file, and can be printed. The following example shows one possible code set for interrogating the ShipConstructor parts and harvest the data:

```

public static DetailInfo GetDetailInfo(Guid DetGuid, out Point3d p3d)
{
    DetailInfo DI = new DetailInfo();
    DI.GUID = DetGuid;
    double x = 0, x1 = 0, y = 0, y1 = 0, z = 0, z1 = 0;
    Guid guid = Guid.Empty;
    p3d = new Point3d(0,0,0);
    SCon.DataLayer.StructPart.STRUCT_PlateParts platePart = new
    SCon.DataLayer.StructPart.STRUCT_PlateParts();
    platePart.Connection = SConApp.ProjectSettings.Connection;
    platePart.DataLevel = SCon.DataLayer.Gen.DataLevelEnum.Full;
    platePart.LengthConversionUnit = GEN_LengthUnit.MM;
    platePart.FillPKGGUID(DetGuid);
    GEOM_3DPoints points3d = new GEOM_3DPoints();
    points3d.Connection = SConApp.ProjectSettings.Connection;
    points3d.DataLevel = DataLevelEnum.Basic;
    points3d.LengthConversionUnit = GEN_LengthUnit.MM;
    if (platePart.MoveNext() == SCon.DataLayer.Gen.DLBOOL.DLTRUE)
    {
        DI.Name = platePart.PlatePartName;
        DI.Stock = platePart.eStockDescription;
        DI.Material = platePart.eMaterialName;
        guid = platePart.ExtMaxPointGUID;
        points3d.FillPKGGUID(guid);
        if (points3d.MoveNext() ==
        SCon.DataLayer.Gen.DLBOOL
        .DLTRUE)
        {
            x = points3d.X;
            y = points3d.Y;
            z = points3d.Z;
        }
        guid = plate-
        Part.ExtMinPointGUID;
        points3d.FillPKGGUID(guid);
        if (points3d.MoveNext() ==
        SCon.DataLayer.Gen.DLBOOL
        .DLTRUE)
        {

```

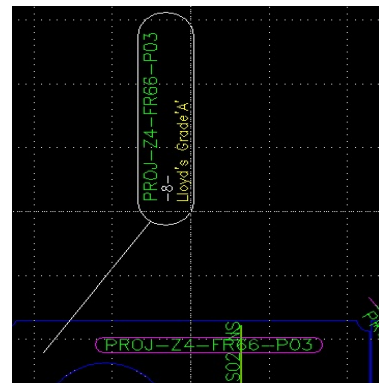


Fig.5: Custom label example.

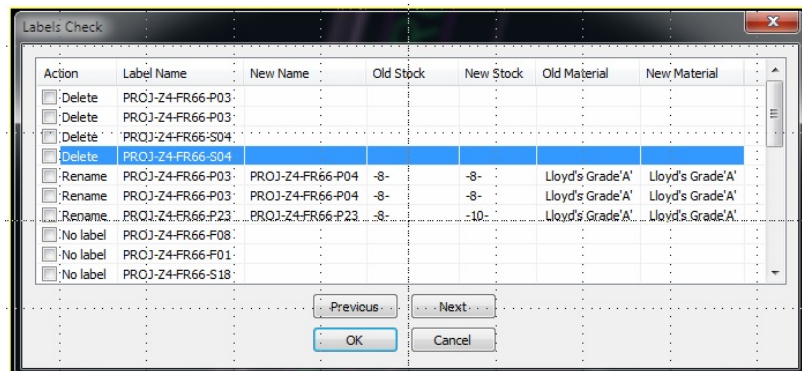


Fig.6: Custom labels management dialog.

```

x1 = points3d.X;
y1 = points3d.Y;
z1 = points3d.Z;
}
p3d = new Point3d((x + x1) / 2, (y + y1) / 2, (z + z1) / 2);
return DI;
}

```

8.2. SQL environment: custom reports of ShipConstructor data

Custom reporting was developed to accommodate specific report formatting and content requirements in a single report, and more specifically:

- custom-defined part count (quantities) and custom-defined part grouping.
- reporting user-selected part properties, including those embedded in the part object as text.

This custom reporting example is written in Crystal Reports, and only requires access to the SQL database. It will be helpful here to describe how a ShipConstructor part is "made". In short, it is composed dynamically from a collection of associated information including CAD data (geometry, text, et.) and non-CAD data (attributes, properties, associations, parameters, User Defined Attributes, etc.). So, custom reports are generated by harvesting data from the ShipConstructor database, and arranging into a custom-formatted report sheet, as follows:

- in the ShipConstructor database, find the required parts employing user-defined criteria.
- harvest the parts' information and collect it in a matrix.
- create a table using the desired parts and the desired, specific part information.

The following example shows one possible code set for interrogating the ShipConstructor database and harvest the required data:

```

void GetData()
{
int num = 0;
GEN_Units unitDataset = new GEN_Units();
STRUCT_PlateParts plateParts = new STRUCT_PlateParts();
STRUCT_PlatePartStringAttributes plateAtt = new STRUCT_PlatePartStringAttributes();
STRUCT_PlatePartStringAttributeValues
plateAttVal = new
STRUCT_PlatePartStringAttributeValues();
STRUCT_PlatePartBevels plateBevels =
new STRUCT_PlatePartBevels();
STRUCT_PlatePartFlangeStandardJoins
plateFS = new
STRUCT_PlatePartFlangeStandardJoins();
STRUCT_PlatePartTextMarkings plateTM =
new STRUCT_PlatePartTextMarkings();
STRUCT_ExtrusionParts extrusionParts =
new STRUCT_ExtrusionParts();
STRUCT_ExtrusionPartStringAttributes ext
PartsSA = new
STRUCT_ExtrusionPartStringAttributes();
STRUCT_ExtrusionPartStringAttributeValue
s extPartsSAV = new
STRUCT_ExtrusionPartStringAttributeValue
s();
STRUCT_ExtrusionPlots extPlots = new
STRUCT_ExtrusionPlots();
STRUCT_ExtrusionPlotSheets extPS = new
STRUCT_ExtrusionPlotSheets();
STRUCT_CorrugationParts corrParts = new
STRUCT_CorrugationParts();
STRUCT_CorrugationPartStringAttributes cpSA = new STRUCT_CorrugationPartStringAttributes();

```

Part Name	Stock	Material	Thickness	NestName	Quantity	Drawing
5800U01-BD-02	PL06	Grade A	6	5800-PL06-030	1	U01_BRIDGEDECK
5800U01-BD-03	FL06	Grade A	6	5800-PL06-037	1	U01_BRIDGEDECK
5800U01-BD-04	PL06	Grade A	6	5800-PL06-042	1	U01_BRIDGEDECK
5800U01-BD-05	HP 140	Grade A			2	U01_BRIDGEDECK
5800U01-BD-06	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-07	HP 140	Grade A			2	U01_BRIDGEDECK
5800U01-BD-08	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-09	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-10	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-11	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-12	HP 140	Grade A			1	U01_BRIDGEDECK
5800U01-BD-13	HP 140	Grade A			5	U01_BRIDGEDECK
5800U01-BD-14	HP 120	Grade A			1	U01_BRIDGEDECK
5800U01-BD-15	HP 120	Grade A			5	U01_BRIDGEDECK
5800U01-BD-16	HP 140	Grade A			4	U01_BRIDGEDECK
5800U01-BD-17	HP 140	Grade A			1	U01_BRIDGEDECK
5800U01-BD-18	HP 140	Grade A			1	U01_BRIDGEDECK
5800U01-BD-19	FL08	Grade A	8	5800-PL08-015	1	U01_BRIDGEDECK
5800U01-BD-20	FB 80x8	Grade A			1	U01_BRIDGEDECK
5800U01-BD-21	FL08	Grade A	8	5800-PL08-025	1	U01_BRIDGEDECK
5800U01-BD-22	FB 80x8	Grade A			1	U01_BRIDGEDECK
5800U01-BD-23	FL08	Grade A	8	5800-PL08-010	1	U01_BRIDGEDECK
5800U01-BD-24	FL08	Grade A	8	5800-PL08-001	1	U01_BRIDGEDECK
5800U01-BD-25	FL08	Grade A	8	5800-PL08-021	1	U01_BRIDGEDECK
5800U01-BD-26	FB 80x8	Grade A			1	U01_BRIDGEDECK
5800U01-BD-27	FL08	Grade A	8	5800-PL08-019	1	U01_BRIDGEDECK
5800U01-BD-28	FB 80x8	Grade A			1	U01_BRIDGEDECK
5800U01-BD-29	FL07	Grade A	7	5800-PL07-041	1	U01_BD
5800U01-BD-30	FL07	Grade A	7	5800-PL07-040	1	U01_BD
5800U01-BD-31	FL07	Grade A	7	5800-PL07-045	1	U01_BD
5800U01-BD-32	FL07	Grade A	7	5800-PL07-045	1	U01_BD
5800U01-BD-33	FL07	Grade A	7	5800-PL07-038	1	U01_BD
5800U01-BD-34	FL07	Grade A	7	5800-PL07-038	1	U01_BD

Fig.7: Custom reports, management dialog.

```

STRUCT_CorrugationPartStringAttributeValues cpSAV = new
STRUCT_CorrugationPartStringAttributeValues();
STRUCT_CurvedPlateParts currPlateParts = new STRUCT_CurvedPlateParts();
STRUCT_CurvedPlatePartStringAttributes cppSA = new STRUCT_CurvedPlatePartStringAttributes();
STRUCT_CurvedPlatePartStringAttributeValues cppSAV = new
STRUCT_CurvedPlatePartStringAttributeValues();
STRUCT_TwistedExtrusionParts tep = new STRUCT_TwistedExtrusionParts();
STRUCT_TwistedExtrusionPartStringAttributes tepSA = new STRUCT_TwistedExtrusionPartStringAttributes();
STRUCT_TwistedExtrusionPartStringAttributeValues tepSAV = new
STRUCT_TwistedExtrusionPartStringAttributeValues();
STRUCT_PlankParts pp = new STRUCT_PlankParts();
STRUCT_PlankStockStringAttributes ppSA = new STRUCT_PlankStockStringAttributes();
STRUCT_PlankStockStringAttributeValues ppSAV = new STRUCT_PlankStockStringAttributeValues();
GEOM_3DPoints points3d = new GEOM_3DPoints();

```

8.3. Combination CAD / database environment: combining CAD and non-CAD ShipConstructor data into User Defined Attributes, in order to embed custom Product Hierarchy information into a standard ShipConstructor Bill of Materials table (BOM)

This application exploits the flexible nature of User Defined Attributes (UDA) to create and document a custom-defined, post-processing hierarchical sequence. The post-processing sequence is described in a verbose fashion and, in this case, uses the names of Product Hierarchy branches according to the company's specific production strategy. The post-processing sequence is documented in the Nest Drawing, by embedding the desired UDAs into the user-defined ShipConstructor Bill of Materials table (BOM) - the BOM is laid out using keywords, which refer to data in the database. The custom information is used by the NC-machine operator to sort the cut parts according to each part's post-processing sequence. This a mixed AutoCAD / ShipConstructor application, using data from both the AutoCAD file and from the project database. The application works as follows:

- interrogate the Nest file to identify the parts therein (AutoCAD environment)
- then, from the SQL database, harvest the Product Hierarchy structure corresponding to each part, starting at the individual part's level, then upstream (SQL environment)
- for each part, populate the corresponding UDAs in the database with the appropriate branch name (write to the database)

The following example shows a possible code set for populating the UDAs by writing to the database, after having interrogated the ShipConstructor Nest file to identify the parts present, and having harvested their relevant data from the database.

```

while (plateParts.MoveNext() ==
DLBOOL.DLTRUE)
{
bool find = false;
try
{
if (plateParts.ePrimaryAssemblyGUID == null)
{
continue;
}
}
catch { continue; }
string attVal =
ChangeAttributeValue(plateParts.ePrimaryAssemblyGUID, ref find);
if
(!checkCondition(plateParts.ePrimaryAssemblyGUID))
{
continue;
}
}
if (find)
{

```

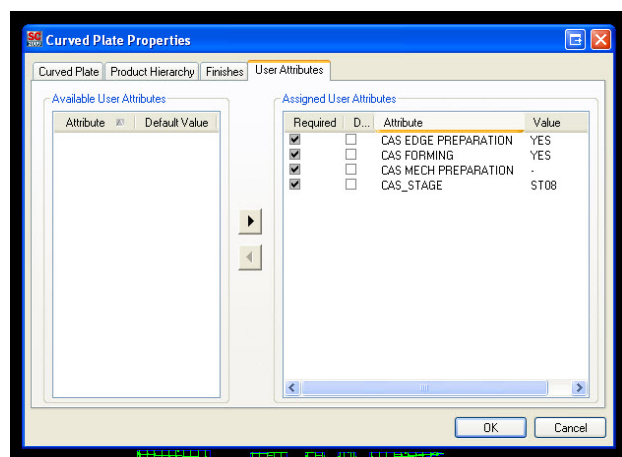


Fig.8: Selection dialog: choosing the UDAs to be populated with Product Hierarchy branch names.

```

plateAttVal.Clear();
plateAttVal.FillFromPlatePart(plateParts.PlatePartGUID);
try
{
    while (plateAttVal.MoveNext() == DLBOOL.DLTRUE)
    {
        plateAtt.FindGUID(plateAttVal.PlatePartStringAttributeGUID);
        if (attibutes[plateAtt.StringAttributeGUID] == AttName)
        {
            numAtt++;
            plateAttVal.AttributeValue = attVal;
            plateAttVal.IsDeferred = DLBOOL.DLFALSE;
            plateAttVal.Update(UpdateMethod.AbortOnError);
        }
    }
}
catch
{
}
}

```

ST	PART	Qty	EDGE	FORM	MECH	No
ST01	U02--01-96-001/DK07	1	YES	YES	-	1
ST01	U02--01-97-010/LG08	1	-	-	-	2
ST02	U02--04-002/LG05	1	YES	-	-	3
ST03	U02--06-01-001/DK06	1	-	-	-	4
ST01	U02--14-01-029/DK40	1	YES	-	-	5
ST01	U10--14-30-009/LG11	1	-	-	-	6
ST01	U10--14-33-001/LG11	1	-	-	-	7
ST01	U10--14-45-009/DK08	1	-	-	-	8

Fig.9: Part of a BOM with UDAs.

8.4. SQL

environment:

external input from an IT neutral, non-ShipConstructor source.

The example described here is a small part of a much larger ShipConstructor-assisted company management environment. Its peculiarity is that its functionality is available through any html compatible environment, irrespective of operating system or device brand. For lack of a proper name, it is referred to as RDM (remote data management).

The goal is to "de-materialize" access to and management of selected ShipConstructor data contained in the SQL database, under the MS-Windows OS. Of course not all data should be modifiable from the outside, and not everyone should have access to all data. The ShipConstructor database is accessed via IIS services, which are a standard part of the Windows OS distribution and installation. This is achieved very simply by a password controlled log-on, which also allows one to dynamically create a user-specific graphical interfaces, tailored to cater to the user's requirements and assignment, in accordance with his permission clearance. While programmatically rather complex, the application is very simple in concept, and works as follows:

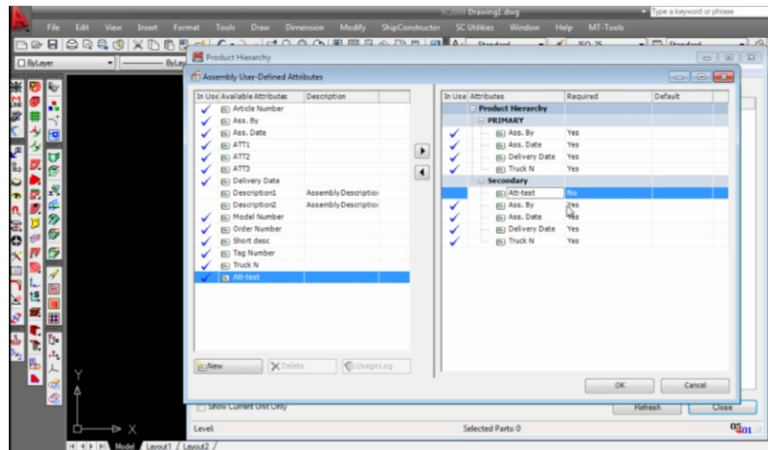


Fig.10: Defining a UDA in ShipConstructor..

- from any html compatible device connected to the internet or to a LAN/WAN network, the user connects to the IP address of the PC hosting the RDM application.
- the RDM sends the log-on html GUI to the device.
- the user logs-on with name and password.

- the RDM sends the user-specific GUI to the device.
- the user operates using keyboard, mouse, touch-screens, etc.
- the RDM establishes and manages the connection to the database.

While much of the source code is proprietary and is not presented in this document, interface read and write examples are illustrated here.

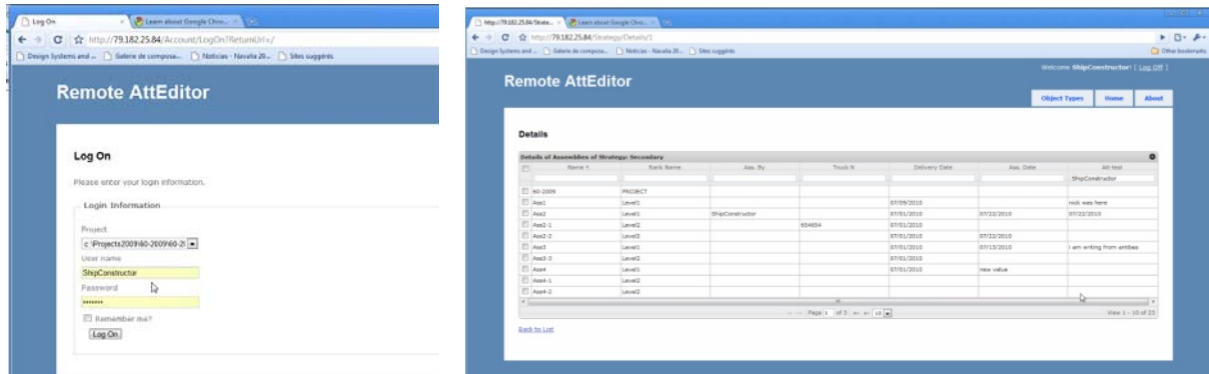


Fig.11: The RDM log-on/off dialog in Google Chrome (note the IP address in the url field), and the dialog for editing UDAs via the remote html interface.

The application has been tested successfully using the following devices: PC (various versions of Windows and web browsers), BlackBerry, Android mobile phones, iPhone, etc.

8.5. SQL environment: Combining & processing data from two ShipConstructor databases, then processing it.

Custom reports can be as simple as specially formatted tables, or as complex as full data processing engines involving data from different sources. In addition, with reference to other examples in this document, the results of data processing can be in turn written back to the desired database(s), for example by populating a UDA. This allows for data processing to take place outside the ShipConstructor environment, while collecting and storing raw data and results in it. Custom reports are based on data harvesting, and subsequent data processing. With reference to the examples already presented in this document, the Crystal Reports document illustrated below harvests data from two ShipConstructor databases and processes it into a combined result.

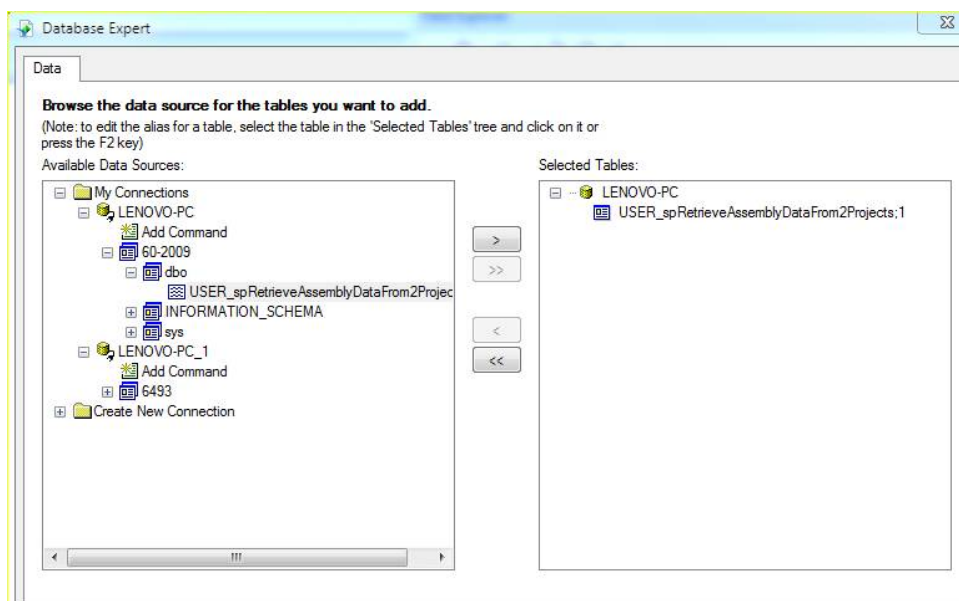


Fig.12: Crystal Reports - connecting to 2 different databases.

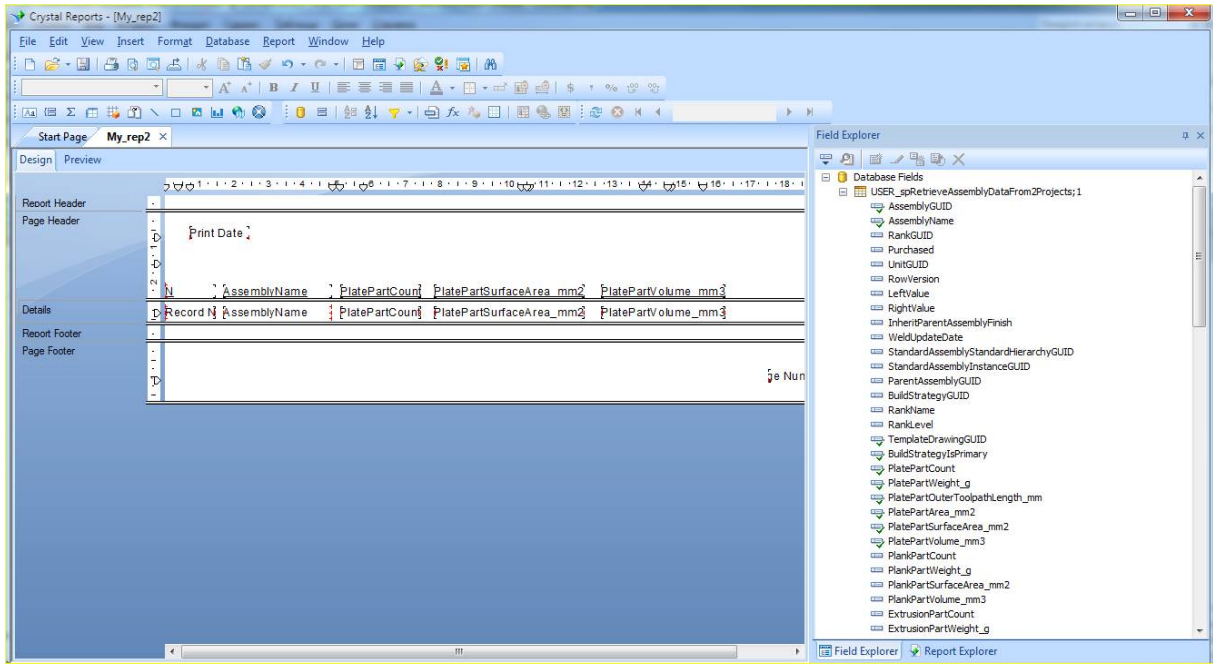


Fig.13: Crystal Reports - drag & drop the desired fields from the database tables to the report layout.

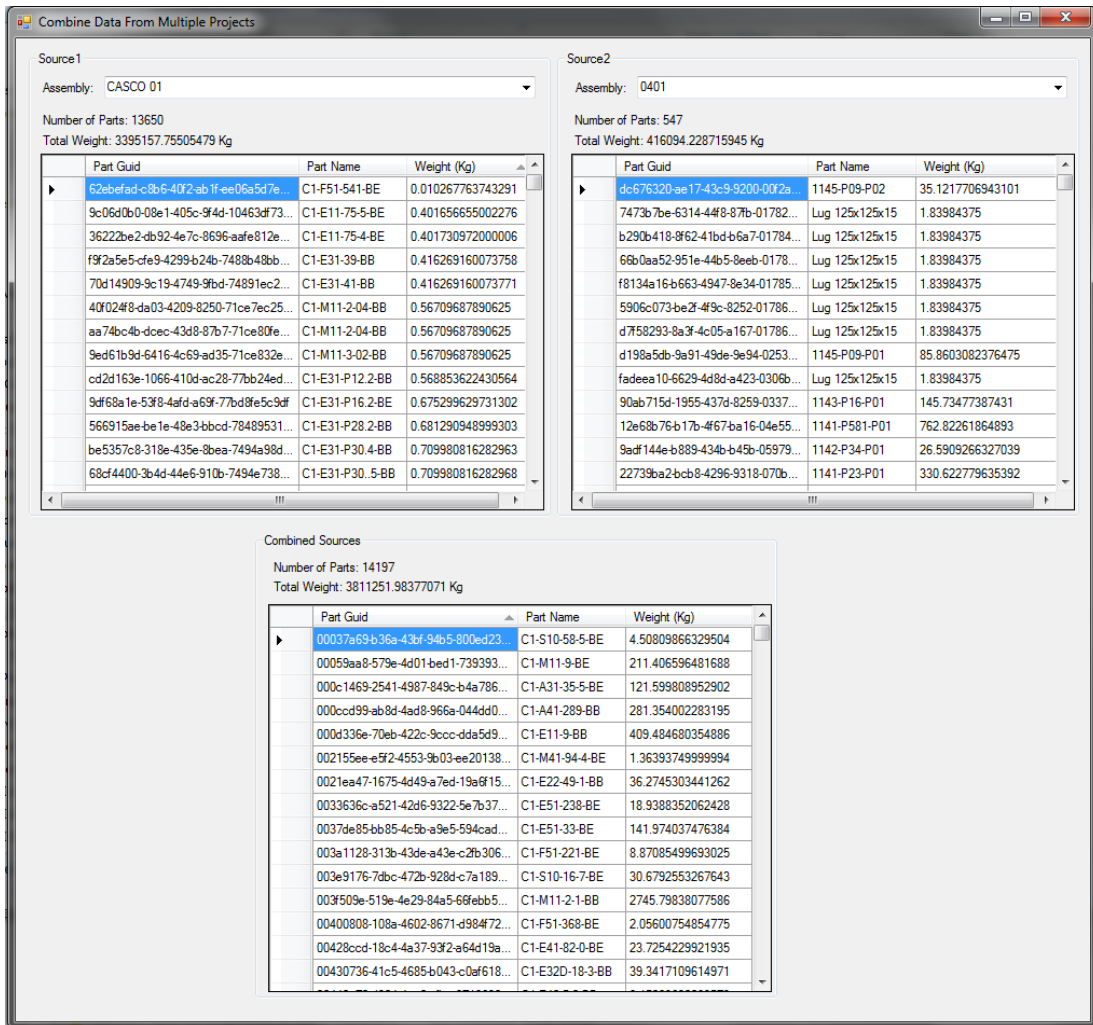


Fig.14: The data from 2 different SQL database sources, and its processing into overall totals.

8.6. Exploitation of the open ShipConstructor Information environment in the early phases of estimation

The following information flow schema was developed to exploit the open ShipConstructor information environment as of the early phases of estimation. The schema is presented succinctly, and reference is made to the information presented so far.

8.6.1. The ShipConstructor information environment comprises (summary)

- AutoCAD drawings, used for 3D model input and output. It is a familiar, configurable environment, and, contains both ShipConstructor and non-ShipConstructor data.
- 3D realistic, solid model: attribute and property rich. Properties include *CAD* and *non-CAD* data. Model is fully integrated and multi-disciplinary, gives overall geometrical and non-geometrical data.
- SQL database: offers export facilities, supports external harvesting of data, reports include user-selected data in user-defined formats, etc.

8.6.2. Estimation using the single model approach, extensible to design and production

This approach consists in modelling a scalable, representative Unit in detail, and the rest of the ship macroscopically (scalable primary structure such as main frames, bulkheads, decks, etc., representative secondary structure, primary and large schedule distributed systems, etc.). This data and ensuing information is then used as-is or further processed, for example as follows:

- as qualitative input to create or validate preliminary scheduling and planning, for example in MS Project.
- to feed preliminary purchasing and required resources estimations (ERP/MRP) and other administrative and management tools (ex. SAP), which in turn will influence the preliminary planning and scheduling.

The CAD portion of the data is computed automatically and stored as object properties, such as weights, number of plates to cut, weld lengths, etc. The *non-CAD* portion of the data will include user defined processing entities, such as blasting grit grade, paint and finish specifications, assembly sequence, etc., and user specific parameters such as manufacturing hours as a function of yard resources and task complexity, etc. *non-CAD* data is yard specific, user supplied and added to the model via object properties and User Defined Attributes.

Information model dependent reports include the relevant data from the SQL database, sorted and grouped according to the user's requirements, either using the ShipConstructor reporting engine, or other software, like Crystal Reports. Data is further processed as required to obtain the relevant quantities for use in management, tactical and strategic decision making.

Raw and processed data is thus available for immediate use, as direct input for building an MS-Project Gantt chart, or for further processing by ERP/MRP and management software.

Other *non-CAD* information also influences the planning (availability of materials and resources, other objective constraints) and the structure of the ShipConstructor Information model. For example, the availability of steel may influence the planned assembly sequence.

8.6.3. Integration of planning, cost analysis and model

One or more iterations may be needed to synchronise the model (choice of yard processing and build sequence, CAD modelling schedule to produce required data, etc.) and the overall planning, also with respect to cost and other objective constraints. Estimated and projected data are input back into the model (ex. model "by" dates, nest "by" dates, names of assigned operators, revised processing infor-

mation such as finish, assigned resources, etc.), principally in UDAs associated with model parts and *non-CAD* objects (ex. the projected elapsed time needed to complete the assembly of a branch of a Product Hierarchy based on available resources at the planned moment in time and on objective difficulty and complexity factors, as well as the corresponding cost). As the information is now collected and managed by several connected tools, it becomes appropriate to use the Information Model definition, the ShipConstructor Information Model being part thereof. As cross-feedback balances out, the Estimation model can be considered achieved.

From this point on, data flow continues to be managed at least multi-directionally between ShipConstructor, planning / scheduling software (ex. MS Project), ERP/MRP software (ex. MARS) and management software (ex. SAP), but perhaps also to/from other data sources (ex. a subcontractor suffers a set-back, or a strike forces re-tasking and re-scheduling, or bad weather delays painting, or the price of a sourced item drops thereby allowing its purchase and its fitting sooner, etc.).

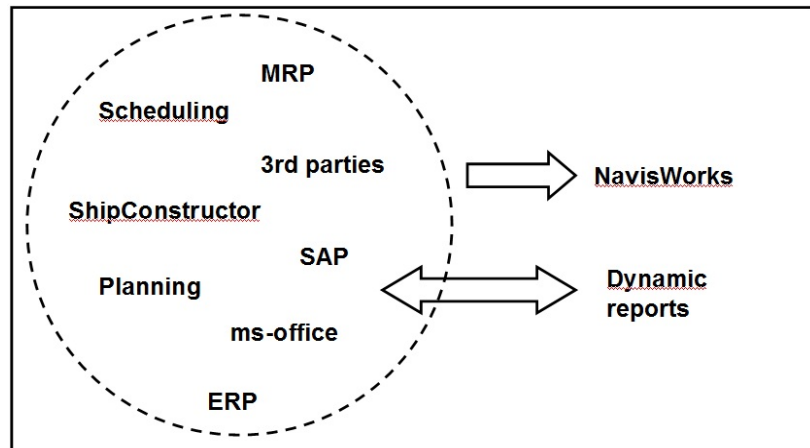


Fig.15: The Information Model.

8.7. SQL environment:

automating the acquisition of planning and scheduling information from third party suppliers using wireless technology such as WAN (2G : GSM / CDMAone, 3G : UMTS / CDMA2000, 4G : LTE / WiMax), LAN (Wi-Fi) or PAN (ZigBee).

The ability to write information to the ShipConstructor database opens innumerable doors towards catering to many management requirements. The example presented here is based on a rather common circumstance, the geographical distance between a cutting company and the shipyard(s) where those parts will be assembled. In fact, the geographical distance often entails crossing of borders varying transport regulations, perhaps vastly different meteorological conditions, traffic jams, etc. Therefore, real-time knowledge of the location of a truck transporting cut parts is of great interest. While geo-localization has become common place technology, it was sought to make available relevant information about the cargo being carried in an operator-less fashion. Therefore, programmable active/passive RFID technology was added to existing GPS/GSM geo-localization. In short, the planned system, based on current off-the-shelf technology and hardware, will work as follows:

- upon loading of cut parts on a pallet or flat rack, an RFID tag attached to the pallet or to the flat rack is programmed with a tracking number and the list of parts (obtained from the NC-cutting file or other ShipConstructor supplied information).
- upon leaving the premises, the tag connects with the active transponder of the RFID system, and the following information is sent to the Information Model via GSM: pallet / rack tracking number, list of parts.
- at pre-set time intervals, the geo-localisation portion of the tag signals its GPS position, which

is fed to the Information model in wireless fashion.

- finally, upon entry of the shipyard's grounds, another active transponder reads the tag's contents and feeds the arrival data to the Information model

Incidentally, it is interesting to note that, when produced in small quantities, the cost of the tag and of the corresponding active RFID transponder described above have been estimated to be a few tens of Euros, negligible at the scale of the ship industry.

9. Conclusion

The use of off-the-shelf technology and tools to compose and exploit a true Information Model has been documented and advocated before, *Danese (2010)*. The few examples of completed and on-going development work discussed here represent only a small cross section of what has already been achieved, and are aimed at documenting the ease and simplicity of building the basic bricks of an Information Model, a goal within just about everyone's reach, given that the required data lends itself to the exercise. Of course, requirements change as new tools become available, and vice-versa. However, experience would suggest that the use of off-the-shelf software, hardware and technology greatly reduces the amount of resources to be expended in catering to change. This intrinsically scalable approach is then deemed viable.

Acknowledgements

A special thanks for supporting the author's research work is owed to Pedro Antunes (Vera Navis, Portugal), Luis Batista (Vera Navis, Portugal), Stefano Cipriani (Nice RFID, France), Darren Larkins (ShipConstructor Software Inc., Canada), Deniz Morais (ShipConstructor Software Inc., Canada), Lambertus Oosterveen (Royal Huisman, The Netherlands), Justin Paquin (ShipConstructor Software Inc., Canada), Stefan Raev (Royal Huisman, The Netherlands), Peykan Sahelnia (PS Development, France), Arkadiy Zagorskiy (Marine Technologies, Russia)

References

DANESE, N. (2010), *Ship CAD systems - past, present and possible future*, COMPIT, Gubbio, pp. 396-408