2005-08-01

The application of Advanced Solid Modelling Techniques to the Design of Precision Air Handling Units for Manufacture

Senan Moynihan
Technological University Dublin

The application of advanced solid modelling techniques to the design of precision air handling units for manufacture

Senan Moynihan BSc (Eng) Dip Eng MIEI

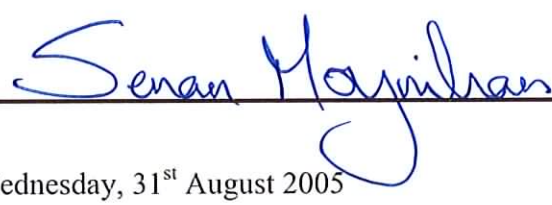
Award:	MPhil
Name of Institute:	D.I.T. Bolton Street
Supervisor:	Mr. William Reddington
School:	Engineering
Department:	Manufacturing
Month and year of submission;	August, 2005
Volume number and total number of volumes:	01 of 02

Declaration Page

I certify that this Thesis which I now submit for examination for the award of MPhil, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This Thesis was prepared according to the regulations for postgraduate study by research of the Dublin Institute of Technology and has not been submitted in whole or in part for an award in any other Institute or University.

The Institute has permission to keep, to lend or to copy this Thesis in whole or in part, on condition that any such use of the material of the Thesis be duly acknowledged.

Signature: 

Date: Wednesday, 31st August 2005

Senan Moynihan BSc (Eng) Dip Eng MIEI

Personal Identifier – MPhil F/T

Acknowledgements

I would like to take this opportunity to express my appreciation and eternal gratitude to all the people who gave me help and encouragement through out the course of my research. They are

My supervisor, Mr. Bill Reddington coached and mentored me since the final year of my degree. He has been a great influence through the later stages of my educational development, I will always be grateful for all he has done for me.

To all the staff of Danann Clean Air Services Ltd, in particular Bríd Sharkey, Connor Moore, Dave McKay and Michael de Burke. They opened the doors of Danann to me and showed me everything I needed to know about how to design and manufacture air-handling units.

Everybody in Dedicated CAD Systems Limited, Martin and Bill Reddington gave me great support while I was working on my research and they gave me endless guidance and support. Keith Clarke and Alan McDonnell were always on hand to help me and advise me as best they could, thank you for everything.

Finally I would like to give special thanks to Robert Simpson and all the staff at Dublin Institute of Technology, Bolton Street who gave me exceptional guidance and resources through out the course my research.

Abstract

This thesis investigates the application of ActiveX technology to customise mid range solid modelling software, with a view to exploring the benefits small to medium sized manufacturing companies can experience from its successful implementation.

A custom application intended to automate solid modelling software was developed that optimises the product design process and also generates the necessary manufacturing information, including working drawings, for precision air handling units. This was successfully accomplished through research and by integrating Computer Aided Design (CAD) with a custom application using ActiveX technology.

A variety of CAD products and associated technologies i.e. solid modelling kernels, were researched in order to investigate which product was most suitable to select. The products were assessed on functionality with heavy emphasis also placed on the potential increases in efficiency and productivity after customisation. A requirement of this research was to find a mid range modelling system that could be programmed to automate the entire design and modelling process. The system chosen was 'Solid Edge' from EDS PLM solutions.

Extensive research was carried out on ActiveX and Data Access technology. ActiveX technology was used to gain access to the Solid Edge and customise it, while Data Access was the technology used to access a database and apply previously stored information to solid models in order to automatically format their characteristics and dimensions.

By successfully integrating ActiveX and Data Access technology with CAD it was possible to undertake a case study which involved redesigning and instigating a new design solution for a manufacturer of precision air-handling units. The case study provided the requisite framework to further develop and test the custom application and to expand on the research project's objectives.

The case study proved that by successfully implementing the custom application the time required for designing products as well as producing the necessary manufacturing information was radically decreased thus improving productivity, efficiency and also reducing the time to market for the companies products.

Table of Contents

Table of Figures

Table of Tables

Table of Formulae

Chapter 1: Introduction

This thesis investigates the application of ActiveX technology to customise mid range solid modelling software, with a view to exploring the benefits small to medium sized manufacturing companies can experience from its successful implementation.

A custom application intended to automate solid modelling software was developed that optimises the product design process and also generates the necessary manufacturing information, including working drawings, for precision air handling units. This was successfully accomplished through research and by integrating Computer Aided Design (CAD) with a custom application using ActiveX technology.

A variety of CAD products and associated technologies i.e. solid modelling kernels, were researched in order to investigate which product was most suitable to select. The products were assessed on functionality with heavy emphasis also placed on the potential increases in efficiency and productivity after customisation. A requirement of this research was to find a mid range modelling system that could be programmed to automate the entire design and modelling process. The system chosen was 'Solid Edge' from EDS PLM solutions.

Extensive research was carried out on ActiveX and Data Access technology. ActiveX technology was used to gain access to Solid Edge and customise it, while Data Access was the technology used to access a database and apply previously stored information to solid models in order to automatically format their characteristics and dimensions.

By successfully integrating ActiveX and Data Access technology with CAD it was possible to undertake a case study which involved redesigning and instigating a new design solution for a manufacturer of precision air-handling units. The case study provided the requisite framework to further develop and test the custom application and to expand on the research project's objectives.

The case study proved that by successfully implementing the custom application the time required for designing products as well as producing the necessary manufacturing information was radically decreased thus improving productivity, efficiency and also reducing the time to market for the companies products.

1.1 Computer Aided Design (CAD)

In the last ten years there have been major advances in CAD technology, most of which are seen in the form of increased functionality, greater stability and increased availability on the Microsoft Windows operating systems. In particular mid-range solid modelling products have progressed well with their ability to compete with high end CAD products on a mechanical design level. In the Literature Review CAD is described along with the selection of product types available in its industry at the present time. The three definitive categories of CAD available for the mechanical design and the manufacturing industries are:

- Low-end
- Mid range
- High end

CAD products are generally categorised by their purpose of design, level of functionality and price. Typically, companies with less than 200 seats implement mid-range systems and as a result it was decided to focus the research on such solid modelling systems. Greco, J., 2003 ([1]www), states that within the mid-range solid modelling industry the most prominent leaders are:

- Solid Edge, EDS PLM Solutions
- Solid Works, Solid Works Corporation
- Inventor, AutoDesk
- Pro/Engineer, PTC - Parametric Technologies Corporation

1.2 Programming utilities

Each year CAD products are updated and generally improved in order to advance usability, functionality along with productivity and efficiency.

Current stand alone CAD products go along way in automating the design process of products, however, the process for constructing parts and sub-assemblies can be repetitive, labour intensive and time consuming. In order to optimise the use of CAD products i.e. reduce the time to produce and revise designs, it was felt that a bespoke

application could be developed to automate the process and eliminate manual and repetitive practices. To achieve this, it would be necessary to integrate the specific CAD product with other technologies and third party programming utilities.

The completed bespoke software application was developed to construct the precision air-handling units in three stages:

- Exterior aluminium frame assembly
- Interior subassembly design and positioning
- Exterior panel assembly and retrofit

The air-handling unit's automated design process was implemented in this fashion so to best resemble that of the manual design and manufacturing practice.

The literature review also outlines details on how companies can develop custom applications to integrate with and automate their solid modelling software systems. The technology used to develop such automated systems including Solid Edge is also described in that chapter.

1.3 Methodology

It was possible to approach the design of the custom software application in two ways.

- Option (A): Write an application to build each part feature by feature. The application would build each sub-assembly part by part and the top level assembly would finally be built sub-assembly by sub-assembly.
- Option (B): Write an application maintaining the same level of effectiveness but with less amounts of programming. This was to be achieved by previously generating all the parametric parts and subassemblies known to make up an air-handling unit. Once the parts were created the application was programmed to use these parts and sub-assemblies to construct the top level assembly.

It was decided to proceed with Option (B). Proceeding with the first option would lead to excessive amounts of programming and code testing. With Option (A) the completed solid models would contain fewer parametric features making the overall automated

system less flexible. The Option (A) approach would also develop a system that would make design revisions more difficult and take longer to complete.

1.4 Overview of information

Chapter 2 contains the Literature Review. The Literature Review contains an analysis of the types of computer aided design options presently available. The characteristics present in each type of computer aided design system are described in detail including an analysis of the package used in this research. The technologies used in modern computer aided design packages including the types of solid modelling kernels are also analysed. The methodology for customising solid modelling software is described and analysed including the technology available to do so.

In Chapter 3 the use of Data Access technology in the custom written application is explained by describing how applications can be programmed to access databases and their information. Chapter 3 also describes the technical aspects of how Data Access technology integrates with both the bespoke application and the solid modelling software to apply information to create and modify solid models.

Chapter 4 outlines the procedure and the preparatory solid modelling work necessary prior to developing a custom application for automating the design of an air-handling unit assembly.

Chapter 5 describes a case study on a Dublin based manufacturing company i.e. Danann Clean Air Services Ltd. This chapter outlines the products that Danann manufacture in addition to the components that are used for their make up. Danann's main business element is the design and manufacture of precision air-handling units for the Irish market. Each air-handling unit Danann manufactures is individually designed making each of Danann's air-handling unit's unique. Because each air-handling unit is unique, for every air-handling unit, time must be spent designing it and creating the necessary manufacturing documentation. This is a long process taking several hours depending on each air-handling unit, in some cases it takes much longer depending on design revisions. The information collated in the case study regarding the make up of each air-handling unit forms the basis for the design of the database prepared to hold information that describes the technical specifications of the air-handling units.

Chapter 6 describes, in detail, the design and implementation of the custom application developed to construct the air-handling unit's. The algorithms and processes that the application uses to build and construct solid models are also described in this chapter along with how it integrates solid modelling software with ActiveX technology. The complete code for the custom application can be seen in Appendix C. Chapter 6 details a post development and implementation critical assessment of the bespoke software.

Chapter 7 contains the conclusions formulated from this research. It was found that after implementing the application considerable time savings were achieved in product design and drawing manufacture. However these savings must be compared to the time necessary developing the application and the further time and resources it will take to modify the application should the product itself or its components be altered.

1.5 Background

It is important to the author to perform this research to clearly accentuate to small to medium sized companies what can be achieved, with respect to efficiency, productivity including savings in time, Engineering Change Orders (ECOs) and costs, by successfully investing resources into implementing a modern custom developed computer aided design and manufacturing system.

It is the author's belief that companies currently operating with basic computer aided design solutions could experience many advantages after successfully migrating to a modern, customised solution designed to meet the specific needs of the company.

Chapter 2: Literature Review

2.1 Introduction to CAD

Walker, J., 1986, states that CAD is the modelling of physical systems on computers, allowing both interactive and automatic analysis of design variants and the expression of designs in a form suitable for manufacturing. Kutz, M., 1998, explains how CAD uses both mathematical and graphic processing power from the computer to help the user create, modify, analyse, and display designs. Kutz, M., 1998, also explains how in recent times there have been many factors that have contributed to CAD technology becoming a more important tool in the modern engineering world e.g. the computer's speed at processing complex equations. CAD is often thought of simply as computer-aided drafting; its use as an electronic drawing board is a powerful tool in itself but the functions of a CAD system go far beyond its ability to represent and manipulate graphics. Geometric modelling, engineering analysis, simulation, and the communication of the design information can also be performed using CAD.

2.2 A comparative analysis of CAD

As discussed in Chapter 1 there are multiple CAD products available on the market. These products can be graded into three main categories.

- Low-end CAD systems
- Mid-range CAD systems
- High end CAD systems

A low-end CAD system's functionality is typically limited to a 2D environment with a small degree of 3D functionality e.g. AutoDesk's AutoCAD LT. Mid-range CAD systems in contrast, are more intuitive and advanced allowing the design process to evolve from a 3D environment. High end CAD systems contain much similar functionality as mid-range systems. One of the main features that separates the systems are that high-end systems contain the ability to develop parts and assemblies with greater ease e.g. Dassault Systemes' CATIA.

Some mid-range CAD systems are commonly referred to as "solid modelling" systems. With products such as Solid Works, Solid Edge and Inventor, completed designs are graphically represented in a 3D environment. From the 3D environment designs can be viewed from an infinite number of positions in both isometric and perspective. The process for developing designs is also performed in a 3D environment where each part is generated from a series of 3D features resembling blocks, spheres, cones and cylinders.

Unlike mid-range systems, low-end systems do not contain the functionality to generate a set 2D views by referencing a solid model. This implies each drawing view within a set produced from a low-end system must be created as an independent entity. Therefore, when an adjustment is made to one view on a set of drawings, the adjustment will have no affect on any of the remaining views. For the adjustment to be replicated on the outstanding views it must be applied manually.

2.2.1 Grounds for migrating from a low-end to mid-range system

By utilising solid modelling software, designs can be produced faster and to a higher standard. Once a design exists as a solid model, it is possible for all the relevant personnel involved with it to access the developing or approved design, at the same time. When it is possible to see a design on the screen as a physical solid model, potential design errors can be studied in greater detail and prevented, critical issues will become clearer sooner and resolved with less effort in the design-to-manufacture processes.

Migrating from a low-end system to a solid modelling package presents both advantages and disadvantages. The advantages include increased productivity and efficiency coupled with the ability to optimise designs at an early stage of the process.

Shepherd, R., 2003, showed in a survey that companies who migrated from a low-end system to a mid-range system showed an increase in productivity. Table 1 depicts the results from the research report. The table shows the productivity changes achieved by MCAD users by type of work and design method.

Classification of main vocational design activity within the sample by work type	Percentage of total sample surveyed (%)	Productivity change as a percentage between 30 months and 60 months after first use of an MCAD system		
		Upgrading 2D drafting (%)	2D drafting to 3D solid modelling (%)	2D drafting to 3D surface modelling (%)
All users	100.00%	+26.00%	+38.00%	+43.00%
General mechanical	25.00%	+26.00%	+40.00%	+42.00%
Other engineering	15.00%	+26.00%	+32.00%	+44.00%
Sheet metal & fabrication	15.00%	+27.00%	+37.50%	+45.00%
Process & Plant	10.00%	+24.00%	+44.00%	+42.00%
Mould & die	8.50%	+29.00%	+57.00%	+44.00%
Tooling & press work	10.00%	+26.50%	+62.00%	+42.00%
Drive train transmission	5.00%	+17.50%	+43.00% *	*
Electrical/Electronic components	9.00%	+26.00%	+40.00% *	*
Engine & power unit	2.50%	+20.00%	+50.00% *	*

* indicates that sample is too small for accurate productivity analysis

Table 1: Productivity change results (Shepherd, R., 2003)

Gott, B., 2003, reports in a white paper titled "Evolution from 2D to 3D, A Design Engineer's Perspective" that the primary advance of modern solid modelling systems over its 2D counterparts was increased productivity and effectiveness. The white paper also explains that 3D CAD provides the tools to create more competitive products, reduce time to market, increase customer satisfaction and enhance sales. The findings of the white paper are based on a survey carried out on a series of engineering companies after they made the transition from 2D CAD software to 3D CAD software. The white paper further explains how, with a 3D CAD system, an engineer can produce a geometrically accurate, digital product model, or 'virtual prototype' that is utilised throughout product development, manufacturing and assembly. Interacting with the 3D

model on screen increases the rate of design development, encourages innovation and instantaneously identifies errors. The white paper concludes that

- with modern 3D solid modelling CAD systems engineers spend more time designing and less time drawing as would be the case with a 2D CAD system
- 3D technology provides a more productive design environment that lends itself to increased creativity and innovation from the designer, increased quality of output and a reduction in errors
- hybrid 2D/3D technology is an important development as it allows flexibility in mode of working and step-by-step transition from 2D drawing to 3D solid modelling as the primary design method

In a separate white paper by Gott, B., 2003, titled "Evolution from 2D to 3D, a senior manager's perspective" the key enabling factors for investing in 3D CAD technology are identified as follows.

- Reduced costs of computing power, graphics processing and data storage
- Emergence of Microsoft windows as a standardised operating system for 3D CAD fully capable of supporting collaborative engineering in the extended enterprise;
- More utilisation of Microsoft standard data handling applications and utilities for lower costs and increased compatibility with other business processes;
- Ability to model large assemblies in 3D and manage all the complex relationships between features, components and assemblies;
- Engineering data management integrated within the 3D cad system so that product data and administration is handled 'on the fly';
- Tools that provide access to the product model and design data by external applications;
- Easier to use systems resulting from improved logical structure and user interface style;
- Full integration of 2D drawing and 3D modelling within the same cad product - known as 'hybrid 2D/3D' technology.
- Mixed 2D/3D semi-automatic workflow aids for assembly layouts and modelling;

Hess, C., 2003 ([2]www), states that the return on investment for 3D CAD technology is much more tangible than it was 15 years ago due to the fact that greater functionality is available for a smaller cost, more companies are deploying 3D CAD and realising more substantial return on investments.

A survey conducted by Mirman, I., June 2003 ([3]www), of more than 1,000 SolidWorks users confirmed that there is greater return on investment impact of current 3D systems. The survey showed that of the companies migrating to SolidWorks:

- 95 per cent reported an increase in productivity
- 69 per cent reported faster time to market
- 90 per cent reported one or more of the following:
    - Faster time to market
    - Reduced volume of ECOs
    - Reduced time spend on the average ECO
    - Reduced scrap from design errors
    - Reduced waste material from CAM integration

According to the survey, the results were consistent across different industries and previous CAD systems used.

2.2.2 The advantages of implementing 3D CAD in the workplace

Gott, B., 2003, suggests that there are a number of advantages to be achieved by implementing a solid modelling system, the fact that so many product development companies continue to work with 2D systems suggests either that the advantages are not fully understood or not accepted.

- Animations - Seeing projects in action without cutting any metal. 3D CAD can allow the engineer simulate the manufacturing process on screen and preview potential errors before they ever happen.
- Automatic bills of materials - Reducing possible errors. Many CAD systems have integrated database technology as a part of their system. As the model is created the database is filled with various types and amounts of information

associated with the model. This information is then extracted automatically on demand to produce manufacturing data such as bills of materials.

- CNC links – It is possible to use the 3D CAD package to generate the required CNC programme to export to a mill or lathe. Time spent and cost spent writing CNC code is reduced.
- Collaboration - The digital product model, allied to the Internet, enables the practice of concurrent and collaborative engineering, providing the basis for design review with customers, planners, manufacturers and subcontractors, and directly feeds engineering analysis and manufacturing software.
- Engineers stay as engineers - 3D CAD software is often described as a design tool and not a drafting package. It enables the engineer to design a component and or assembly on the computer instead of drafting a series of 2D elements to resemble it like with some common 2D CAD software.
- Finite Element Analysis (FEA) - Allows simulation and analysis of real life loads, stresses and strains. FEA software will calculate how much of a load can be placed onto a component. It will tell the engineer where the part or assembly is most vulnerable to failure and under what conditions it will occur.
- Interpretation - If a solid model fits together on a computer screen it can therefore be built. Sets of 2D drawings are subject to misinterpretation, leading to errors and late ECOs thus time and money can be lost. Interacting with the 3D model on screen speeds up design thinking, it encourages innovation and identifies errors as soon as they occur.
- Isometrics – The design can be communicated from the designer to others with out misrepresentation. From the model on the screen, 3D CAD systems automatically generate isometric and perspective views. These views are used to accurately represent the engineer's design to people not trained in visualising 2D views such as elevations, plans and end views.
- Parametric technology - Updating one component automatically updates all associated parts. Parametric technology makes editing seamless and it streamlines the design process
- Product development, manufacturing and assembly - Using a 3D CAD system, the engineer produces a geometrically accurate, unambiguous, digital product

model (or 'virtual prototype') that is utilised throughout product development, manufacturing and assembly.

- Reduced time to market. New products can reach the market place in shorter periods of time.
- Sheet metal development – Solid modelling sheet metal software can be used to eliminate calculations for fabrication.
- Software Customisation. By customising software using a Graphical User Interface it is possible to automate the process of manual repetitive tasks as well as reducing the amount of user input and understanding necessary, thus, dramatically decreasing the amount of time taken on each project.
- Visualisations - 3D CAD packages also have built in rendering engines so that presentation quality images can be created on demand.

2.2.3 The disadvantages of implementing 3D CAD in the workplace

Although there are advantages and incentives to implement a 3D system into the workplace as seen in section 2.2.2, there are also disadvantages that may be experienced during the process.

- File sizes – Parasolid and ACIS based CAD systems generally associated with creating large files from their software packages. This can lead to large sections of a computers hard disk being taken up by CAD data also resulting poor performance by the computer. CAD offices storing files on a server may easily experience a sluggish performance while opening and closing files over the network as the software packages can take up large percentages of the available bandwidth.
- High software maintenance - 3D software is a fast developing industry, it is not uncommon for systems to be updated on a three monthly basis. This can lead to large maintenance agreements for companies to keep CAD software fully up to date.
- Problems with compatibility - Many major software systems create standard files that are illegible by other systems. There are many 3D modelling CAD systems available using a variety of solid modelling kernels making it almost

impossible for all products to be compatible with each other. Companies working with different systems that wish to communicate with each other via their 3D models may incur difficulties unless specially written translators are available because of the contrast in file types between those systems.

- Recruitment and training - during the initial implementation staff will have to be trained at a cost, or recruited to meet the company's needs for the software.
- Resistance to change – Companies who implement a 3D CAD system after using 2D often find resistance from employees who have been using 2D for a long time. This can often be a source of conflict for managers where software vendors and the companies CAD operators can vary over which system is best, the older 2D system or the newer 3D one.
- Temporary productivity loss – productivity levels will decrease for a time while a 3D system is being fully integrated into the company's day to day activities. This includes the period that staff needs to adopt the most efficient methods of using the system.

2.3 Solid modelling kernels: Parasolid and ACIS

Before selecting the solid modelling system for use in the project, an investigation into solid modelling kernels was undertaken.

Unigraphics Solutions Inc., 2000, define the Parasolid solid modelling kernel as an exact boundary-representation geometric modeller supporting solid modelling, comprehensive cellular modelling and incorporated freeform surface/sheet modelling. Spatial Technology, Inc., 2000, defines the ACIS kernel as a solid modeller that integrates wireframe, surface, and solid modelling by allowing these alternative model representations to coexist naturally in a unified data structure, which is implemented as a hierarchy of C++ classes.

Limitations in CAD software functionality can generally be traced back to the kernel and whether it supports that particular functionality or not. An application developed with a powerful kernel will have the potential to be a powerful application and it will benefit from the kernels technology.

General characteristics that would define the power of a kernel would include:

- Robustness & Reliability - The ability to process long and difficult calculations without crashing the computer system.
- Functionality – The ability to support the necessary tools and commands essential to CAD applications.
- Intelligence – The ability to demonstrate an understanding for the manufacturing process e.g. by having the capacity to recognise a feature or part that cannot be manufactured.
- Speed – The ability to perform calculations and generate solid models promptly with a limited portion of the computers resources.

The results of the investigation showed that the industry standard and most popular kernel in the market today is:

- PARASOLID developed by UGS PLM Solutions

Others include:

- ACIS developed by Spatial
- ShapeManager developed by AutoDesk

2.3.1 Parasolid

As described in section 2.3, paragraph 2, Parasolid (Unigraphics Solutions Inc., 1999) is as an exact boundary-representation geometric modeller supporting solid modelling, generalised cellular modelling and integrated freeform surface/sheet modelling. Parasolid was originally designed for high-end MCAD applications, e.g. complex surface meshing, but is now also used in a wide range of diverse mid-range systems such as Solid Edge, IronCAD and SolidWorks.

Parasolid generates parts from simplistic features such as blocks, spheres and cylinders. The profile of these features are first sketched and applied with constraints in a 2D environment. Parasolid also supports the construction of complicated surface geometries such as the shapes found on modern road vehicles such as cars and motor cycles.

Parasolid supports the design and construction of products using pure digital data and digital processes. It is supported on the Microsoft Windows and UNIX operating systems.

2.3.1.1 The Parasolid model structure

Unigraphics Solutions Inc., 1999, describes the Parasolid model structure into three classes; their general relationships are shown in Fig. 1.

- Topological,
- Geometric, and
- Associated data

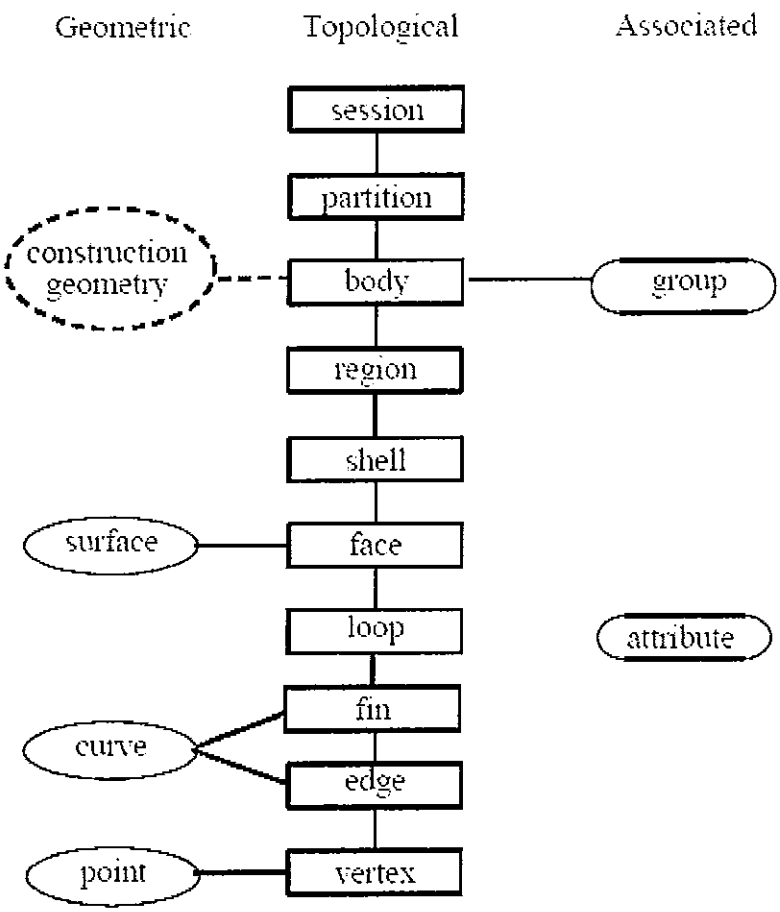


Fig. 1: The Entities of the Kernel Model (Unigraphics Solutions Inc., 1999)

2.3.1.2 Topological entities

This class contains all the entities, which constitute the structure or skeleton of the model. As Fig. 1 shows, there are ten types of topological entity.

2.3.1.3 Geometric entities

There are four types of geometric entity as described in Fig. 1 In a completely defined part, the geometric relationships within a body or an assembly must be specified by geometric entities, which are attached to topological entities. Fig. 1 shows them attached to the structure according to this function only. But they also have other uses, and can be manipulated independently of any body or assembly.

2.3.1.4 Associated data entities

Associated data entities allow additional data to be manipulated and attached to the model, or extra structure to be defined. They are described in Fig. 1

Groups are defined as a collection of entities. They can be created by the user and attached to bodies. Groups can refer to faces, edges, vertices, surfaces, curves and points, or a mixture of these entities.

Attributes are data structures and are used to attach information to entities. They contain fields of particular types which are filled in when an attribute is attached to an entity.

2.3.1.5 De facto standard for CAM

In an article published by CIMdata, 2003 ([4]www), it was claimed that Parasolid had expanded its leading position in the global computer-aided manufacturing (CAM) software industry. The survey was based on the leading CAM software vendors listed in the May 2003 version of the NC Software and Related Services Market Assessment and yielded the following statistics:

- Eight of the top ten vendors (based on 2002 install base) are Parasolid licensees and seven of the top ten uses Parasolid exclusively for solid modelling.

- Parasolid is selected for modelling capabilities by a margin of 6:1 over its closest rival by the top 25 CAM vendors (based on 2002 install base.)
- The number one vendors in three different categories - 2002 revenue (UGS PLM Solutions), 2002 install base (CNC Software) and 2003-projected growth (Vero International) - all use Parasolid as their exclusive geometric modelling kernel. Parasolid is the modelling foundation for some of the best-known brand names in the CAM software industry including Unigraphics NX (UGS PLM Solutions), Mastercam (CNC Software), GibbsCAM (Gibbs and Associates), EdgeCAM (Pathtrace), SURFCAM (Surfware), VISI-Series (Vero International) and ESPRIT (DP Technology).

2.3.2 ACIS 3D modeller

As described in section 2.3, paragraph 2, ACIS (Spatial Technology, Inc., 2000) is an object-oriented 3D geometric modelling engine. ACIS is written in C++ and consists of a set of C++ classes (including data members and methods) and functions. It is possible to use the C++ classes and functions to create an end user 3D application. ACIS complements existing applications by offering a unified environment for the modelling curves, surfaces, and solids. ACIS also supports the integration of proprietary curve and surface subsystems.

2.3.2.1 ACIS architectural overview

The ACIS product line is designed using software component technology. Software components are collections of related pieces of software. The components are developed globally by Spatial and its partners. A product is one or more components that Spatial packages and sells as a separate item.

A product corresponds to how the software is packaged and sold, but a component corresponds to how the software's architecture is designed and how the software maps to an installed top-level file system directory. While components could be packaged differently to form new products without affecting the system architecture, changes in components would change that architecture.

2.3.2.2 Modelling overview

A wireframe model defines an object only by its edges and vertices. A surface model is similar to a wireframe model, but it defines an object by its visible surfaces, including its faces. A solid model defines an object in terms of its size, shape, density and physical properties (weight, volume, centre of gravity, etc.). ACIS is a solid modeller, but wireframe and surface models may also be represented in ACIS.

ACIS separately represents the geometry and the topology of objects. This concept is called boundary representation, or B-rep modelling. This provides the ability to

determine whether a position is inside, outside, or on the boundary of a volume, and distinguishes a solid modeller from surface or wireframe modellers.

The ACIS model representation consists of various geometric and topologic entities, as well as attributes that may be attached to the entities. The model is implemented in C++ using a hierarchy of classes.

2.3.2.3 Tolerant modelling

In order to allow ACIS based applications exchange data files, efficiently and accurately, from applications that use other modelling kernels ACIS' architecture has been developed to be capable of accepting new data formats (as well as its native industry standard SAT) or translators for a peak level of interoperability. Spatial has improved ACIS' tolerance for processing data in the following 3D modelling categories to ensure that the minimal amount of information is lost while transferring data into or out of its platform.

- Stitching – Enables surface models to be stitched into tolerant bodies without regard to the tolerance of the original models.
- Surface Construction and Editing-includes blending, local operations, skinning and lofting.

2.4 Kernel test: Solid Edge (Parasolid) vs. Inventor (ACIS)

In 1997 Solid Edge switched from the ACIS modelling kernel to Parasolid. Tests carried out by Andriulli, M., 2002, against Parasolid based SolidWorks revealed that Parasolid out-performed ACIS in three primary functions thus justifying the choice to cease using ACIS for Solid Edge and progress forward with Parasolid. In the tests, the three primary functions where Parasolid out-performed ACIS were:

- Round/edge blending
- Thin-wall
- Add draft

In 1997, ACIS 3.0, either under performed on equivalent functions or required additional code to implement functions fully integrated in Parasolid. Following the release of ACIS 7.0, AutoDesk implemented an in-house, modelling kernel, Shape Manager, in order to improve performance and also to reduce revision lag.

The following is based on the test report from Andriulli, M., 2002, where a series of tests were performed on Solid Edge and AutoDesk Inventor 5.3 in an attempt to verify if ShapeManager (kernel used in AutoDesk Inventor 5.3) presented an improvement over ACIS 3.0 as tested in 1997 when Solid Edge ceased using ACIS for Parasolid.

2.4.1 Add draft

Adding draft for ACIS involved offsetting existing surfaces and recalculating the intersections and extensions of the new surfaces. These recalculations can consume significant time if they are handled inefficiently by the kernel. ACIS 3.0 did not handle offset and add draft calculations and required additional code to create extensions. ACIS 7.0, on which Shape Manager is based claims to support add draft and hollowing capability. The performance lag and lack of robustness, however, exhibited by Shape Manager during the tests indicated that those functions are added onto the kernel or they are inefficiently handled. Parasolid handled add draft and thin wall as internal function calls. Parasolid performed these functions more robustly and rapidly than any third-party extension to or recent update of Shape Manager.

Inventor 5.3 added draft angle to features differently than Solid Edge. Solid Edge permitted selection of all normal features to be drafted relative to the reference plane and treated the entire component as a single body. Inventor either added draft at the creation of the feature relative to the base plane, or added draft as a separate feature. Adding draft angle to a component as a separate feature requires separate selection of non-continuous normal surfaces. Inventor would not treat the component as a continuous entity. This resulted in Inventor requiring two add draft steps to create the same feature that Solid Edge requires in one feature step.

2.4.2 Edge blending

This test exercised the edge based blending capabilities of the solid modelling kernels. Speed and success rate were given for both competing applications for five different case combinations of surface topology.

Solid Edge Version 11 proved to be both faster and more reliable than Inventor 5 for each of the tests, all combinations included. Solid Edge was able to generate the required features two to four times more quickly than Inventor in all cases. Solid Edge also required fewer steps to place rounds over complex edge combinations because it was capable of handling multiple round placements without failing. With one exception, Solid Edge generated all rounds within the same edge selection step.

2.4.3 Thin wall

ACIS 3.0 did not support thin wall functionality when tested against Parasolid in 1997. Additional implementation to achieve this functionality resulted in significantly slower performance compared to Parasolid. Adding the thin wall for ACIS function involved offsetting existing surfaces and recalculating the intersections and extensions of the new surfaces. These recalculations consumed significant time if handled inefficiently by the kernel or implemented on top of the kernel. ACIS 3.0 did not handle offset and add draft calculations and required additional code to create extensions. ACIS 7.0, on which Shape Manager is based, claimed to handle add draft and hollowing capability. The performance lag and lack of robustness, however, exhibited by Shape Manager during testing indicated that those functions were added onto the kernel or are inefficiently handled. Parasolid handled add draft and thin wall as internal function calls. Parasolid had many years to refine these function calls. Parasolid performed these functions more robustly and rapidly than any third party extension to or recent update of Shape Manager.

2.4.4 Report conclusions

The report claims that justification for switching Solid Edge from ACIS to Parasolid was valid. It was clear that Parasolid based Solid Edge both out performed and succeeded at a higher rate than Inventor's ACIS based Shape Manager. Most notable

was the ability of Solid Edge to add draft to an entire body of separately constructed features. Functionally, Inventor would add draft at the creation of an extrusion, but doing so increased the complexity of containing or revising the draft angle. Inventor also required a fully constrained base profile in order to add draft, while not requiring full constraint to extrude the feature.

Solid Edge thin-walled significantly faster than Inventor. The Shape Manager test results are approximately proportional to the 1997 Parasolid and ACIS comparison results. Solid Edge placed draft views of complex geometry significantly quicker than Inventor. The test results showed that Solid Edge is seven to fourteen times faster than Inventor 5.3.

2.5 Customising solid modelling software

Many CAD users can appreciate the potential productivity gains to be had with the use of fully parametric of solid modelling software, especially after just migrating from 2D CAD. However many users do not realise that the initial gains in productivity can be further amplified by automating repetitive tasks using simple automation techniques with readily available tools such as Microsoft Excel, Access, Visual Basic and Visual C++.

Mid range solid modelling software such as Solid Edge and SolidWorks can be programmed or automated in various ways. The most effective of these being the technique of writing customised routines to control the solid modelling software. Any manual process that is used to perform a task can be programmed i.e. from modifying a dimension, inserting a part into an assembly or automatically creating a draft document.

Many, however, fail to realise the full potential that its software may have by neglecting the customisable tools it may have to offer.

There are many advantages to successfully automating solid modelling software. If implemented correctly it is possible that automation could encourage better design practices and further reduce design lead times and errors. Automation can also be an invaluable for reducing the level of skill needed to produce designs. By applying

strategic programming techniques, design rules and calculations can be applied to all commands, thus, removing the level of complexity and time necessary to complete tasks.

Solid Edge and SolidWorks can be programmed with different languages compatible with the Microsoft Windows operating system e.g. Visual Basic or Visual C++. It is possible to integrate Visual Basic or Visual C++ with a large percentage of desktop applications available on the Microsoft Windows platform which make it possible to integrate the solid modelling software with programs such as Microsoft Excel or Microsoft Access.

2.5.1 Writing code for SolidWorks

Lacey, R., 2000 ([5]www), describes three core methods of automating SolidWorks with Visual Basic.

- The Macro Recorder in SolidWorks
- The Microsoft Visual Basic for Applications or 'VBa' built into Microsoft Office applications
- Use Microsoft Visual Basic to write stand-alone programs that can control SolidWorks or any other Windows application

The first method of automating SolidWorks is to use its Macro Recorder. The Macro Recorder included with SolidWorks allows the recording of a series of SolidWorks commands; these are saved as a macro file and played back when desired. As these commands are written in Visual Basic, the macro recorder can be used to write large portions of the program code as opposed to being written manually.

The second method of using Visual Basic with SolidWorks is to use the Visual Basic for Applications or 'VBa' built into Microsoft Office applications. Using VBa, programs can be written in Microsoft Excel or Microsoft Access and then integrated with SolidWorks. In Microsoft Access the input dimension values are keyed into a database. SolidWorks can then use the Microsoft Access database to drive a solid model.

The most powerful method of automating SolidWorks is the third method. The third method involves using Visual Basic to write a stand-alone application to integrate with SolidWorks and automate repetitive tasks and perform calculations. A licensed version of Visual Basic is necessary to create the stand-alone application; however, it is possible to distribute the stand-alone application to other users without the need for them to have the development software on the machine on which they are running SolidWorks.

2.5.2 Writing code for Solid Edge

According to Unigraphics Solutions Inc., 2001, in order to customise Solid Edge it is necessary to have access to a Windows programming tool to be used as an automation client. An automation client is an application that can access objects from other applications that support client-server automation. There are multiple applications that can be used to create an automation client, such as Visual Basic and Visual C++. Any product that supports client-server automation that follows the Microsoft COM automation model can be used to automate Solid Edge.

The Solid Edge programming utilities allow the customisation or automation of Solid Edge with client-server automation. With these tools, it is possible to modify and enhance standard commands to tailor Solid Edge to meet specific needs. It is possible also create tools that allow the reduction or automation of repetitive tasks. Using standard Windows programming tools and languages it is possible to create command extensions that precisely match specific.

UGS PLM Solutions Inc., 2002 ([6]www), states that Solid Edge provides a complete and fully documented application programming interface (API) that adheres to standard Windows OLE Automation and Component Object Model (COM) technology. With the API, it is possible to customise Solid Edge with Visual Basic, Visual C++ or other standard programming languages.

As with SolidWorks, Solid Edge also supports the use of Microsoft Excel, Visual Basic and Visual C++. Similarly to SolidWorks, using a bespoke application to automate Solid Edge provides more potential for productivity and efficiency.

Solid Edge has an open architecture allowing access to all objects with in it for customisation. This provides the ability to create custom software that may access Solid Edge and all of its commands with no boundaries. As Solid Edge and Visual Basic both support ActiveX technology it is possible to create software with the functionality to perform all commands that may be performed manually.

2.5.3 The advantage of customising solid modelling software

In a case study described by CAD/CAM Publishing, Inc., 2003 ([7]www), RBS (a manufacturer of custom baking equipment) considerably reduced labour costs and errors in manufacturing by customising their existing manual solid modelling design software.

Each project undertaken by the company is customer specific, although some sub-systems from previous projects are reused in new projects. Completed projects may contain as many as 2000 parts. The software application used by the company was written in Visual Basic and is designed to extract part lists from assembly models stored in Solid Edge Insight and feeds them to a Manufacturing Resource Planning (MRP) system. The custom software application eradicates the need for bills of materials to be keyed manually into the MRP system thus eliminating the possibility of error by automating the entire process.

The case study outlines the level of speed and accuracy that can be achieved by automating manual processes. By successfully implementing customised software it is possible to increase productivity and efficiency, increasing the amount of work possible to complete in the same amount of time.

By successfully developing software to automate 3D CAD software it is possible to implement a graphical user interface to replace command menus. It is also possible for multiple commands to run consecutively, decreasing the time taken to manually perform them. The custom software application can perform calculations as it executes routines and apply the results in the subsequent stages of operation.

Another advantage attributed to this style of system is that new personnel can be recruited with less difficulty, than before, as the level of skill necessary to perform daily activities has been reduced. Also, the time necessary to train new and existing staff into

daily operations will be less than what was previously feasible prior to the customisation of the solid modelling software.

It is the author's intention to validate the advantages described above by developing a customised solid modelling design system specifically for an Irish manufacturing company.

2.6 CAD system selection

After exploring existing solid modelling technology, the solid modelling software to be used in this project was selected. After appraising different systems that use the kernels and the object technologies mentioned previously it was decided that a Parasolid based CAD system was the best approach. Solid Edge from UGS PLM Solutions, whom also own and develop Parasolid, would be used.

Solid Edge was chosen on the basis that the software has an open architecture structure that provides the necessary ActiveX tools for programming an automated modelling system and suits the needs of this project.

Characteristics that were sought before system selection were as follows

- User interface
- 3D functionality – be fully parametric
- Manage complex and large assemblies
- Produce high quality manufacturing data output
- Provide productivity tools to allow customisation
- Advanced sheet metal capabilities
- Previous exposure to the product at undergraduate level

A feature that was felt could be an advantage in the future would be integration with a Product Data Management (PDM) system. PDM is an essential requirement for any manufacturing company hoping to move from 2D to a fully integrated solid modelling system. The PDM system associated with Solid Edge, Insight, provides ([8]www) design management capabilities and unlike conventional PDM systems, Insight is not a separate software package as it is embedded directly into the CAD application.

Research by Cohn, D., 2003, into Insight technology concluded that companies could reduce ECOs and related rework by 50 per cent or more. In the past only large manufacturing corporations availed of the advantages of PDM systems due to the cost and technical issues associated with their implementation.

The fact that Solid Edge is created from Parasolid was a major factor for choosing it over other CAD packages.

Another key factor in the choice of using Solid Edge was the previous knowledge and experience the author had using the software at undergraduate level. From previous experience the author had confidence and faith that the software would be able to accomplish the necessary tasks put before it in this research.

2.7 Solid Edge

EDS, 2002, describe Solid Edge as a CAD system for mechanical assembly, part modelling, and drawing production. It is developed with STREAM technology and is designed to increase software productivity and efficiency. In Solid Edge's product overview UGS PLM Solutions state that the CAD software aims to help customers get products to market faster by providing the tools for designing products correctly the first time. It contains tools for eliminating design errors, thus reducing product development time and costs.

Solid Edge can be used to create graphical 3D prototypes of products on computer, for detailed analysis of design and also aides the design process of products by reducing design errors.

From the Solid Edge user guide, Unigraphics Solutions Inc., 2001, Solid Edge can be described as having has five main environments. The five environments are:

- The Part environment - Part modelling
- The Sheet metal environment - Part modelling
- The Assembly environment - Mechanical assembly
- The Weldment environment - Part modelling
- The Draft environment - Drawing production

Individual components designed in Solid Edge are represented with their own file. Assemblies built in Solid Edge are represented with a separate file also, but they are generated from a collection of parts and sheet metal objects. The collection of parts is represented on the computer hard disk as a series of files all located in the same folder.

One of the characteristics important to this research is its ability to make parts associative to each other between all environments. In the design of large complicated assemblies containing many parts and sheet metal objects, a change to one part can have an impact on the whole assembly. The ability of software to make parts associative to each other was important. A small change to an individual part can have global implications as far as the top level assembly is concerned. When this type of situation occurs during the design stage the ability of the software to modify the part, and then to carry out the necessary chain of edits up the assembly hierarchy to the top level is vital.

During the design process for this project, access to this technology was an advantage while aiming to design such a manufacturing system for precision air-handling units. Having to manually review every file that may be affected by a change to one part would make design revisions inefficient. Document management would be effected in the same manner. Throughout the course of this project the Solid Edge part, sheet metal, assembly and draft environments were used extensively.

2.8 ActiveX technology

Having researched all the appropriate solid modelling technologies and products available and after selecting the system to be used, it was necessary to research the technologies involved in customising solid modelling technology.

Microsoft Corporation, 1999 ([9]www), describes ActiveX as a set of technologies that enable software components to interact with one another in a networked environment, regardless of the language in which the components were created. An ActiveX control is a user interface element created using ActiveX technology. ActiveX controls are small, fast, powerful, and make it easy to integrate and reuse software components.

The GUIs can be used to act as fronts, or as user friendly screens for complicated command extensions. The GUIs are designed in logical order for the user so that information can be inputted in logical sequence to streamline automated advanced assembly design and construction.

Unigraphics Solutions Inc., 2001, states ActiveX is the term used to describe the Microsoft Component Object technology (COM). This is the same technology frequently referred to as OLE Automation. The ability to customise Solid Edge is based on an important part of this object technology—ActiveX Automation. ActiveX Data Objects (ADO) is a Data Access paradigm, which enables the management of databases; it also takes advantage of ActiveX technology as the name suggests – ActiveX Data Objects.

2.8.1 Objects, properties and methods

ActiveX automation is based on three characteristics

- Objects
- Properties
- Methods

Properties are the characteristics that define an object e.g. the length, height and width are certain Properties of a cuboid. Here, the cuboid is the Object. Other properties would include materials and colour.

Methods can be defined as the operations that can be performed on the object. For example, a method that may be performed on the cuboid may include, drag, copy, rotate or align.

The concepts of properties and methods used in Solid Edge to define objects are common to Visual Basic programming. Visual Basic programming also uses some object-oriented programming techniques. An example of an object in Visual Basic is the text box; a text box can be used to enter specific data (numeric or alphabetical). When a text box is placed on a dialog box, a set of properties is provided to control the various aspects of it including its appearance and behaviour. Some of these properties are

caption, font, height, width, name, and tag. The text box also supports methods such as drag and move.

With regard to automating a Solid Edge part or assembly drawing, Visual Basic can use its set of objects, properties and methods and apply them to Solid Edge's objects, properties and methods. For example, Visual Basic can use the value property of the text box mentioned above and apply it to a property of a Solid Edge object and change it. The Solid Edge property may be a property of any object.

Unigraphics Solutions Inc., 2001, states when programming Solid Edge, every part of the product, every document, every model, every part, and every feature of that part is treated as an object and has its own set of properties and methods. Using these objects and their properties and methods from Visual Basic, it is possible to interact with Solid Edge to create, edit, and query the objects. These objects can be programmed using Visual Basic because Solid Edge has exposed them, that is, made them available, to automation.

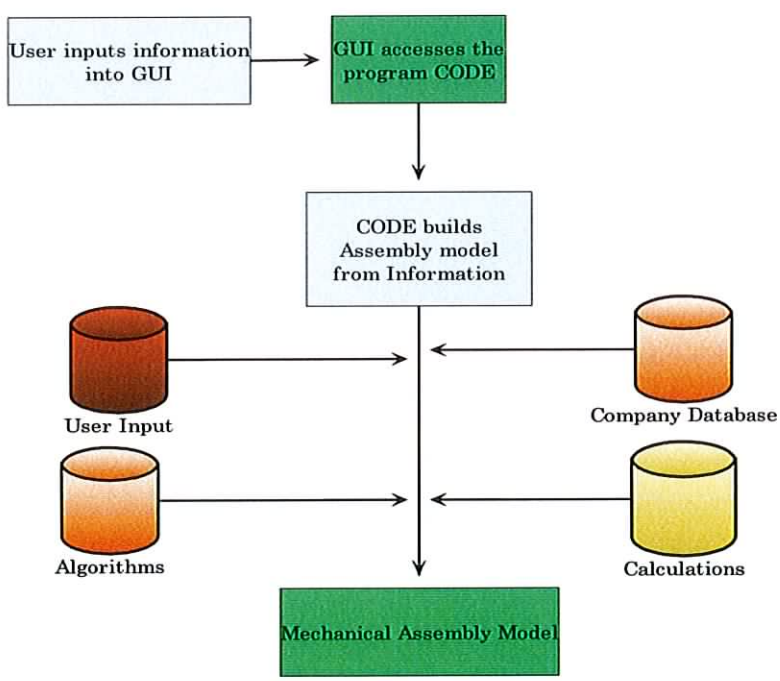


Fig. 2: Visual Basic code building an Assembly Model

ActiveX technology provides the GUIs with the ability to access the applications program code and generate the solid models. Fig. 2 above represents the process involved in generating assembly models using application extensions.

Conventional programming techniques within Visual Basic can be used to take advantage of standard mathematical equations and formulae so that they can be used for calculation purposes. These calculations performed within the code can be used to reduce the amount of user input required. The less user input required can reduce the amount of time it takes the user to build and assembly with the custom application extension.

ADO technology can also be used to reduce the amount of time it takes the user to build an assembly. ADO as mentioned earlier is a Data Access technology. It can be used to search for and retrieve specific information stored in their databases. Once the programming code can access the correct information with out user assistance it too can also reduce the amount of time spent constructing the assembly models. ADO achieves this by

- Reducing the amount of user input required
- Saving the user time spent manual searching through catalogues of suppliers specifications manuals and then keying in the information

2.8.2 ActiveX object hierarchy

Unigraphics Solutions Inc., 2001, states objects are related to one another in a form that is called an object model or object hierarchy. Customising Solid Edge requires an understanding of the various application hierarchies. In the Fig. 3 below, an object hierarchy is depicted as a cascading diagram with the parent object always being the application.

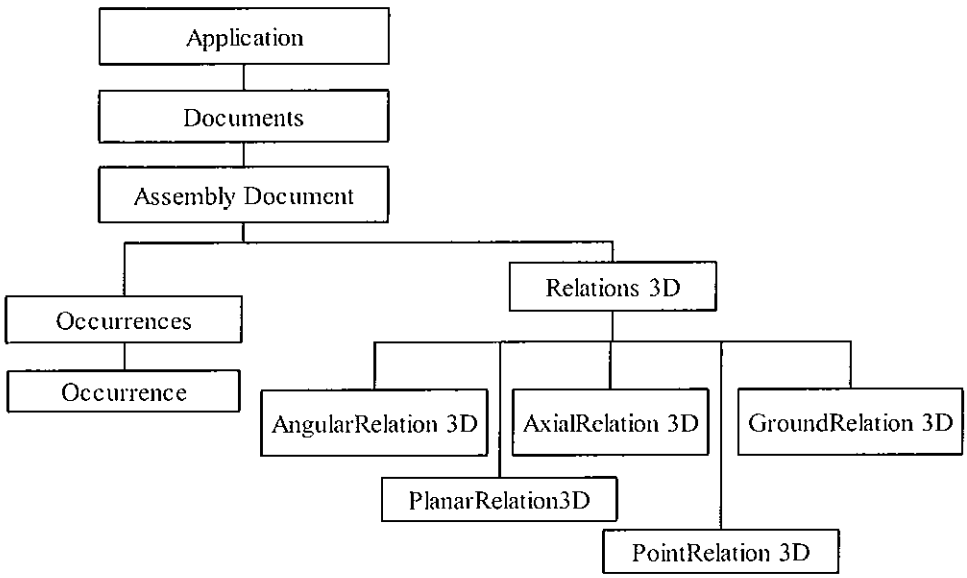


Fig. 3: A simplified object hierarchy of the Assembly Environment (Unigraphics Solutions Inc., 2001, Programmers guide)

ActiveX technology is vital to this project. A good deal of time and effort was spent learning its most effective uses. ActiveX technology provides links to databases as well as CAD Drawings. Both sets of links are present within the one application that then enables linking the database with the CAD Drawings.

ActiveX technology provides the tools to generate drawings from the information stored in databases. Using ActiveX technology stored database information can be automatically accessed and retrieved to contribute to the construction of 3D. Once the information is in the CAD systems it can used to generate the manufacturing information that is needed i.e.

- working drawings
- bills of materials
- cut lists
- costs

For this system to be fully functional when instigated it is essential to create accurate 3D models. The systems will not provide a good quality of information if a poor quality model is generated.

2.9 Conclusion

After researching CAD the object of this research was to show via the implementation of bespoke software using solid modelling and customisation techniques, better product design efficiency and productivity can be achieved. By means of a case study a design system for precision air handling units was created in partial. Chapter 6 describes the design system in detail. Solid Edge from EDS PLM Solutions, a Parasolid based solid modelling package, was chosen for the bespoke system. The bespoke system was created by integrating the solid modelling package with ActiveX and Data Access technology. The benefits of the new system were gauged by testing the difference in time taken to carry out specific tasks regarding the production of manufacturing drawings and documentation compared to current design systems. The final test results showed that over all the tasks performed that on average the bespoke system increased productivity by over 400 per cent. The test results can be seen in section 6.4.1.3, Table 5.

Chapter 3: Data Access Technology

To design an air-handling unit, many different parts, components and subassemblies must first be designed and specified in order for it to perform to its required capacity. Information and design criteria must be on hand from supplier's catalogues, customer's requirements, and in house regulations so that the air-handling unit can be built to specification.

Data Access technology provides programming utilities and tools so that programmers can create applications that can automate the process of accessing and retrieving data stored in locations such as databases, spreadsheets and text files. If it was possible to store all the information necessary, regarding product design, using Data Access it would be possible to automate the process of searching for and retrieving precise information. This would further reduce the time taken to design an air-handling unit. By providing access to the data source Data Access also supports customisation and editing of it.

There are many Data Access technologies available

- UDA: Universal Data Access
- ODBC: Open Database Connectivity
- OLE DB: Object Linking & Embedding for Databases
- ADO: ActiveX Data objects
- SQL: Structured Query Language
- DAO: Data Access objects
- RDO: Remote Data objects

3.1 Universal Data Access (UDA) technology

UDA is a technology that can be used to access information from one or more types of data source within the same computer application.

Amongst others, the following is a list of types of data sources

- traditional database applications

- workflow type applications make use of message stores in Microsoft Exchange or Lotus Notes
- a programme that manipulates the file system by looking at the directory and file structures
- a desktop application that uses one of the office object models to manipulate data in an Excel Spreadsheet or even Word document
- a communications program that identifies all users currently logged onto a network

The data sources mentioned in the above list need to represent their data in specific tabular format for their applications. For each type of data source mentioned above there is a unique interface for reading, writing and customising the data.

Williams, C., 1999, describes UDA as the solution from Microsoft to access a variety of information sources, as mentioned in the previous list of bullets. This includes relational and non-relational data sources. UDA is a simplistic programming interface and it is a tool and language independent. UDA can be described as a group of technologies that enable the integration of diverse data sources, relational and otherwise through a single client application.

Connell, J., 1998, states that the way to employ UDA is to use ADO. By using ADO, data can be accessed using ODBC or an OLE DB provider.

From Fig. 4 below, the relationship between the custom written client application and the files that it must access can be seen. For every file type that the client application wishes to gain access to it must go through a specific driver for that file type.

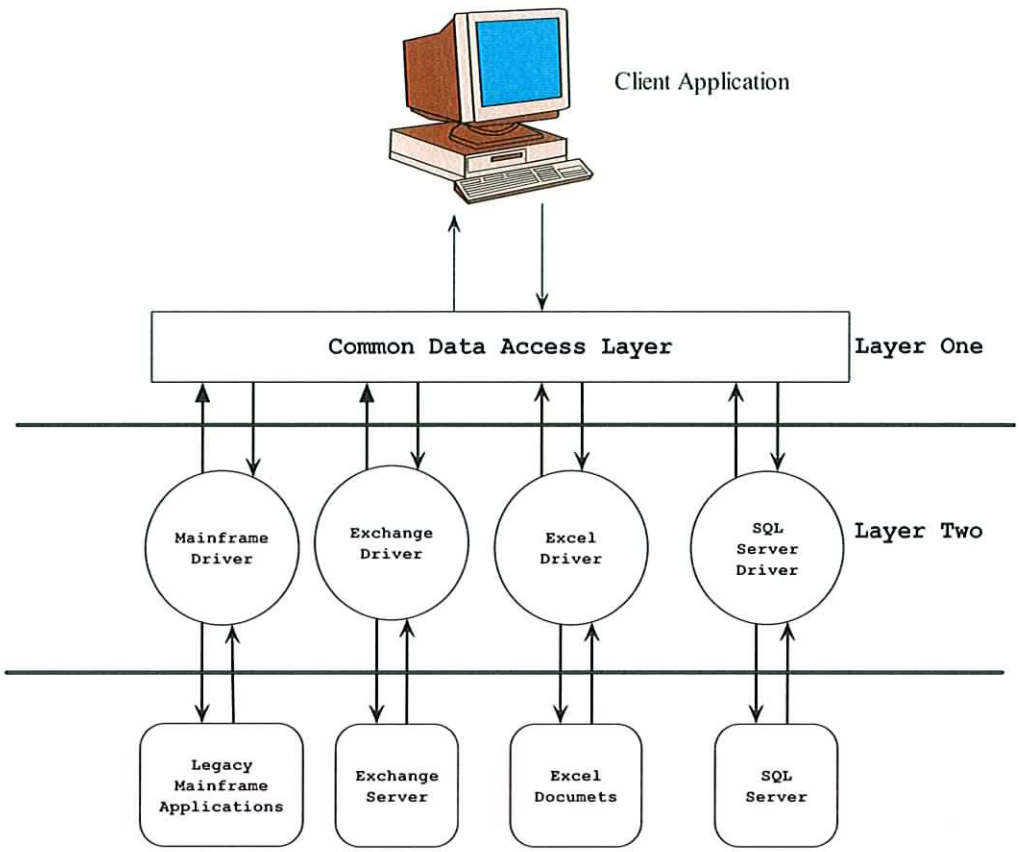


Fig. 4: Architecture of an application using a common Data Access Layer (Williams, C., 1999)

3.2 Object Linking & Embedding for Databases (OLE DB)

Smith, R. and Sussman, D., 2003, state OLE DB is a technology developed by Microsoft that provides access to both relational and non-relational data. OLE DB can be used to extract data from Access databases and from SQL Server databases.

OLE DB is a set of COM-based programming interfaces (Williams, C., 1999) that allow transparent Data Access, but in such a way that the client applications do not know or need to know what kind of data they are connected to.

Connell, J., 1998, states that OLE DB is designed to provide universal access to several relational and non-relational data sources. By using ADO in conjunction with OLE DB it is possible to communicate with Access, Oracle, SQL Server or any other data source by using the ADO object Model, see section 3.3.

The OLE DB architecture is composed of two entities, Providers and Consumers.

An OLE DB data provider is (Thomsen, C., 2002) another phrase for driver, meaning it is a binary library that exposes an API that can be called from within an application. ADO calls OLE DB data providers using Component Object Models (COM).

Roman, S., 2002, states that the purpose of an OLE DB data provider is to expose the data stores of a particular type in tabular format, with rows (records) and columns (fields). The role of the data provider is to make data from a data store appear like a table, even if the raw data format does not resemble a table. Thus, a data provider usually has direct access to the data in data stores of that type.

OLE DB data providers are COM components that are built upon a native data format. The format they are built on is the same format that data store uses. OLE DB providers' purpose is to break down the native data into a stream like structure so that information can be abstracted from it. Once the native data is broken down it can be accessed using standard interfaces. An example of a Provider is a data source i.e. supplies a stream of access to the "Rowset", see Fig. 5.

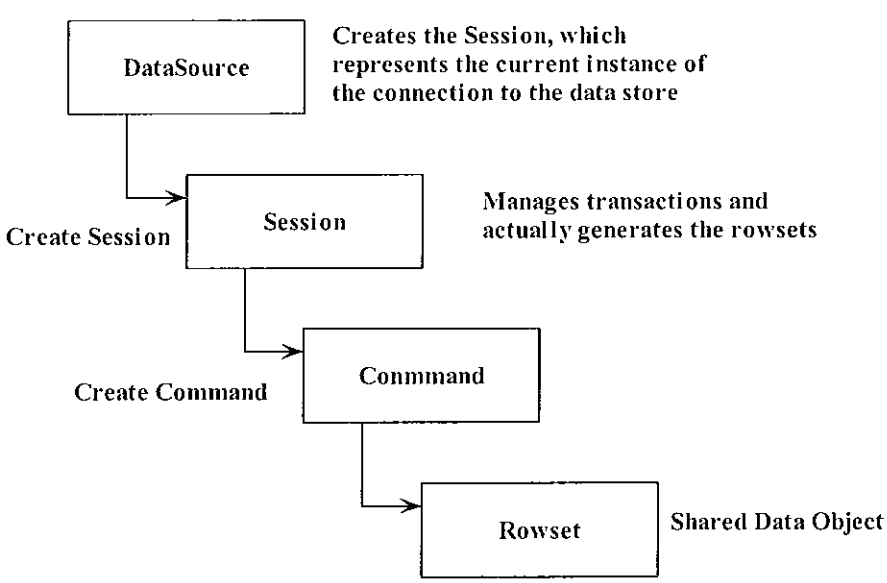


Fig. 5: Relation stream between the main OLE DB objects ([10]www)

Williams, C., 1999, states the Component object Model (COM) provides a standard mechanism by which objects can communicate regardless of what language is used to create the components. There are two major Windows technologies that rely on COM to

function. These are OLE and ActiveX. Both technologies use COM to facilitate the interaction between objects. OLE uses COM to communicate between applications, allowing users to link or embed parts of one application's data and display it in another. ActiveX has a wider brief. ActiveX uses COM to communicate between controls (i.e. ActiveX controls such as Microsoft's calendar control) and components.

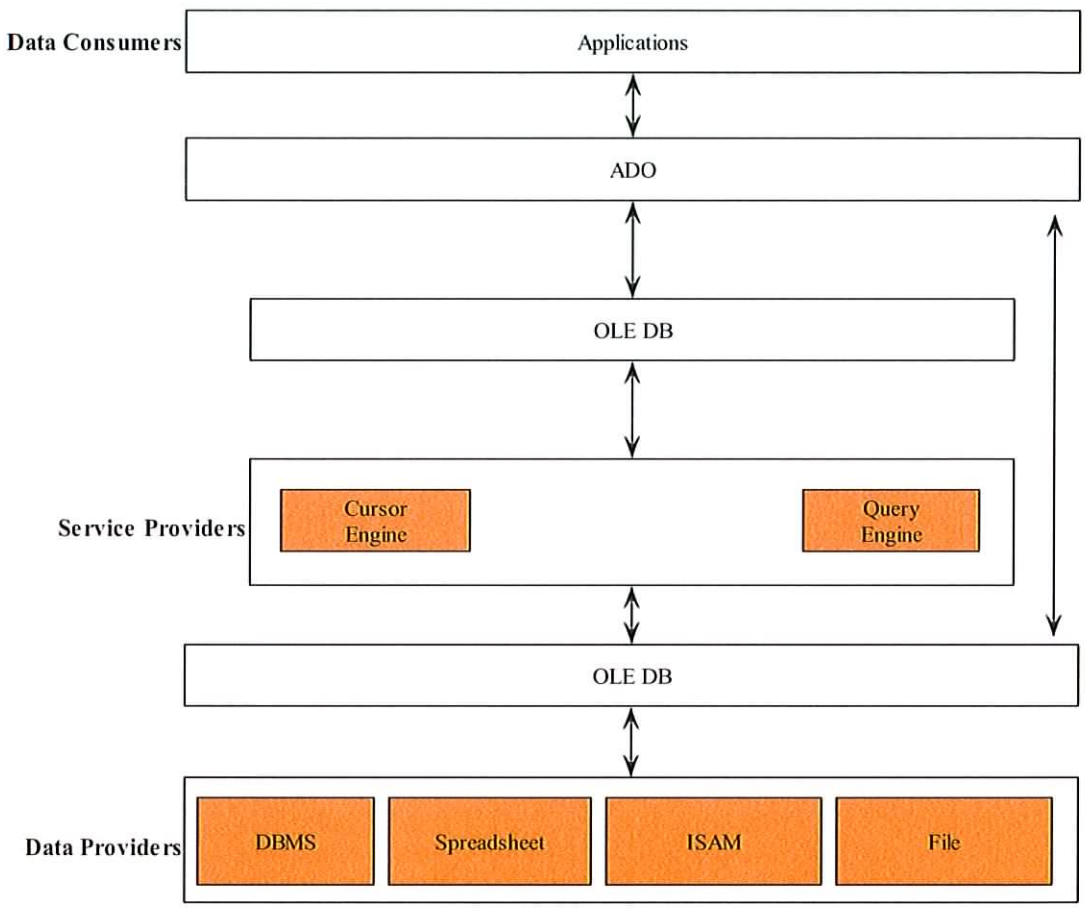


Fig. 6: OLE DB and ADO (Roman, S., 2002)

Consumers expose the information in the rowset by obtaining data in a tabular format from the rowset object.

Roman, S., 1999, states that an OLE DB data consumer is a software component that communicates with a data provider in order to gain access to and manipulate a data store. To a data consumer, all OLE DB data has a tabular format, with rows and columns

Consumers are components that provide or expose a common interface for accessing data. This interface is independent of the OLE DB provider. An interface may be thought of as a set of functions.

3.3 ActiveX Data Objects (ADO)

ADO is the programming model for the UDA interface OLE DB (Roman, S., 1999). ADO is a technology used for accessing and retrieving information from databases and is the data access technology of choice for this project. Roff, J.T., 2001, describes ADO as one out of a series of technologies collectively known as Microsoft Data Access Components (MDAC) that allow UDA to be implemented. Other MDAC technologies include, Open Database Connectivity (ODBC), OLE Database, and RDS. Williams, C., 1999, describes ADO as a code layer built over the specification of OLE DB. Architecturally, ADO is the application-level interface to OLE DB. See Fig. 7 for ADO's architecture.

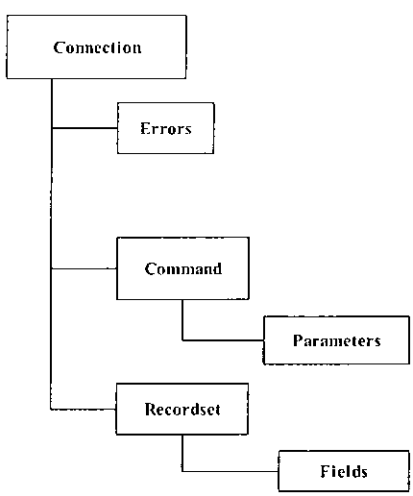


Fig. 7: The ADO Object Model ([11]www)

ADO is Microsoft's strategic, high-level interface to all kinds of data. ADO provides consistent, high-performance access to data, whether it is to a front-end database client or middle-tier business object using an application, tool, language, or even an Internet browser. ADO is the single data interface that needs to be known for 1 to n tier client/server and Web-based data-driven solution development ([12]www).

From Fig. 7 the Connection represents the link that exists between the software accessing the database. The Recordset and the Fields represent different levels of information stored in the database.

ADO is a cross-language technology for Data Access that exposes an object model incorporating data connection objects, data command objects, Recordset objects, and collections within these objects. The ADO object model provides a set of objects, properties, and methods for creating script, which accesses data in databases.

3.3.1 The ADO architecture

As previously stated, ADO is built upon layer of technologies (Roff, J.T., 2001) that allow applications communicate with data regardless of where it resides or how it is structured using any language or scripting language.

Microsoft, 2004 ([13]www) states that ADO is designed as an application level interface to Microsoft's Data Access paradigm, OLE DB. OLE DB provides access to any data source, including relational and non-relational databases, email and file systems, text and graphics, custom business objects etc. ADO is implemented for minimal network traffic in key Internet scenarios, and a minimal number of layers between the front-end and data source - all to provide a lightweight, high-performance interface. ADO is called using a familiar metaphor - the OLE Automation interface. And ADO uses conventions and features similar to DAO and RDO, with simplified semantics that make it easy to learn.

Roff, J.T., 2000, states that ADO consists of a generic style data access structure that allows access to any data source regardless of its structure with the same programming interface. The individual objects within the ADO object model are used to provide all the data storage, manipulation and retrieval commands needed when writing a data based application. ADO includes the following objects and collections.

There are three main objects to ADO, plus four subsidiary objects and four associated collections:

- The Connection object
- The Command object

- The Recordset object

Along with the Connection objects, Command objects and the Recordset objects, ADO makes use of four other subsidiary objects:

- Property
- Error
- Parameter
- Field

And with these four subsidiary objects there are four associated collections that are used to access the subsidiary objects.

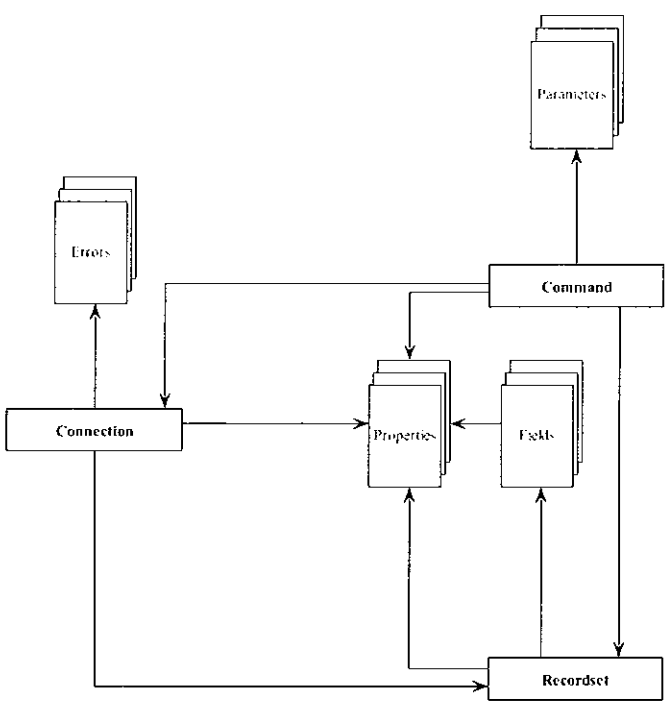


Fig. 8: How ADO Objects and Collections relate (Williams, C., 1999)

ADO commands are executed by the following sequence of actions:

- Connect to a data source. Optionally, it can be ensured that all changes to the data source occur either successfully or not at all.
- Specify a command to gain access to the data source, optionally with variable parameters, or optionally optimised for performance.

- Execute the command. If the command causes data to be returned in the form of rows in a table, store the rows in a cache that it can easily be examined, manipulated, or changed.
- If appropriate, update the data source with changes from the cache of rows.
- Provide a general means to detect errors (usually as a result of making a connection or executing a command).

3.3.1.1 The Connection object

The connection object is the portal for all data access through ADO. To access data from a source, a connection to it must be established. ADO uses the Connection object to achieve this. The standard information accepted by a Connection object includes filenames, data provider names, usernames and passwords. A Connection object is used is used to accomplish the following responsibilities:

- Select a data source and data provider
- Open and close a connection on a selected data source
- Manage transactions on a data source
- Execute queries on a data source

3.3.1.2 The Command object

The Command object is used to execute instructions. There are five types of commands that the Command object can execute.

- A SQL statement
- A parameterised query (a query with input and output parameters)
- A stored procedure from within the current data source
- A statement to open a single table
- A string command passed directly to the data provider

If the Command object is used to retrieve data, then a Recordset object containing the requisite data is created and returned to the application. The Command object can be associated with a currently open connection, or it can be created independently of any existing Connection objects.

3.3.1.3 The Recordset object

A Recordset object is used to access data on a record level. The Recordset object consists of a Fields collection of individual Field objects, each with its own properties, characteristics and values.

A Recordset object can be created by the developer to return data, or it can be returned from executing a command with a Command and Connection object. The information can be obtained from a table in the underlying data source or from a previous SQL statement, query or stored procedure executed through the Command object.

3.4 Structured Query Language (SQL)

The name "SQL" (Groff, J.R. and Weinberg, P.N., 1999) is an abbreviation for Structured Query Language. Date, C.J. and Darwen, H., 1993, state that SQL consists of a set of facilities used for defining, accessing and otherwise managing a specific type of database called a relational database. A relational database is composed (Ritchie, C., 2002) of a number of tables i.e. relations, and is perceived (Date, C.J., 1986) by the user as a collection of tables where a table is an unordered set of rows. The term "relation" is a mathematical expression for such a table.

Fig. 9 below depicts the process by which SQL functions. SQL is functionally expressed in the form of a single statement that is applied to a database via the database's Database Management System (DBMS). A DBMS is a (Papazoglou, M. and Valder, W., 1989) is a system that provides a means of communication with the database efficiently. The purpose of the DBMS is to provide the user with better methods of data access to increase productivity. The DBMS allows the formulation of requests at a logical level, without regard to how the data is stored in physical files. The DBMS determines which physical files are involved in a user request and how these files are to be accessed by referring to a stored data mapping description (schema). It then reads the required database records and converts the data obtained into the form requested by the database requestor (user).

An example of such a statement is a request to return information i.e. SQL sends a request to the database via the DBMS and the database retrieves the information as requested in return through the DBMS.

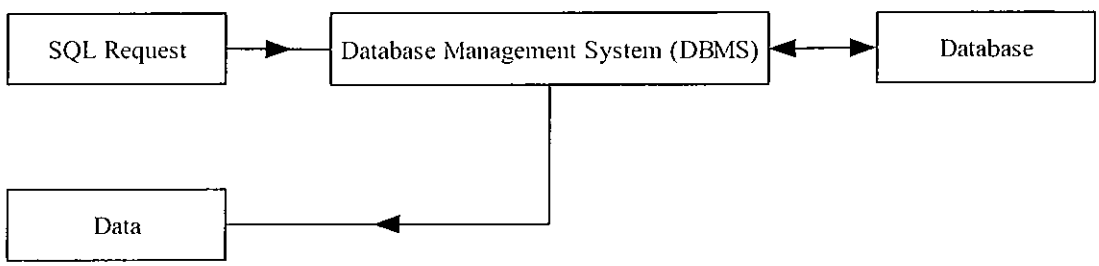


Fig. 9: Using SQL for database access (Groff, J. R. and Weinberg, P. N., 1999)

Although SQL was primarily designed (Groff, J. R. and Weinberg, P. N., 1999) as a retrieval tool it is also used to control all the functions that a DBMS provides to its users.

- Data definition
- Data retrieval
- Data manipulation
- Data access control
- Data sharing
- Data integrity

SQL was used is association with ADO in this project to retrieve specific items of information from the database. After ADO established a connection to the information database a SQL strings were used to query the database and to retrieve specific information. A good working knowledge of SQL had to be established before this was possible.

In the context of data manipulation, SQL can be thought of as an extension of ADO. SQL is a non-procedural language that can be used to communicate to the database what information is required. The database retrieves the information and communicates it to the application. By non-procedural, instead of specifying how to perform a task, SQL communicates the tasks to be performed and allows the DBMS identify the most appropriate procedure to carry it out.

SQL is not a stand alone product, it is part of a DBMS, Connell, J., 1998, states that SQL is more representative of a sub language consisting of about thirty or so specialised statements for database management tasks. These statements are then embedded into another real programming language, like Visual Basic.

A single SQL statement applied to a table or tables in a database can retrieve precise records quickly and easily. The SQL statement can also perform actions on that data retrieved. SQL is referred to as a set-oriented language i.e. it provides access to all the tables in the database simultaneously and retrieves the requisite information in a spreadsheet style Recordset. SQL communicates to the database what tasks need to be performed and allows the DBMS perform them.

Chapter 4: Automated Design

In this Chapter the methodology for customising computer aided design technology with a view to creating an automated system for generating CAD drawings is explained. By writing custom software to automate repetitive tasks it is possible to remove all manual operations involved for designing a part or an assembly.

4.1 Fabrication & design of the air-handling unit external frame

The air-handling unit external frame is an assembly manufactured from eight corner posts and twelve lengths of aluminium cross-section. A diagram depicting the aluminium cross-section can be seen in Fig. 13.

Using the external aluminium frame as a starting point a directory of independent parts was created. Prior to creating the parts, each one was investigated to see how it should behave within the frame assembly. This was accomplished by investigating how each part would be individually manufactured along with how possible design revisions would affect them.

At each intersection where the aluminium cross-sections meet, trihedral corner connections are used to fix the aluminium cross-sections together. At each junction, three supports are placed to lock the junction intersection in place, to support it and restrict the amount of movement possible. The junction supports help create a rigid structure that act as the shell of an air-handling unit. In total, twenty-four supports are used in the frame. The total required number of parts for the assembly is forty-four. In Fig. 11 the junction intersection supports can be seen in blue located at the corners of the air-handling unit frames.

As only one variation of the trihedral corner connection is required it was only necessary to model one trihedral corner connection part file with using the solid modelling software.

Three variations of the same aluminium cross-section part file would be present in the assembly file.

- One set to construct the length of the air-handling unit
- One set to construct the width of the air-handling unit, and
- One set to construct the height of the air-handling unit

To facilitate this, three part files were to be created, each one to represent a different variation of the aluminium cross-section. If one part file were used to represent cross-sections for the length, width and height of the air-handling unit, a modification to that part would result in a modification to all twelve cross-sections resulting in a cube shaped air-handling unit frame.

Therefore by creating three variations of the one part file it would be possible to use one variation set of parts to represent the cross-sections for the length, one set to represent the width and one set to represent the height. A variation to one set would result in only one particular set been modified i.e. the set representing the length, width or the height.

After the directory with the part files was created the next stage of design for the air-handling unit frame was to represent it in the Solid Edge assembly environment and to investigate how the parametric parts could be transferred from representing a regular solid model assembly into a fully parametric solid model assembly.

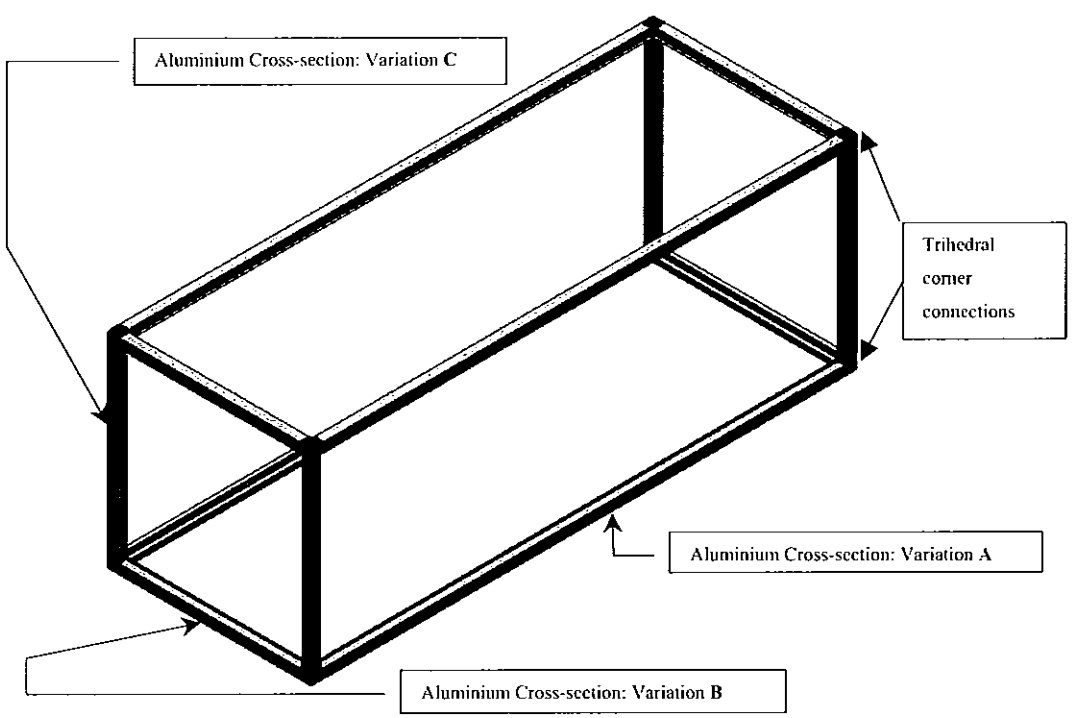


Fig. 10: A computer generated Frame Assembly

Shown above in Fig. 10 is a computer generated air-handling unit frame. The figure clearly indicates why three sets of variations of the aluminium cross-section part are required.

Fig. 11: A Photograph of Frame Assemblies

For the purpose of manufacturing drawings it was not necessary to create a parametric part file to graphically represent the junction supports. Instead by inserting blank part files where the properties were modified, on the bills of materials the junction supports would be displayed.

4.1.1 The trihedral corner connection

The trihedral corner connection is an example of a pre-cast or pre-manufactured part. Purchased from its supplier the trihedral corner connection requires no in house modifications. This particular part is an example of a component that will not be required to have any editable parameters. It is designed to connect the frames aluminium sections together at junctions either end of the frame. While it contributes to the overall dimensions of the frame, regardless of the frame dimensions its dimensions will always remain fixed.

Shown above in Fig. 11 is a photograph of an assembled air-handling unit frame containing trihedral corner connections.

Shown below in Fig. 12 is an image of a computer generated trihedral corner connection. The trihedral corner connection is modelled to the level of detail necessary so that it can represent the connection accurately in the 3D assembly environment as well as the 2D draft environment.

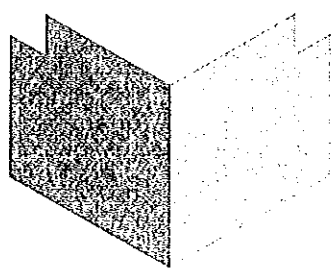


Fig. 12: A computer generated Trihedral corner connection

4.1.2 The aluminium cross-section

The Solid Edge part file that represents the aluminium cross-sections in the assembly is different to that of the trihedral corner connection. The aluminium cross-sections are purchased from its suppliers in long lengths. The aluminium cross-sections must be cut to the appropriate length prior to fabrication of the air-handling unit frame assembly.

The three variations of the aluminium cross-section are directly responsible for the overall size of the frame. A design modification to the size of the frame assembly directly affects the aluminium cross-sections. Therefore it is necessary to have parametric aluminium cross-sections so that an intelligent frame assembly can be developed.

From Fig. 13 below, the profile from a length of aluminium cross section can be seen. Although the profile of the aluminium frame is quite intricate, there is no requirement for it to be parametric. The only required feature to be parametric is the value that the profile will be extruded, i.e. the cross-section's length.

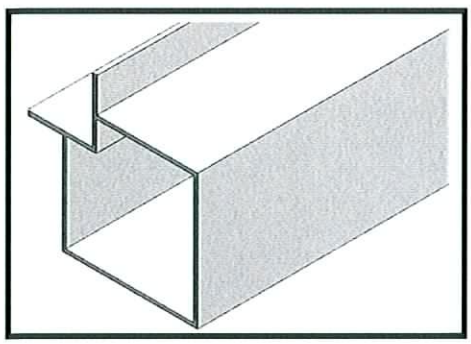


Fig. 13: A computer generated Aluminium cross-section

Shown below in Fig. 14 are necessary part files required to build the air-handling unit frame assembly in the Solid Edge assembly environment. The diagram clearly depicts that no parts from the sheet metal environment are required to assemble the air-handling unit frame.

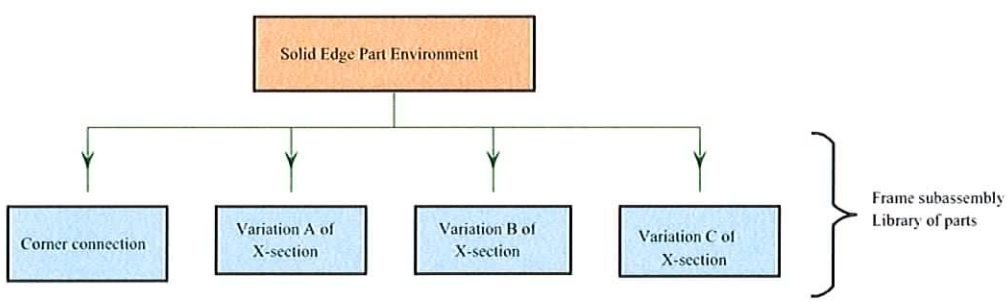


Fig. 14: Frame subassembly library of Parts flowchart

4.2 Introduction to the automated design of a single part

To automate the design process of any part it is important to observe specific methods and procedures during its original construction to facilitate automated design at a later stage.

- The part or component must be created in a 3D or solid modelling environment
- The part must be created with a base or reference point from which all dimensional and geometrical changes will be calculated and applied
- Variable parameters must be clearly defined within the parts file e.g. height, width, length. The variable parameters allow the parts dimensions and geometric location to be modified. By defining the variable parameters custom

software can access and modify them automatically thus removing the manual and repetitive processes involved.

- The part should be saved and closed in a directory for easy access and reference

Once the original part has been created the manual course for editing it involves

- Open the part
- Change the values for its variable parameters
- Update the part
- Save and close it.

This manual practice must be followed each time a modification to the part is requisite.

In order to automate the manual and repetitive tasks involved in adjusting the part file, a custom software application can be utilised. The software must be automotive in that it must have the capability to adjust the part file in the same fashion as the manual process with minimal user input. To achieve this, the software must to be programmed to access the parts variable parameters, as were defined during the construction of the part, apply new values to them, update the part and save it.

The repetitive tasks that the software application removes thus reducing the time taken to perform them are

- Opening the file
- Accessing the variable parameters
- Updating the part
- Saving and closing the part

The only outstanding repetitive task that must be performed on each occasion where a modification to the part is required involves providing the specialist values that must be applied to the variable parameters.

4.3 Introduction to the automated design of an assembly

To automate the design of an assembly similar applies. Software must be programmed to have the capacity to apply new values to all variable parameters of all parts in the assembly. This also includes all the parts positional and geometrical information. The software must be programmed to also understand how parts and components in an assembly are related and how they interact so that the software can move them and place them into the correct locations during and after their dimensions and properties are revised.

4.4 Creating a parametric assembly

All of the parts in the directory that were created to represent the air-handling unit frame were created in the Solid Edge part environment, thus each part was created independent of each other and modelled to do a specific job. In the Solid Edge part environment it is only possible to design and model one Solid Edge part at a time. To examine the behaviour of a part in an assembly, it must be placed into that environment with the other parts of the assembly and constructed together.

To create a subassembly from the library of parts they must be imported into an assembly file. In the assembly environment it is possible for multiple parts to exist within one file. The assembly environment contains tools to build assemblies from libraries of parts. By placing parts into an assembly the software generates links from the assembly file back to the part file's parameters and attributes.

A modification to a part in an assembly will result in both the assembly file being edited along with the specific part file. This process also works in reverse. A modification to a part file that is included in an assembly will update the part file as well as the assembly file.

To make a modification to the length of the unit a specific link was placed from the assembly file, to the parameter in the part file controlling its length. This allowed the length of the part to be accessed and modified from the assembly file.

Two more specific links were created from the frame assembly file. One of the links was created back to the part file controlling the width of the unit; the final link was created back to the part file controlling the height of the unit.

After the assembly file was saved and closed, a fully working parametric model of a representative air-handling unit frame subassembly was generated. The process for inserting files from the part environment to a file in the assembly environment can be seen in Fig. 15.

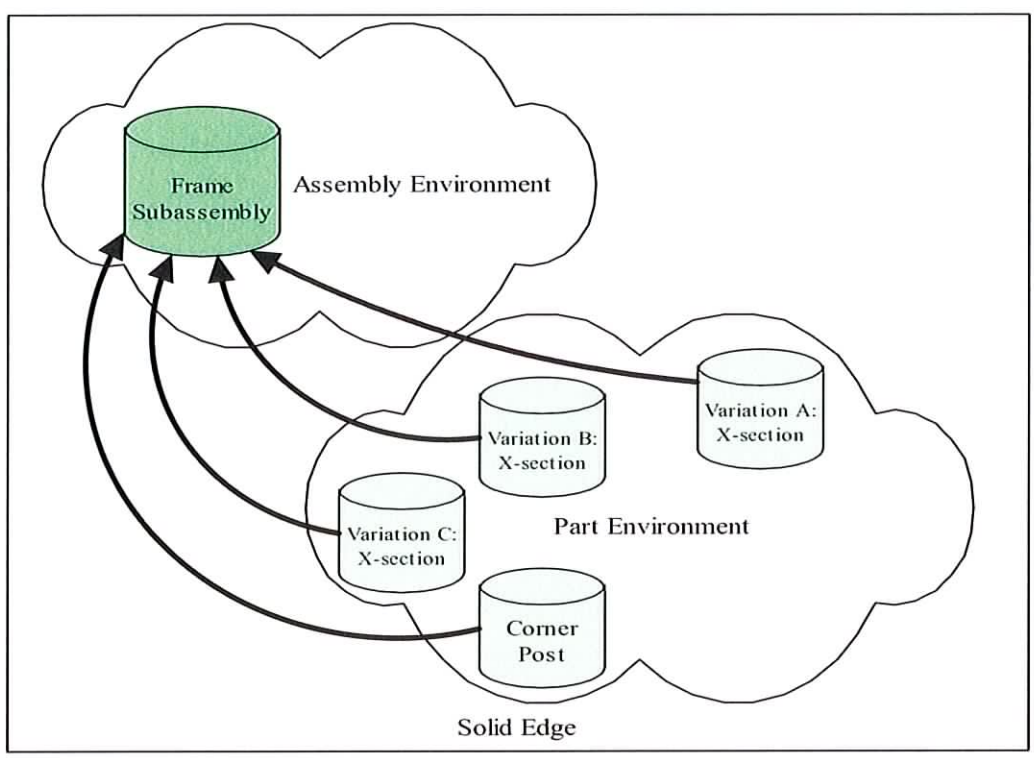


Fig. 15: Part and Assembly Environments

4.5 Writing bespoke software for automation

As with software that can be created for a part file there are multiple sources from which software can ascertain information about assemblies before it begins to construct and or modify them.

- Programming code
- Databases
- Text files

- Excel spreadsheets
- Calculations and algorithms
- User input

Automation software can be written to automate designs using solid modelling software in two primary ways.

- The first method as described in section 1.3, involves writing custom software to design a part using solid modelling software in the same fashion as it would be done manually i.e. using the software's same processes and procedures, but automating them (using the custom software) to build each part, feature by feature and step by step.
- The second approach, as described in detail in section 4.2, involves writing custom software to open existing part and assembly files and to change their parameters and update the file as an alternative to designing and building the files from scratch. When the software has opened a part or an assembly it can then gain access to its parameters and make adjustments to them.

The second approach is deemed a more efficient approach as more time can be saved in completing the design. As well as changing dimensions and moving parts the software can be programmed to add and remove parts from the assembly also.

Previously Data Access technology was discussed and it was mentioned how ActiveX technology could be used to access Solid Edge via custom written software to automate repetitive manual processes and commands. In this next section the procedures and operations for writing software will be described in full.

Writing bespoke software for the automation of Solid Edge involves creating custom applications, or custom application extensions that will operate in tandem with Solid Edge. The Solid Edge programming utilities provide the tools to automate Solid Edge with ActiveX technology. With these custom applications, it is possible to modify and enhance standard commands to tailor Solid Edge for specific needs. To customise Solid Edge, it is necessary to access a programming tool to be used as an automation client. An automation client is an application that can access objects from other applications that support ActiveX technology.

The custom application is described as working in tandem with Solid Edge as it can run independent of it, but for a custom application to perform a task or a series of tasks to automate Solid Edge and it commands, Solid Edge must be installed on the specific computer so that the custom application can gain access to it. Once Solid Edge is installed on the computer the custom application must open up Solid Edge, or Solid Edge must be open for it to perform its operations.

Therefore, the custom application is not replacing Solid Edge or what it does on the computer; instead it provides a pathway for automating the manual processes in producing solid models and draft documentation.

Using programming tools and languages, such as Visual Basic or Visual C++ it is possible to create custom applications using ActiveX technology to gain access to Solid Edge and automate its commands.

To the user the custom application will be represented on screen as a Graphical User Interface (GUI), or as a series of GUIs depending on the design of the software and on the information that it requires to perform its tasks.

4.6 Using bespoke software to automate assembly design

The variable parameters that were created for the air-handling unit frame Solid Edge assembly file to facilitate automation from a custom application include

- Overall length
- Overall width
- Overall height

These variables are all present with in the assembly file. Each variable contains a link back to the specific part file from which it originated. When each variable has a new value applied, it updates the new value back to the part file and updates both the part and assembly file.

In Fig. 25 the GUI that was designed to complete this section of the air-handling unit design, it can be seen that the user input necessary to complete the frame is different to the variables listed above. Instead the user is asked to enter specific design information

regarding the number of air filter units that will make up the inside of the completed air-handling unit.

The custom software is programmed to use this design information to calculate the values, via algorithms, that must be applied to the variable parameters listed above. After the calculations have been completed the software will then perform an automatic process resulting with a complete air-handling unit frame assembly open in the Solid Edge assembly environment. The processes performed by the custom software to complete the assembly are an exact mimic of the manual process that would be followed to generate the same assembly.

The details of the custom application that was designed to automate the air-handling unit external frame can be seen in section 6.1. The details of all the programming code for this section of the custom application can be found in Appendix C, page 2, Design Stage 01 – frmStg_1.frm.

Chapter 5: Case Study

5.1 Industrial model, Danann Clean Air Services Ltd.

Danann was founded in November 1997, in Swords Co. Dublin by two mechanical engineers experienced in the field of air-handling unit design, Michael De Burke, and Dave McKay. Since then the company has migrated from two to a thriving company of nearly 25 employees.

Currently Danann has 8 per cent to 10 per cent of the Irish market share in air-handling unit manufacture. Danann's goal over the next five years is to be Irelands largest manufacturers having increased their market share to 30 per cent. In addition to this they intend to break into the international market. Danann hope that involvement with this research project will go some way in achieving these goals.

Danann are based in Swords, Co. Dublin. Currently, there is a team of four engineers at Danann two of which concentrate in both sales and design leaving the remaining engineers to concentrate on production and manufacture. A team of up to twenty operatives exist in the factory; this team is divided into two groups.

- Fabrication
- Assembly

The two groups of staff work independent to each other. The manufacturing staff will work ahead on projects that have been confirmed so that when the time arrives the assembly team will have everything prepared for them.

Currently the fabrication team is made up of one unit of six staff while the assembly team consists of two units of four staff.

5.2 Investigation of AHU design at Danann

Some time was spent analysing existing design and manufacturing methodology at Danann. The following chart in Fig. 16 shows a graphic description of the company hierarchy.

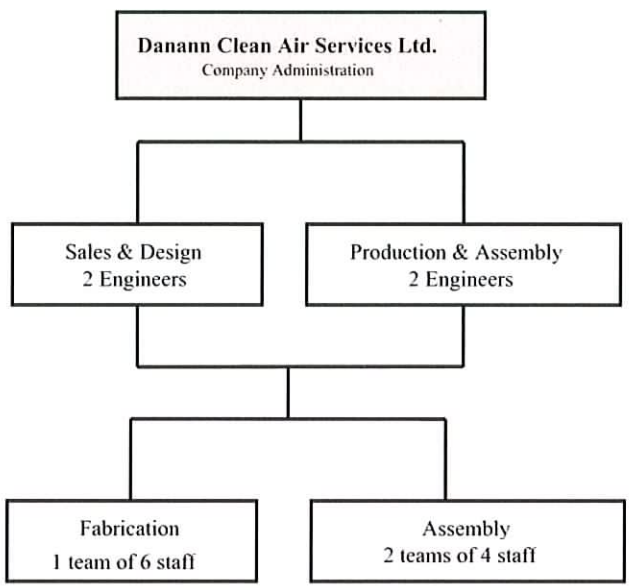


Fig. 16: Danann's company hierarchy

The company has two directors, both of whom are Mechanical Engineers. Each director is involved with different divisions in the company as shown in the above diagram. While the first division concentrates on sales of the air-handling units they also take responsibility for the design of the units. The other division takes control over the production and assembly for Danann's products.

5.3 Investigation into existing manufactured products

Initial investigation of all products discovered that two types of air-handling system are manufactured by Danann.

- Constant-air-volume system
- Variable-air-volume system

Sun, T.Y., 1994, defines constant-air-volume systems as systems that maintain a constant supply of air volume to each conditioned space, and rely on varying the temperature difference (i.e. changing supply air temperature) to meet the load requirements in different conditioned spaces. A schematic diagram of a constant air volume system can be seen in Fig. 17 while the Psychrometric processes can be seen in Fig. 19

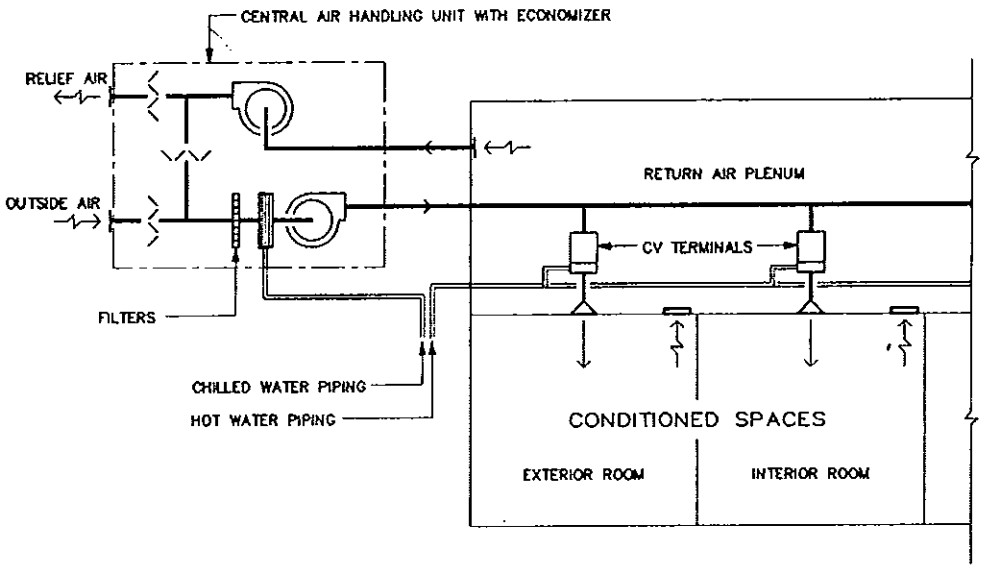


Fig. 17: Constant Volume terminal reheat System (Sun, T.Y., 1994)

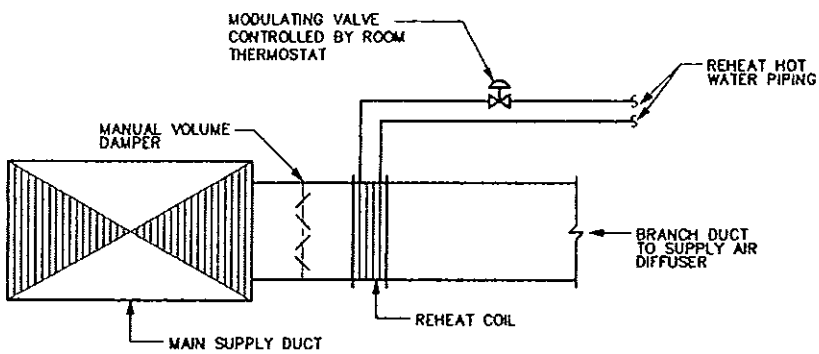


Fig. 18: Arrangements of Reheat at a duct branch (Sun, T.Y., 1994)

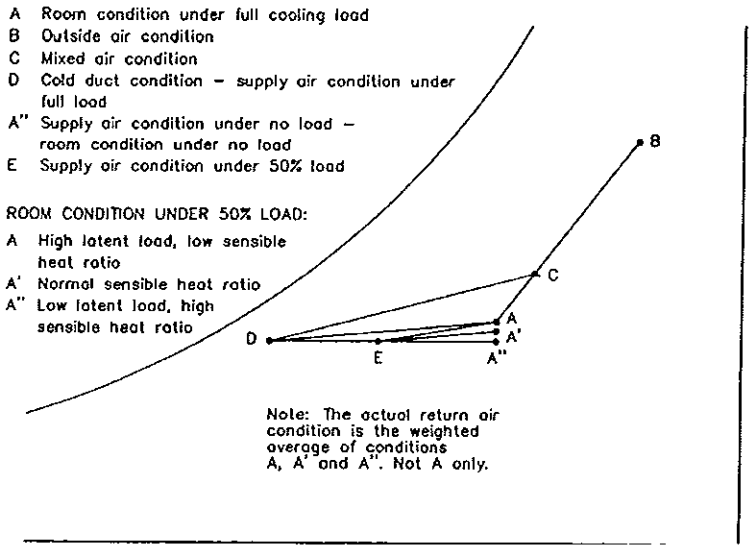


Fig. 19: Psychrometric processes of a constant volume terminal reheat system (Sun, T.Y., 1994)

Variable-air-volume (VAV) system with no volume compensation – variable volume systems operate on the principle of modulating the amount of supply air, rather than varying the supply air temperature to each conditioned space. Sun, T.Y., 1994, defines variable-air-volume systems where the supply air temperature in a true variable volume system is held constant. This implies that a variable volume system can either be a cooling system or a heating system, but not both. A schematic diagram of a constant air volume system can be seen in Fig. 20 while the Psychrometric processes can be seen in Fig. 22.

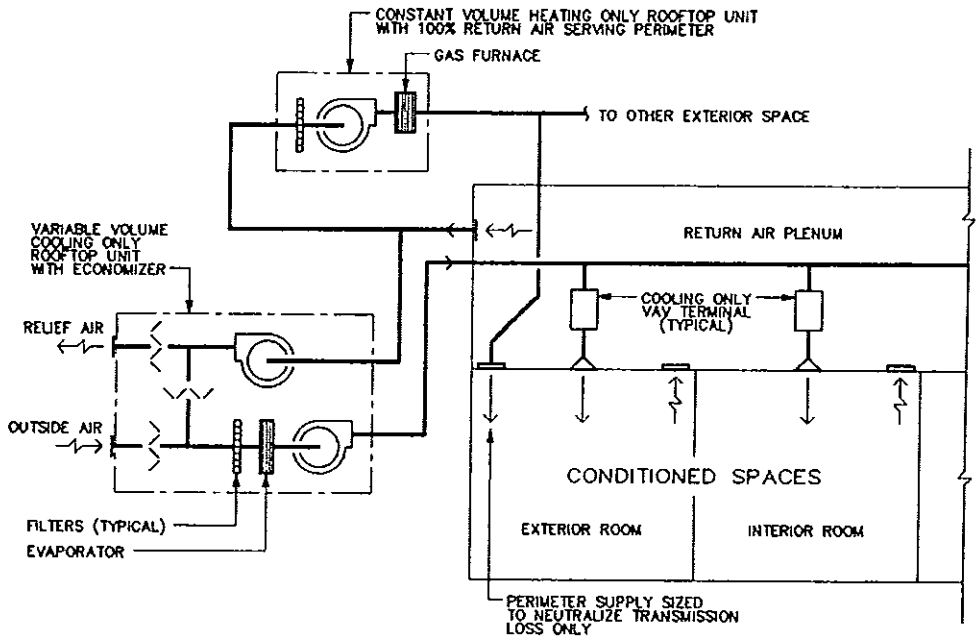


Fig. 20: Variable Volume System with heating-only perimeter system (Sun, T.Y., 1994)

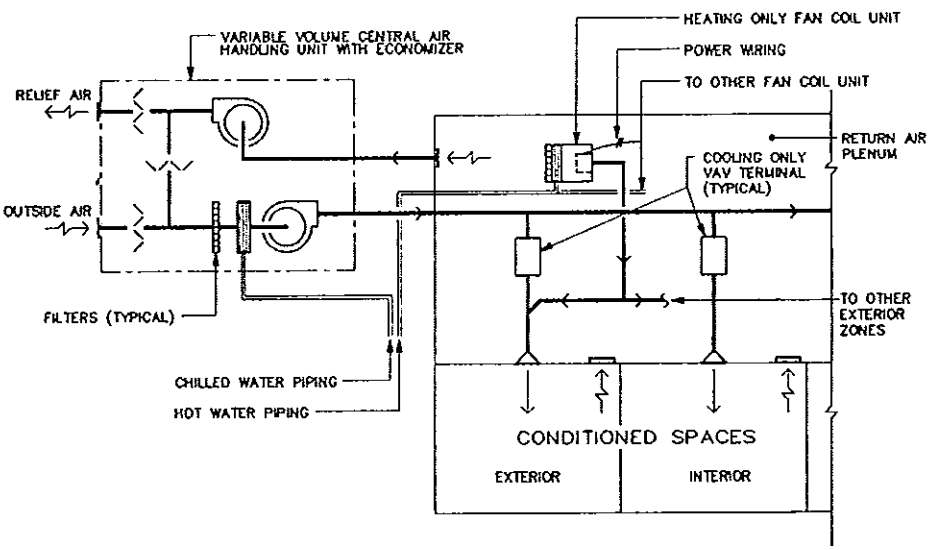


Fig. 21: Schematic: Variable Volume System with fan coil units serving building perimeter (Sun, T.Y., 1994)

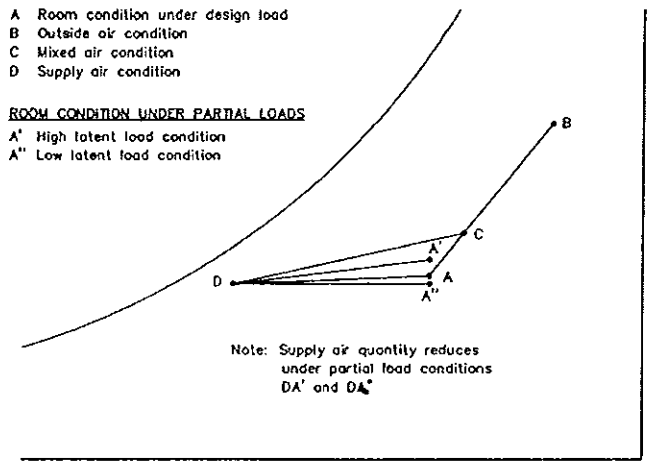


Fig. 22: Psychrometric process of a cooling-only variable volume system (Sun, T.Y., 1994)

5.4 Investigation into existing design systems and techniques

Danann use a series of software packages to produce the various documentation required.

- All quotations, BOM's and Cut Lists are generated from spreadsheets.
- All working drawings and diagrams are created from a set of views in a 2D drafting CAD package.

From this investigation it was ascertained that the design system at Danann was very one-dimensional. There was no association between any of the files from the software packages that Danann was using. In the event of a design modification to an air-handling unit all documentation and drawings associated with it would have to be reviewed for changes.

One of the principal objectives of this research was to design and create a system for Danann where all of the documentation concerned with an air-handling unit would immediately be updated upon any revision to any part of an air-handling unit or its design.

The intended effect of this system would be to reduce the amount of repetitive processes necessary during creating a design and also in implementing a design revision, thus reducing the time required to complete a full set of documentation for an air-handling unit including design revisions and modifications.

5.5 Danann air-handling units general characteristics

The air-handling units are designed and manufactured in modular system so that they are suitable for outdoor or indoor installation. General characteristics of a Danann air-handling unit are strength of frame, split construction and noiseless operation. The casing is generally made from aluminium profile. The aluminium profiles are connected with special trihedral corner connections made of cast aluminium, forming the supporting casing of the side panels.

Fig. 23: Typical Frame Construction for Air-Handling Units

5.5.1 Typical components

Typical components found in a Danann air-handling unit include:

- Aluminium frame
- Side panels
- Fan section
- Coils section
- By-pass section
- Humidification section
- Filters section
- Mixing box section
- Attenuator section
- Heat recovery section
- Electric resistances section
- Supporting Base

See Appendix B for photographs.

Chapter 6: Software Development

An imperative feature of the design system is the capability to model air-handling units in 3D as accurately as possible. With this in mind, the automated design system (bespoke software) was developed not only to try and model the units in 3D but also to model them in the same fashion as they are assembled on the factory floor.

After analysing the assembly and manufacturing processes of the units, and with the design system in mind, assembly of the units was defined into three stages:

- Stage 1: Exterior aluminium frame assembly
- Stage 2: Interior subassembly assembly, and fitting
- Stage 3: Exterior panel assembly and fitting

Stage 1 was chosen to represent the design of the external frame of the air-handling unit along with the design of the air-handling unit's base frame. At completion of Stage 1, the external shell of the air-handling unit would be complete including a supporting base frame.

Stage 2 was designed to be the natural progression of Stage 1. Stage 2 would include the design of the necessary components and subassemblies for inside the air-handling unit, components such as filters, coils, dampers and attenuators.

Stage 3 of the design process is the final stage. It is designed to specify and to deal with the exterior panels of the air handling. Only after the internal components of the air-handling unit were placed within the air-handling during Stage 2 would it be possible to begin designing them to the correct sizes and arrangement. After Stage 3 is complete next step is to begin creating the construction documentation for the air-handling unit.

Each stage of the design phase would have its own graphical user interface. Each graphical user interface would contain the facilities for the user to input the required design information in a convenient manner for that particular stage. The graphical user interfaces would appear in the same logical order to the user as the units would be assembled on the factory floor.

This style of automated design system provides the engineer/designer/user with the ability to enter convenient and understandable design information, not just a series of numbers into the system. This way the air-handling units can be designed and modelled on the computer and not just drafted it up in a number of 2D views. The engineer/designer/user no longer has to try and visualise the unit from a set of drafted two-dimensional views, nor must it question the consistency of its work and doubt if the air-handling unit can be built. The air-handling unit can be seen in 3D on the computer screen where it can be analysed to ensure that everything has been designed satisfactorily.

6.1 Stage 1: Exterior aluminium frame assembly

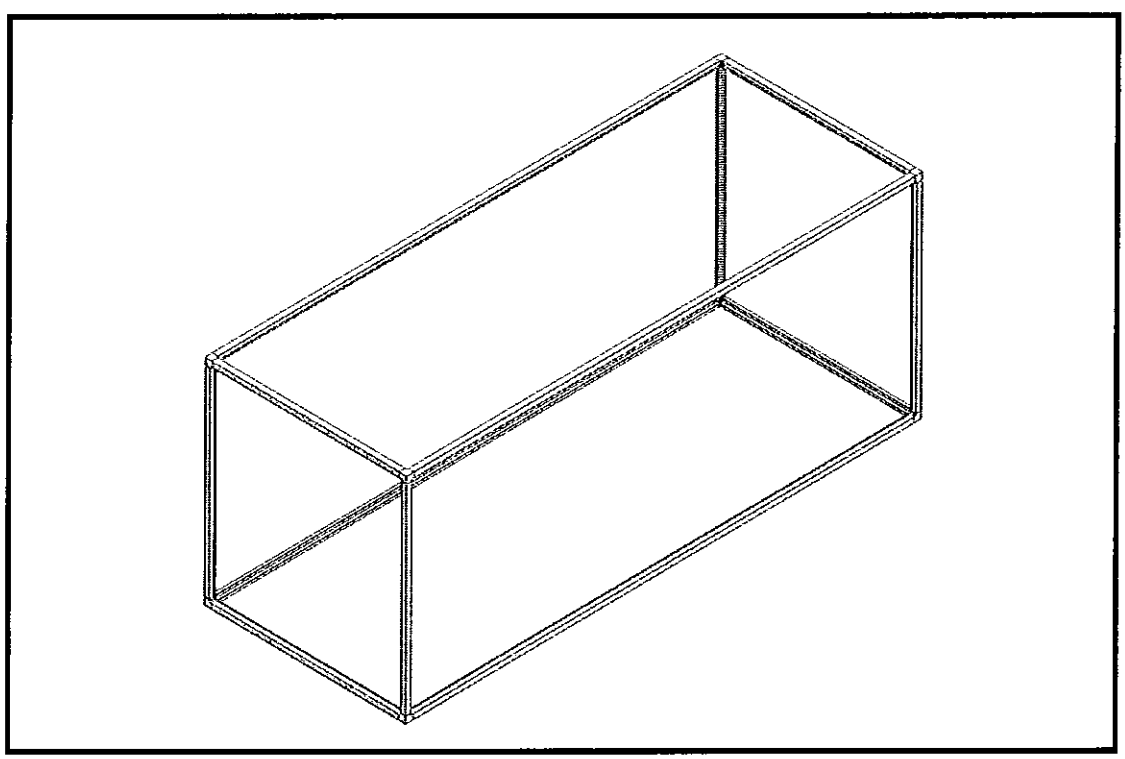


Fig. 24: Stage 1 - An air-handling unit aluminium frame

This section describes the design of the software for Stage 1. Fig. 24 is an example of the external frame of a typical single story air-handling unit. The make up this type of particular frame consists of three sets of different lengths of aluminium cross-section and eight cast iron trihedral corner connections. Each set of aluminium cross-sections contains four bars of the same length.

From a manufacturing point of view the design criteria that is compulsory for this particular structure of external frame includes its:

- Overall length
- Overall width
- Overall height
- Aluminium cross-section specification

After providing this information into the software it is possible to programme the code to calculate the lengths and specifications for all the components in the external frame assembly.

The software eliminates the need for the user to supply information that is consistent for every air-handling unit. This includes:

- Trihedral corner connection dimensions
- Aluminium cross-section dimensions
- Various aluminium cross-section lengths

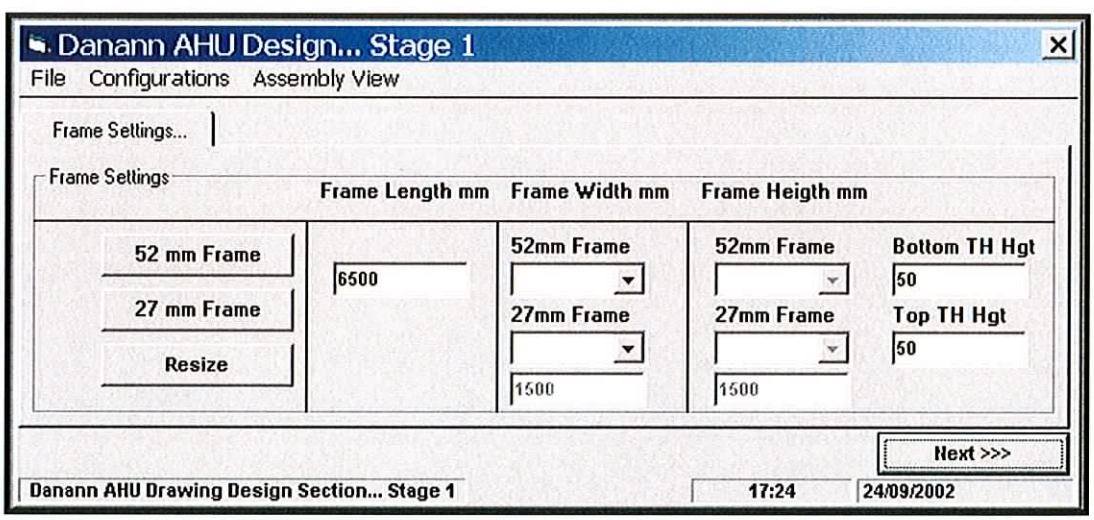


Fig. 25: Stage 1

Fig. 25 displays the graphical user interface that is designed and programmed to represent Stage 1. From it can be seen that there are four sections for the user to input information about the external frame of the air-handling unit.

- Frame length. The Frame's length is ultimately determined by a combination of the air-handling unit's design requirements and the maximum space that is available for it on site where it will be installed. An educated guess can be used to initially to provide a value for the Frame length input section if the final value has not been determined and at a later time the resize button can be used to correct the value.
- Frame width. Most air-handling units have a panel filter section, a bag filter section or a combination of both inside its external. Any filter section is required to fill the cross-sectional area of the air-handling unit so that all air that will passes through will be filtered. The panels used to generate the filter section are of standard sizes (610 mm x 610 mm and 305 mm x 305 mm). From the number of panels that are used along the horizontal of the air-handling unit, it is possible to calculate the external width using the frames cross-sectional specification also.
- Frame height. The number of panels that will be used along the air-handling unit's vertical cross-section is used to calculate the external height of the air-handling unit.
- "Bottom TH Hgt" and "Top TH Hgt". Bottom TH Hgt is an abbreviation for Bottom TopHat height, and Top TH Hgt is an abbreviation for Top TopHat height. The bottom TopHat and the Top TopHat are each a fabricated piece of sheet metal manufactured to mount inside the air-handling unit frame. The cross-section of the TopHat can be shown below in Fig. 26. The top and bottom TopHats span the internal width of the air-handling unit along the bottom and the top. The bag filter or panel filter sub-assembly will be fitted between the two TopHats.

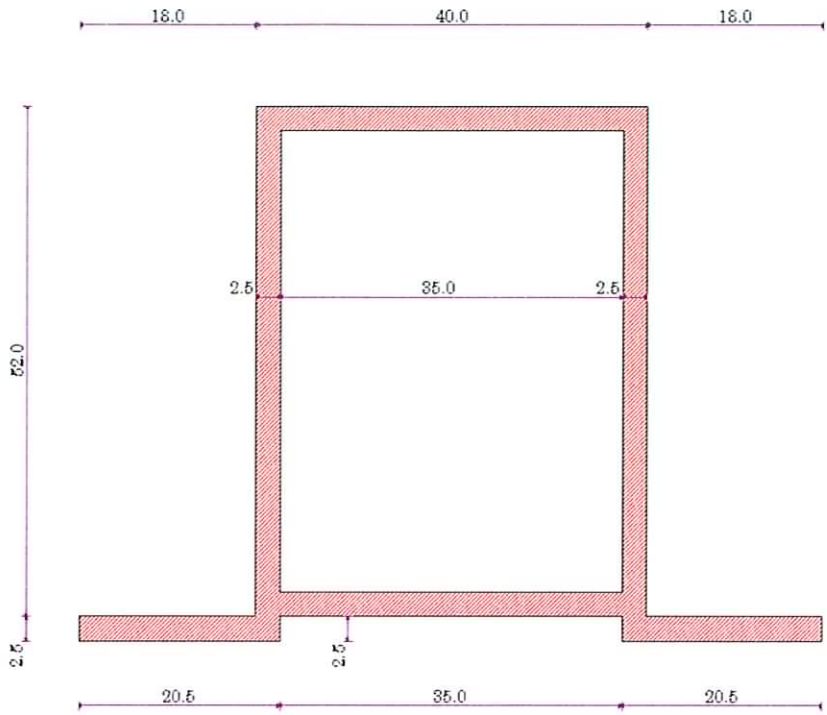


Fig. 26: Top Hat profile

These characteristics were identified earlier as the key sources of information necessary to build the air-handling unit external frame.

The graphical user interface for Stage 1 is designed so that the method for entering information in to it is straightforward. From Fig. 25 drop down menus can be seen on the graphical user interface. The drop-down menus will only contain options that are valid when designing the air-handling unit. They help the user make a good choice.

The software for Stage 1 calculates the exterior dimensions of the air-handling unit from the information that the user supplies (i.e. width and height). The user specifies the format in which the panel/bag filter sub-assembly will be manufactured. From this format the software can calculate the air-handling unit's external dimensions.

The following formula is used to calculate the frame width:

Frame width = {((Number of filters in X direction)* 610) + (109)}

Formula 1: External frame Width

In Formula 1 610 represents the width of a filter in millimetres. The 610 mm value is fixed as the two types of filter that can be used in an air-handling unit contain the same external dimensions.

In the formula 109 represents the thickness of the aluminium cross-section in millimetres. The value 190 mm will vary depending on the type of aluminium cross-section that may be used.

The overall dimensions for the height can be calculated from the following formula.

Frame height = ((Number of filters in Y direction) * 610) + (109) + (height of the Upper TopHat) + (height of the Lower TopHat)

Formula 2: External Frame Height

The values 610 mm and 109 mm represent the same values as already explained for Formula 1.

From the dialog in Fig. 25 the length of the air-handling unit is determined from entering its value into graphical user interface. After all of the necessary design information has been entered the user must click on one of the buttons on the left of the graphical user interface labelled "52 mm Frame" or "27 mm Frame" for the software to commence modelling the air-handling unit frame.

- 52 mm Frame - This button is used to define that the user wishes to use a 52 mm aluminium cross-section for the external frame of the air-handling unit and to make the software go and model the external frame of the air-handling unit. See Fig. 27.
- 27 mm Frame - This button is used to define that the user wishes to use a 27 mm aluminium cross-section for the external frame of the air-handling unit and to make the software go and model the external frame of the air-handling unit.
- Resize - After the external frame of the air-handling unit has been modelled by the user using the "52 mm Frame" button or the "27 mm Frame" button the user can input new specifications into the input sections and can click on the "Resize" button to apply the new specifications.

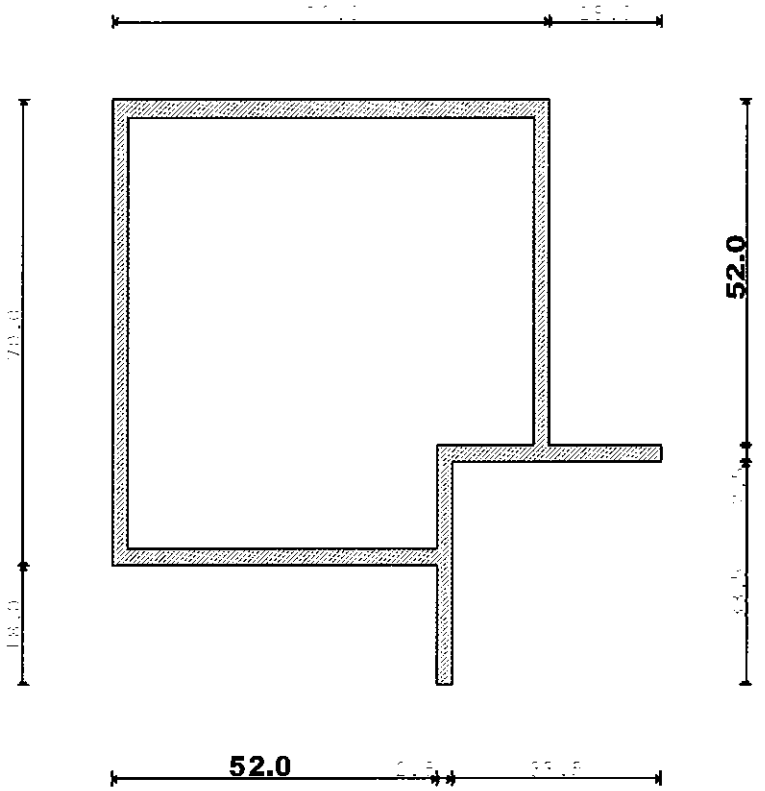


Fig. 27: 52 mm aluminium cross-section

By clicking on the appropriate button the user is indicating to the software that the length, width and height of the air-handling unit frame have been defined and that the type of aluminium cross-section that is to be used is a 52 mm Frame or a 27 mm Frame. See Fig. 27 for the profile of a 52 mm aluminium frame. The TopHat profile is named as a 52 mm TopHat due to the 52 mm dimension seen in heavy black illustrated on Fig. 27. Once the user clicks on the button the software will begin to model the air-handling unit frame.

Fig. 28 is a Flow Chart illustrating the processes and procedures that the software cycles trough after the user has chosen the "52 mm Frame" or "27 mm Frame".

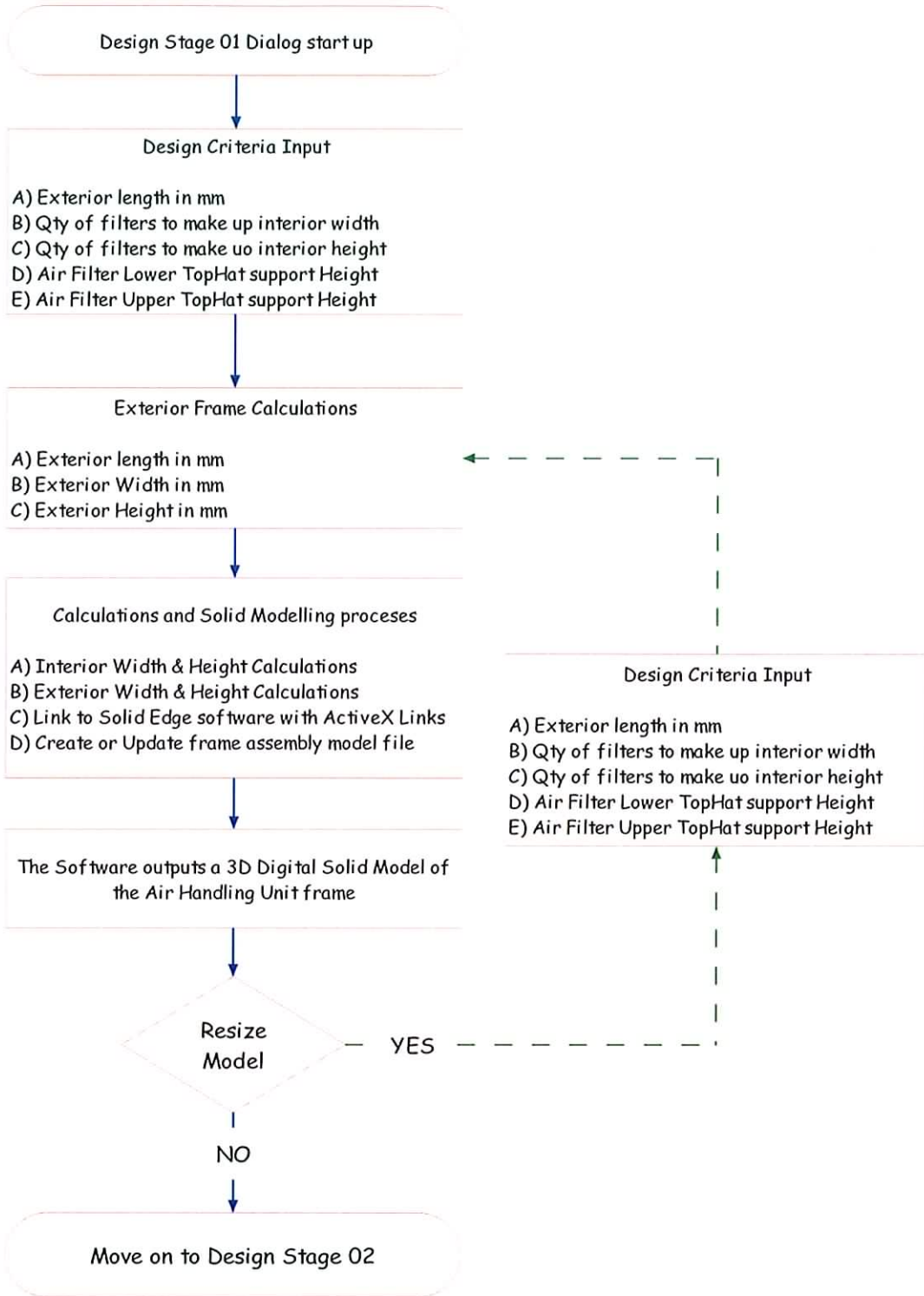


Fig. 28: Stage 1 Input - Output Flow Chart

The beginning of Stage 1 takes place when the dialog first opens up. The end of the Stage 1 takes place when the transition to Stage 2 is made. From the Flow Chart it can be seen that it is possible to resize the air-handling unit frame as many times as is necessary to get the frame to the correct dimensions.

Once the dialog for Stage 1 has opened up it cannot take further action without instructions from the user. Before it can begin to model the air-handling unit frame it must first calculate the frames external dimensions from the dimensions of the components in the frame assembly. As the user begins to input information the dialog immediately begins its calculations.

Fig. 29 is a section of code that illustrates the calculations the dialog completes when information regarding the frames width is supplied.

```
Public Sub Combo2_Click()
cmdFrame_A.Enabled = True
cmdFrame_B.Enabled = False
Combo1.Enabled = False
Combo4.Enabled = False
Combo3.Enabled = True
Text1(1).Text = (Val(Combo2.Text) * 610) + 1209
End Sub
```

Fig. 29: Programme Code for Frame Width Calculations

The second last line of code beginning "Text1(1).Text" is the algorithm for calculating the external width of the air-handling unit. Text1(1) in the algorithm refers to the text box on the dialog that displays the overall width of the air-handling unit frame after it has been calculated. Combo2 in the algorithm refers to drop-down menu list that displays the list of values that can determine the width of the air-handling unit.

Fig. 30 shows a section of code that illustrates the calculations the dialog completes when information regarding the frames height is supplied.

```
Public Sub Combo3_Click()
cmdFrame_A.Enabled = True
cmdFrame_B.Enabled = False
Combo3.Enabled = True
Combo4.Enabled = False
Text1(2).Text = (Val(Combo3.Text) * 610) + 1209 + Val(txtBotHpanel.Text) + Val(txtTopHpanel.Text)
End Sub
```

Fig. 30: Programme Code for Frame Height Calculations

The third last line of code beginning "Text1(2).Text" is the algorithm for calculating the external height of the air-handling unit. Text1(2) in the algorithm refers to the text box on the dialog that displays the overall height of the air-handling unit frame after it has been calculated. Combo3 in the algorithm refers to dropdown menu list that displays the list of values that can determine the width of the air-handling unit.

At the end of the algorithm there are two more values included in the formula that are not found in the formula for the external width of the air-handling unit frame.

Val(txtBotTHpanel.Text), and Val(txtTopTHpanel.Text)

As stated previously the external width and height are exclusively reliant on the format at which filters are to be arranged across the internal cross-sectional area of the air-handling unit. The matrix of filters is mounted onto fabricated pieces of sheet metal referred to as "TopHats", named after the shape of its profile, that span the internal width of the air-handling unit. The TopHats are mounted above and below the matrix of filters so the vertical height of the TopHats must be included into the calculation for the external height of the air-handling unit frame.

Two text boxes were positioned onto the dialog for Stage 1 so the values for the vertical heights of the TopHats could be supplied or edited. As their values were supplied or edited, the value for the external height of the air-handling unit frame could be re-calculated automatically by the software.

Fig. 31 shows the calculations performed when the TopHat dimensions are edited.

```
Public Sub txtBotTHpanel_Change()
Text102.Text = (ValCombo3.Text) * 610 + 1090 + Val(txtBotTHPanel.Text) + Val(txtTopTHPanel.Text)
End Sub
```

Fig. 31: Calculations performed after the TopHats are edited

The calculations performed in Fig. 31 are the same as the calculations performed in Fig. 30. Once all of the calculations have been performed the software can build the air-handling unit frame.

The software builds the air-handling unit frame to the specifications the user inputted to the system. Firstly, the software executes a set of procedures to ensure that no incorrect information will be passed through to software. The first of these procedures is to check a value for the external height of the air-handling unit frame has been supplied. The next of these procedures is to check a value for the external width of the air-handling unit frame has been supplied. Fig. 32 shows the code that performs the safety checks.

```
If Combo1.Text = "" And Combo2.Text = "" Then
MsgBox "Select the number of filters so a width can be Calculated", vbInformation
Exit Sub
End If
If Combo3.Text = "" And Combo4.Text = "" Then
MsgBox "Select the number of filters so a Heigth can be Calculated", vbInformation
Exit Sub
End If
```

Fig. 32: Code representing safety procedures

Fig. 32 is a safety feature so that the software can not progress to the next stage of design if there is a blank value for the width and/or height of the air-handling unit. If the software detects a blank value for the width and/or height of the air-handling unit it will display an error message to the user stating that a value must be selected in order to proceed.

Once all of the safety procedures have been processed and found to be valid, the software will build the air-handling unit frame. To build the air-handling unit frame the software must execute one of the following:

- Connect to Solid Edge if it is already open on the system
- Open Solid Edge if it is closed

Before the software can build an air-handling unit frame Solid Edge must be open on the computer for it to do so. This code illustrating how the software achieves this is shown in Fig. 33.

```
'Turn on error handling.
On Error Resume Next
'Connect to a running instance of Solid Edge.
   Set ObjApp = GetObject(, "SolidEdge.Application")
   If Err Then
'Clear the error.
   Err.Clear
'Start Solid Edge.
   Set ObjApp = CreateObject("SolidEdge.Application")
   End If
'Turn off error handling.
   On Error GoTo 0
'Make the application window visible.
   ObjApp.Visible = True
```

Fig. 33: Generating a link to Solid Edge

This software achieves this by using ActiveX technology as described in Data Access Technology. From line 4 in Fig. 33 the programming code refers to Solid Edge as an "object":

```
GetObject(, "SolidEdge.Application")
```

The code gains access to Solid Edge by getting the **"SolidEdge.Application"** object. To open up Solid Edge the code creates an object as shown:

```
Createobject("SolidEdge.Application")
```

Here the software creates the **"SolidEdge.Application"** object to open or to start up Solid Edge.

This particular command will be reproduced every time that the software wishes to execute a command related to Solid Edge.

After the link to Solid Edge has been created the software must access various objects associated with Solid Edge depending on the various commands and operations that it must process.

Firstly the software must open up an existing document. There the software must access the "documents Collection" object in Solid Edge to do this.

```
'Access the documents collection.
  Set ObjDocs = ObjApp.documents
  Set objDoc = ObjApp.Documents.Open(App.Path & "\Drawings\Top_Level_Assembly\TLA_Frame_Assembly.asm")
```

Fig. 34: Accessing the Solid Edge Documents Collection

From Fig. 34 it can be seen that the software accesses the Solid Edge documents Collection and selects a specific file on the computers hard disk.

In line 2 of Fig. 34 the code states:

Set ObjDocs = ObjApp.documents

This line of code is used to access the Solid Edge "documents Collection".

ObjDocs. Obj being an abbreviation for object, Docs being an abbreviation for documents, i.e. the documents object

Will be set to equal:

ObjApp.Documents. ObjApp being an abbreviation for Object Application, i.e. Solid Edge. The documents of ObjApp.Documents being the documents collection of Solid Edge, hence, the ObjApp.Documents.

The line of code represents the custom application gaining access the Solid Edge documents collection using ActiveX technology.

After the documents collection has been accessed the software must ensure that the correct Solid Edge environment is activated. Since the frame of the air-handling unit to be built is an assembly the software must make sure that Solid Edge is running in the assembly environment.

```
'Check to make sure the active environment is assembly
  If ObjApp.ActiveEnvironment <> "assembly" Then
    MsgBox "This program must be run in the assembly environment.", vbInformation
    End
  End If
```

Fig. 35: Procedure to ensure the correct Solid Edge environment is active

At this stage of the software's procedure the open Solid Edge assembly file is blank. This is because the file will be the top level assembly document when air-handling unit has been designed and built. From this point onwards all further assemblies will be inserted into this assembly file and modified accordingly.

The next step is to add the correct existing air-handling unit frame assembly.

```
'Adding a part document from a file to the assembly document
  Set objOccurrences = objDoc.Occurrences
    If objDoc.Occurrences.Count <> 0 Then
      For Z = 1 To objOccurrences.Count
        objOccurrences.Item(Z).Activate = True
      Next Z
        For I = 1 To objOccurrences.Count
          objOccurrences.Item(1).Delete
        Next I
    End If
  Call objDoc.Save
Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_A1
```

Fig. 36: Adding the correct assembly file

Before the software adds an existing document to the top level assembly it must warrant that there are no existing parts or assemblies present. The air-handling unit frame must always be the first document to be added to the assembly. To certify the software runs through a loop to investigate the parts and assemblies present so that they may be removed. Once the investigation has been completed the software then adds the correct Solid Edge assembly file to the top level assembly.

From Fig. 36 it can be seen that ActiveX technology plays a distinct role in this command. Before the software can access and remove an object within Solid Edge it

must first access it using ActiveX technology. The software must access existing parts (occurrences) or components in the Solid Edge assembly via the

ObjOccurrences, i.e. the Occurrences objects

It gains access to the occurrences through the Solid Edge documents objects

Set objOccurrences = objDoc.Occurrences

The ObjDoc variable was linked to the Solid Edge documents objects earlier in this Visual Basic command. Therefore, the Visual Basic custom application is accessing Solid Edge by using ActiveX technology to gain access to the Solid Edge objects.

A diagram explaining the Solid Edge object Hierarchy was explained and shown in Data Access Technology.

From Fig. 36 it can be seen that the code refers to the air-handling unit frame assembly as "**TESTFILE_A**". Earlier on in the procedure the software defined **TESTFILE_A** as a variable within the Visual Basic Code whose value was set to:

```
TESTFILE_A = App.Path & "\Drawings\Frame_Large\Frame_Assembly_Large.asm"
```

Fig. 37: TESTFILE_A

At this point of the automated design of the air-handling unit frame it is necessary to apply the users design specifications to the Solid Edge assembly. The software must now access the air-handling unit frame assembly and modify it so that its properties match those that that the user specified.

The air-handling unit frame assembly file was designed so that it would be possible to modify it through its variables. When the assembly was created, it was created with links (variables) back to the parts that the assembly was crated from. A change in the value of a variable results in a change to the related part as well as to the air-handling unit frame assembly.

For the software to gain access to the variable in the Solid Edge assembly Environment the following line of code was written

```
'Access the variables Collection
Set objVars = objDoc.variables
```

Fig. 38: Accessing the Solid Edge Variable Collection

Similar to the manner in which the software accessed the Solid Edge documents collection Fig. 38 shows the software accessing the Solid Edge variables. For the changes to be fully implemented the software will then check to make sure that every component in the assembly is fully active.

```
For Z = 1 To objOccurrences.Count
objOccurrences.Item(Z).Activate = True
Next Z
```

Fig. 39: Activating all the parts in the assembly

Once the variables have been accessed and all the parts have been activated the software can edit the air-handling unit assembly's variables and update the assembly.

```
'Change the variables to their new Value
Call objVars.Edit("Height", Frame_Height)
Call objVars.Edit("length", Frame_length)
Call objVars.Edit("width", Frame_width)
```

Fig. 40: Changing the Assembly's variables to their new values

```
'Update all Links
Call ObjApp.StartCommand(33068) 'Update All Links
Call ObjApp.StartCommand(40018) 'Hide All Reference Planes
Call objDoc.Windows(1).View.Fit 'Fit the View
```

Fig. 41: Updating the Assembly after editing it

From Fig. 40 above it can be seen that the software changes the Solid Edge assembly's variables named height, length, width to the corresponding Visual Basic variable that match up to the users input. Once the Solid Edge assembly has been modified the software:

- Updates the files links
- Hides all the parts reference planes as not to clutter the computer screen with too much information for the user
- Fits the Solid Edge assembly drawing to the computer screen as to give the user the best possible view of the new air-handling unit frame

This is shown in Fig. 41.

Once the air-handling unit frame has been built and created as a 3D model the user may resize it or proceed to Stage 2.

In the event that the user wishes to proceed and does not need to resize the air-handling unit the software is programmed to perform extra calculations and save the results prior to commencing Stage 2. By doing so, after the software moves on to Stage 2 it is unnecessary to repeat the calculations and the user will not be obliged to re-enter the information again.

```
TB_panel_length = Frame_width - 140
ANonA_panel_Heigth = Frame_Heigth - 140
filter_panel_Width = 140
Standard_panel_width = 648
Door_panel_width = 509
Access_panel_Width = 350
panel_ThicKness = 52
InsideComponentSpace = Frame_width - 109
QtyHzfilters = IntuValCombo2.Text()
QtyVtfilters = IntuValCombo3.Text()
HorzTHs = Frame_width - 140
VerTHs = Frame_Heigth - 140
Inside_TH_Space_Access = (Frame_length - 140) * 0.00)
Inside_TH_Space_NonAccess = (Frame_length - 140) * 0.00)
Inside_TH_Space_Top = (Frame_length - 140) * 0.00)
Inside_TH_Space_Bottom = (Frame_length - 140) * 0.00)
Inside_Space_Access = (Frame_length - 140) * 0.00)
Inside_Space_NonAccess = (Frame_length - 140) * 0.00)
Inside_Space_Top = (Frame_length - 140) * 0.00)
Inside_Space_Bottom = (Frame_length - 140) * 0.00)0
```

Fig. 42: Extra Calculations to be used after Stage 1

```
Call objVars.Edit("DoorAcc_Pan", Door_panel_width)
Call objVars.Edit("Pan_Thickness", panel_Thickness)
Call objVars.Edit("StdPan_width", Standard_panel_width)
Call objVars.Edit("TB_Pan_length", TB_panel_length - 3)
Call objVars.Edit("ANonA_Pan_Heigth", ANonA_panel_Heigth - 3)
Call objVars.Edit("ANonA_filter_Pan_1", filter_panel_Width)
Call objVars.Edit("panelfilterTHBottom", BottomfilterTH)
Call objVars.Edit("panelfilterTHTop", TopfilterTH)
Call objVars.Edit("AHUFrameInsidewidth", InsideComponentSpace)
Call objVars.Edit("ANonA_filter_Pan_1", filter_panel_Width - 3)
Call objVars.Edit("StdPan_width", Standard_panel_width - 3)
Call objVars.Edit("DoorAcc_Pan", Door_panel_width - 3)
Call objVars.Edit("Acces_panel_width", Access_panel_Width - 3)
Call objVars.Edit("TB_TH_length", HorzTHs)
Call objVars.Edit("ANonA_TH_length", VerTHs)
```

Fig. 43: Extra Variables being edited during Stage 1

As seen from Fig. 42 and from Fig. 43 during the process of Stage 1 the software saves the results of extra calculations that will be used in Stage 2. It also edits and updates extra variables in the top level assembly so that when the user moves on to Stage 2 the software will already have some information previously stored helping to design the air-handling unit faster.

This next paragraph will illustrate the grounds for performing such calculations.

As previously shown, to calculate the exterior width and height of the air-handling unit frame the user specified information regarding the quantity of air filters to be used.

Stage 2 is the stage at which the software inserts the air filters into the assembly. As the information regarding the size and quantity of air filters was supplied in Stage 1, when Stage 2 arrives the software must not allow the user to re-enter in this information. To achieve this, the software assigns the information to variables that can be saved somewhere within the software that Stage 2 can access them. The only information the user may specify regarding the air filters in Stage 2 will be their location.

The user has the option of resizing the air-handling unit frame after the software has built it. This is accomplished clicking on the "RESIZE" button on the Stage 1 graphical user interface after specifying new design criteria. From the Flow Chart in Fig. 28 it can be seen that the process the software must follow to resize the air-handling unit frame is as follows:

- Recalculate the new figures using the same algorithms and formulae that were originally used to model the frame
- Access the Solid Edge assembly's variable objects
- Apply the new values to the Solid Edge assembly's file variables
- Update the Solid Edge assembly file

The Solid Edge assembly document may be resized an infinite amount of times until the final dimensions are decided. After the correct external dimensions have been achieved the user may proceed onto Stage 2.

Fig. 44 shows an exploded view of the frame assembly after Stage 1.

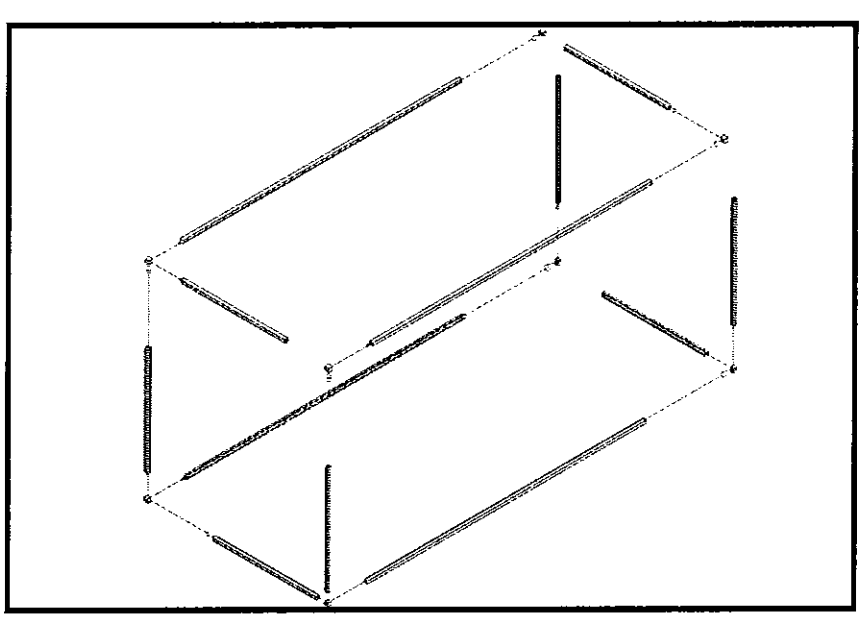


Fig. 44: Exploded view of Assembled Frame

6.1.1 Stage 1; Base frame design and manufacture

After the external frame of the air-handling unit has been designed the designer may add a base frame to the air handling. This is done before the user progresses onto Stage 2.

Fig. 45 illustrates the air-handling unit frame assembly fixed onto a base frame.

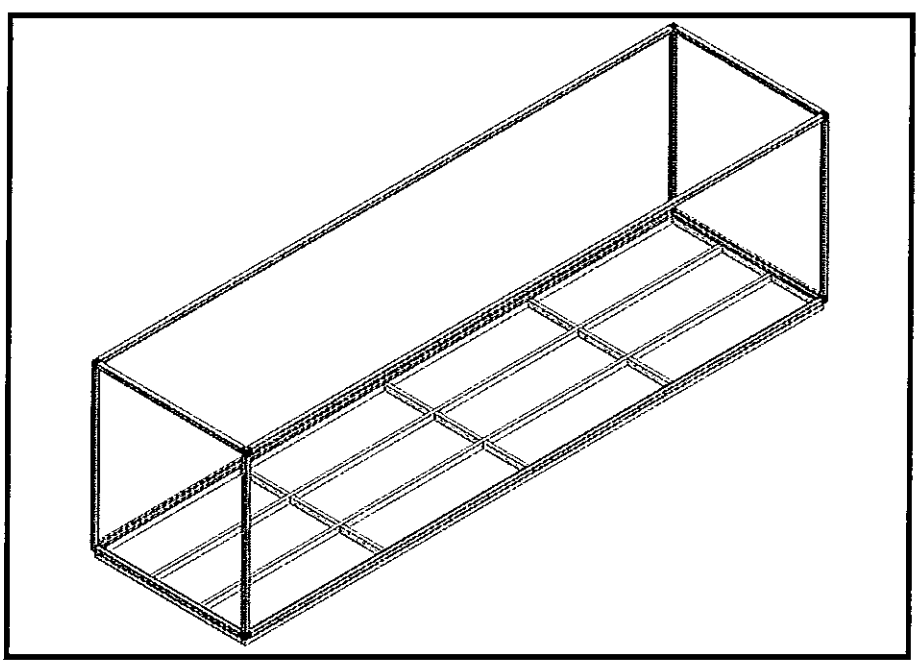


Fig. 45: External Frame Assembly including Base Frame

The base frame that the external aluminium frame sits on is used to support the complete weight of the air-handling unit after it is manufactured. The base frame of the air-handling unit is made up of steel cross-sections welded together in a specific fashion to provide support for the air-handling unit.

Unlike the air-handling unit's external frame the base frame has more than one possible assembly configuration. The two main assembly configurations can be seen in Fig. 46 and Fig. 47. The difference between the two main configurations is whether the internal steel cross sections will run the internal length or the internal width of the air-handling unit.

A graphical user interface was designed to manage the design of the base frame. From the code used to create the base frame, obvious similarities can be seen between it and the code used to build the external frame.

The procedure for manufacturing the base frame can be seen in Fig. 48.

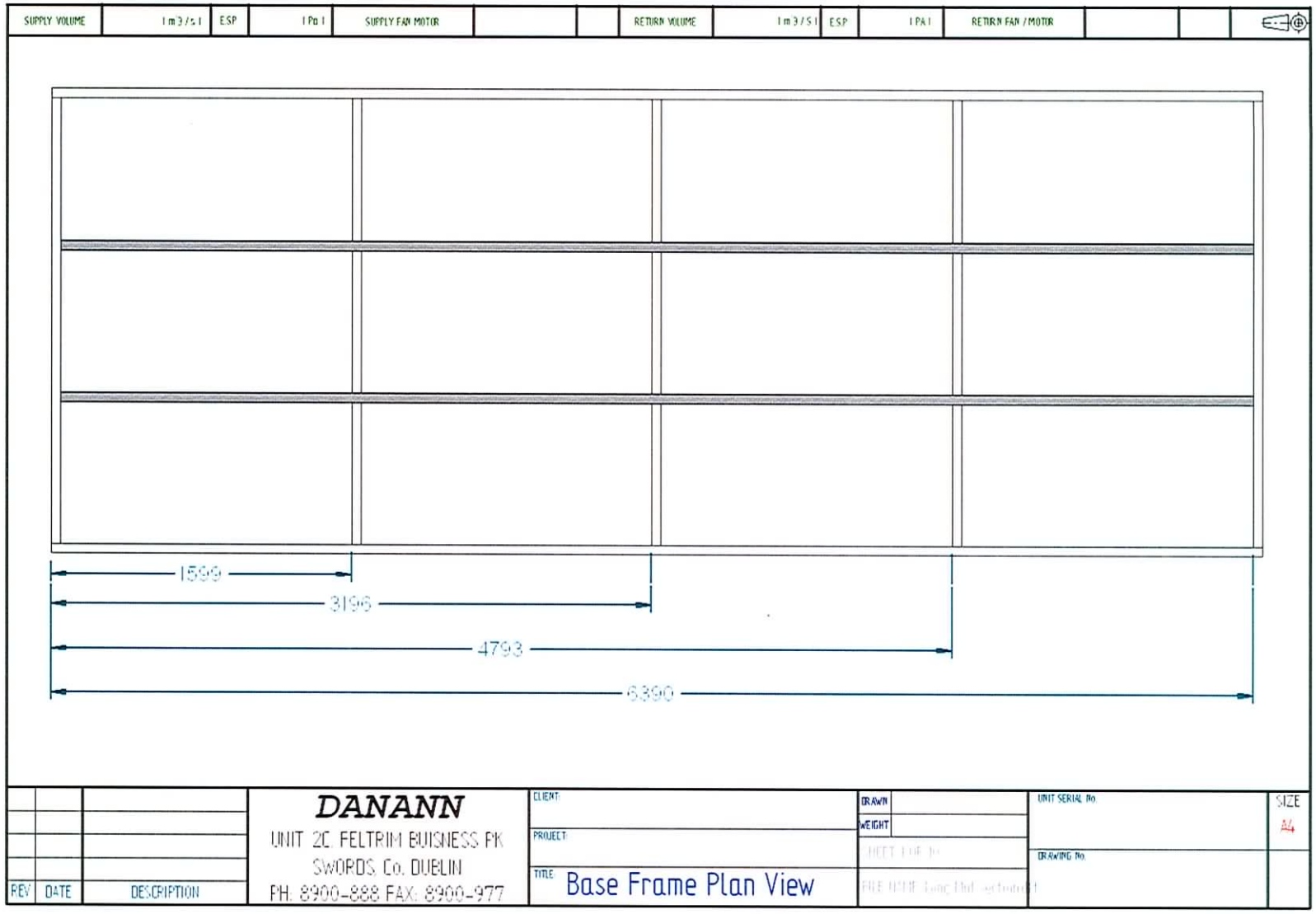


Fig. 46: Base Frame Configuration A

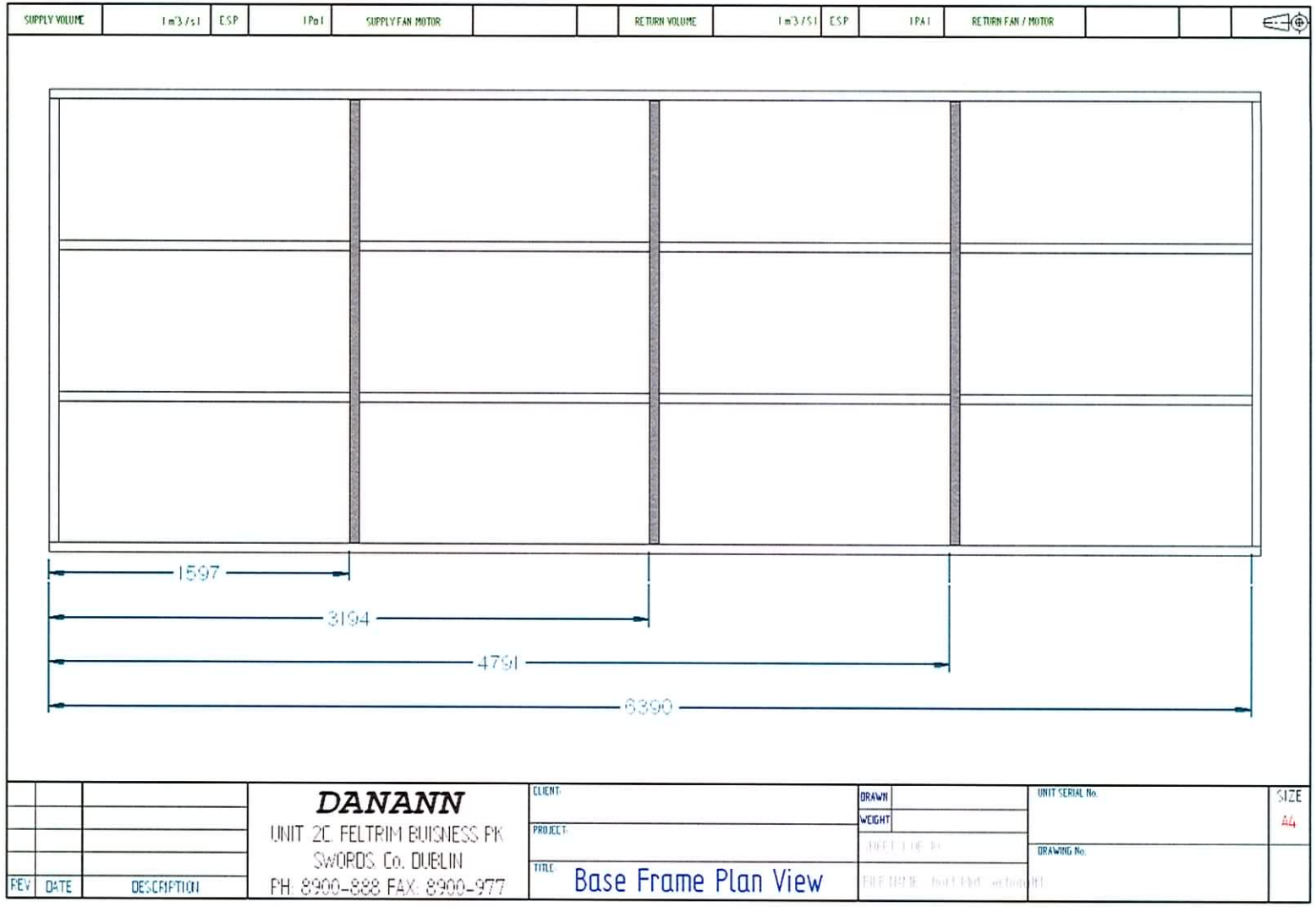


Fig. 47: Base Frame Configuration B

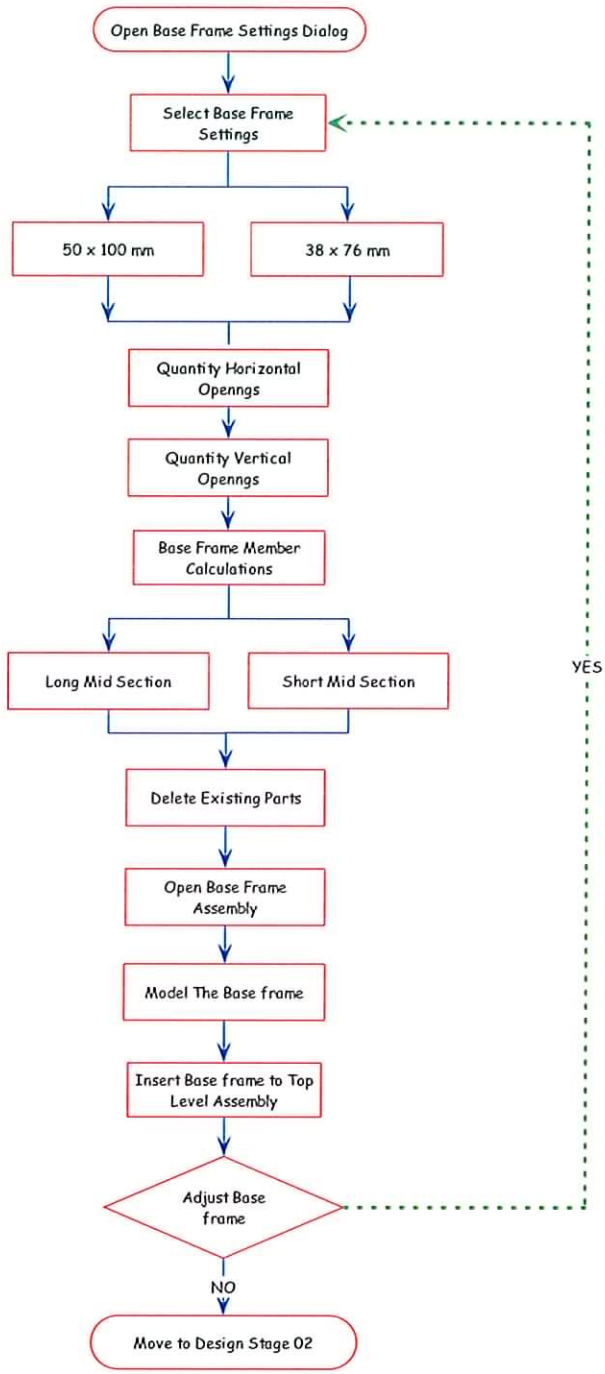


Fig. 48: Base Frame Design Flow Chart

From the flow chart in Fig. 48 the first step in designing the base frame assembly is to decide which style of steel cross-section to use. There are two options:

- 50 mm x 100 mm
- 38 mm x 76 mm

After the correct steel cross-section is chosen the designer must decide on the required format that the base frame must conform to. From Fig. 46 and Fig. 47 the base frame has four horizontal openings and three vertical openings. Five lengths of the steel cross-section are required to make up the steel to run vertically in the frame assembly and four lengths of steel are required to run horizontally. The designer must decide on which configuration to use by checking/un-checking the "long mid-section" check box located on the dialog. The exact arrangement of the steel may be decided by entering the amount of openings that will be seen in the frame after it is manufactured. From the 2D view shown embedded into the dialog four types of steel cross-section are used to manufacture the base frame. They are indicated in the colours, Red, Blue, Green and Yellow in Fig. 49.

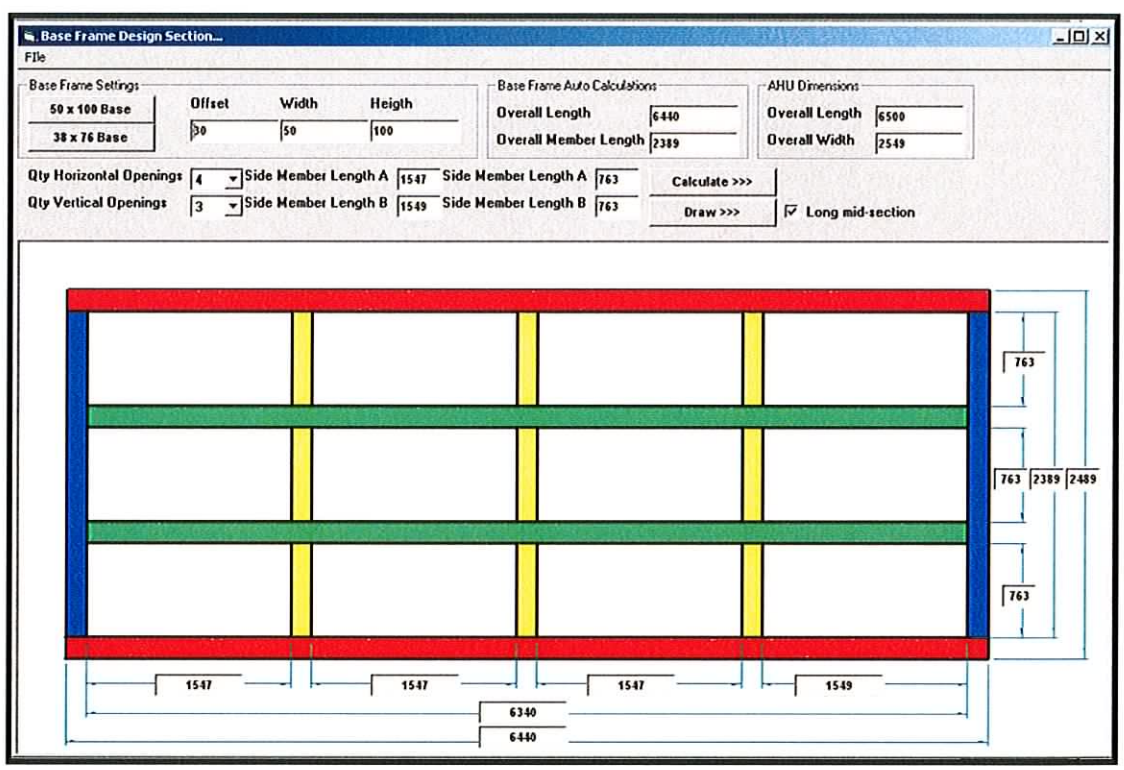


Fig. 49: Air-handling Unit Base Frame Design Dialog

The instance a modification is made to any feature of the base frames design the values for the lengths of steel will be recalculated. By clicking on the button labelled "Draw" the software will automatically model the base frame.

Shown below in Fig. 50 is the code used to calculate the dimensions of the lengths of steel in the base frame.

```
Private Sub Command4_Click()
If ValCombo1.Text() = 0 Or ValCombo2.Text() = 0 Then
MsgBox "The quantity for Horizontal or Vertical openings cannot be blank! 123", vbInformation
Exit Sub
End If
Text1(20).Text = Frame_length
Text1(21).Text = Frame_width
Text1(17).Text = Frame_length - (2 * ValcText1(7).Text())
Text1(5).Text = Frame_width - (2 * ValcText1(7).Text()) - (2 * ValcText1(8).Text())
Text1(2).Text = Int((ValcText1(17).Text() - (2 * ValcText1(8).Text())) - (ValCombo1.Text - 1) * ValcText2(8).Text())) / ValCombo1.Text()
Text1(16).Text = ValcText1(17).Text() - (2 * ValcText1(8).Text()) - (ValcText1(2).Text * (Combo1.Text - 1)) - ValcText1(8).Text *
(Combo1.Text - 1))
Text1(14).Text = Int((ValcText1(5).Text()) - (ValCombo2.Text - 1) * ValcText1(8).Text())) / ValCombo2.Text()
Text1(13).Text = ValcText1(5).Text() - (ValcText1(14).Text * (Combo2.Text - 1)) - ValcText1(8).Text * (Combo2.Text - 1))
Text1(6).Text = Text1(14).Text
Text1(3).Text = Text1(14).Text
Text1(4).Text = Text1(13).Text
Text1(10).Text = Frame_width - (2 * ValcText1(7).Text()) - 2 * ValcText1(8).Text())
Text1(11).Text = Frame_width - (2 * ValcText1(7).Text())
Text1(0).Text = Text1(2).Text
Text1(19).Text = Text1(2).Text
Text1(1).Text = Text1(2).Text
Text1(18).Text = Text1(16).Text
Text1(15).Text = Frame_length - (2 * ValcText1(7).Text()) - 2 * ValcText1(8).Text())
Text1(12).Text = Frame_length - (2 * ValcText1(7).Text())
Text1(28).Text = Text1(14).Text
Text1(30).Text = Text1(14).Text
Text1(29).Text = Text1(13).Text
Text1(27).Text = Frame_width - (2 * ValcText1(7).Text()) - 2 * ValcText1(8).Text())
Text1(26).Text = Frame_width - (2 * ValcText1(7).Text())
Text1(32).Text = Text1(2).Text
Text1(31).Text = Text1(2).Text
Text1(22).Text = Text1(2).Text
Text1(23).Text = Text1(16).Text
Text1(24).Text = Frame_length - (2 * ValcText1(7).Text()) - 2 * ValcText1(8).Text())
Text1(25).Text = Frame_length - (2 * ValcText1(7).Text())
End Sub
```

Fig. 50: Programming Calculations prior to Base frame Assembly

Once the calculations have been performed the software models the base frame assembly for the air-handling unit.

6.2 Stage 2: Interior subassembly design and positioning

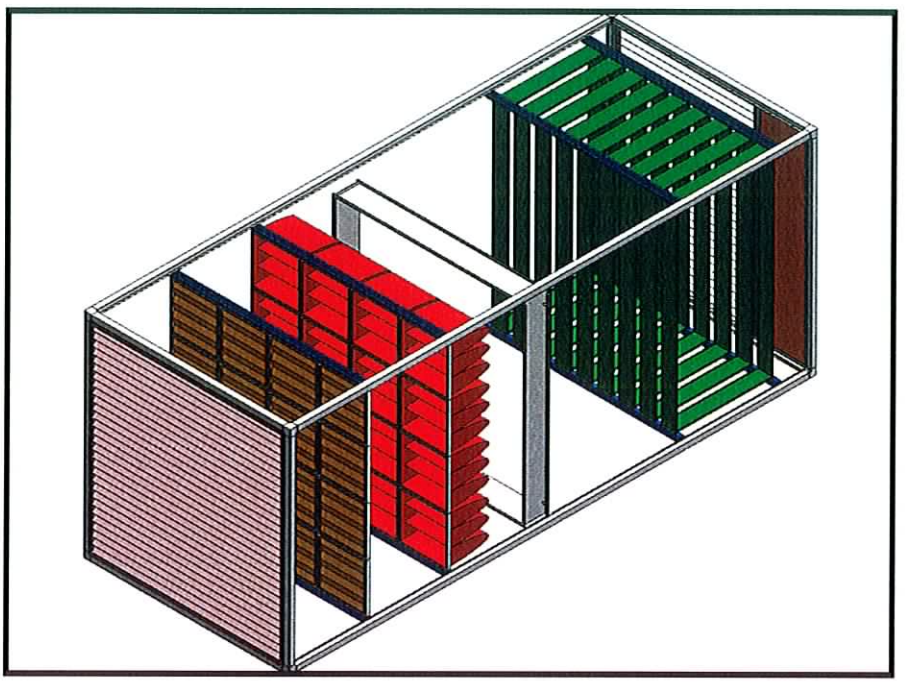


Fig. 51: Frame & interior of air-handling unit

Fig. 51 shows the external frame of a typical single storey air-handling unit. Inside the frame contains a series of different components and sub-assemblies that are found in standard air-handling units. At the front of the air-handling unit a louvre can be seen, followed by panel filters, bag filters, a Cooling coil, attenuators and a damper at the rear. Other components such as Plate Heat Exchangers, Humidifiers, Fans and Motors can also be found in air-handling units.

As mentioned previously, Stage 1 deals with the automated design of the air-handling unit's external frame. Stage 2 deals with the automated design and placement of the subassemblies that make up the internal capacity of an air-handling unit.

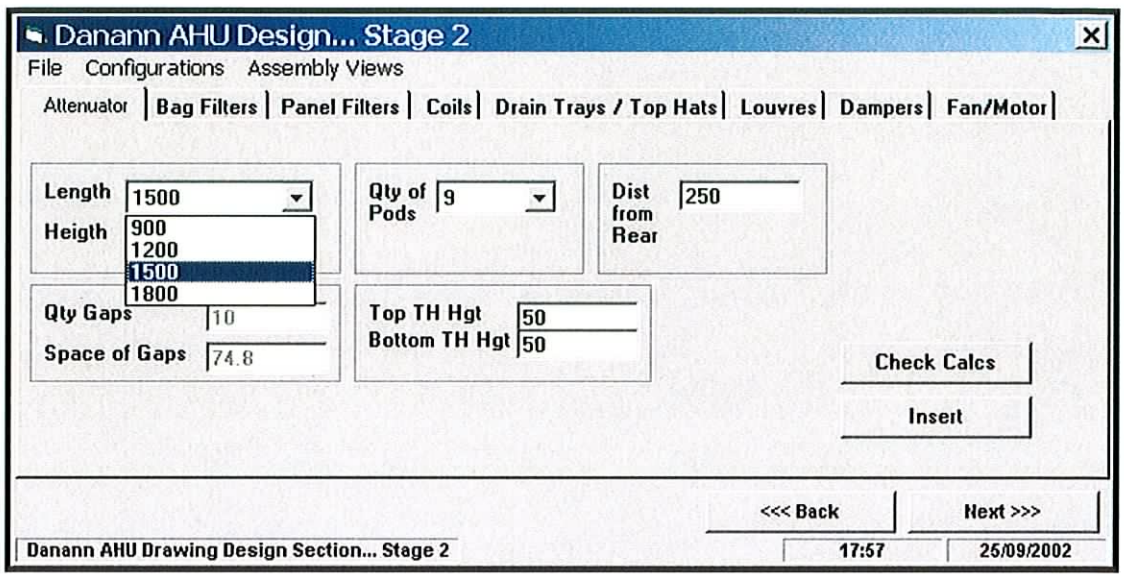


Fig. 52: Stage 2

Fig. 52 depicts the graphical user interface for Stage 2. It illustrates the subassemblies that can be added to the inside of an air-handling unit frame that was designed in Stage 1. They are:

- Attenuators
- Bag filters
- Panel filters
- Coils
- Drain trays/Top Hats
- Louvre's
- Dampers
- Fan/Motor sections

Unlike Stage 1 there is more than one type of air-handling unit subassembly to be designed. The air-handling unit subassemblies in Stage 2 must also be positioned within the air-handling unit's external frame. It was not necessary in Stage 1 to position the air-handling unit frame after it was designed.

It was necessary to design the software for Stage 2 to facilitate inserting any one of a number of sub-assemblies in a random fashion. Fig. 52 demonstrates how this was achieved by appropriately designing the graphical user interface for Stage 2 using objects called "TABS".

A flow chart in Fig. 53 displays the design workflow and automated processes involved in designing and positioning the sub-assemblies that are make up Stage 2. From the flow chart it can be seen that the process involved in designing a sub-assembly for the inside the air-handling unit's frame contains a similar pattern for all sub-assemblies involved.

The first decision that must be made in Stage 2 is to choose an appropriate sub-assembly. With Stage 1 there was no choice, the user had to design the air-handling unit external frame. The subsequent step after choosing the appropriate sub-assembly is to enter its design criteria by using the graphical user interface shown in Fig. 52. Once the design criteria are specified the software can cycle through set routines and procedures performing necessary calculations. It is necessary for the software to perform its routines and procedures before it will model the sub-assembly and place it into the air-handling unit assembly. The flow chart in Fig. 53 clearly describes this process

Once the sub-assembly has been modelled the user has the option to edit it after it has been placed into the air-handling unit. This facility was added in the event of errors being incurred by the user resulting in an inaccurate sub-assembly being inserted into the air-handling unit.

Once all of the required sub-assemblies have been added to the air-handling unit the user then can progress to the final Stage, Stage 3.

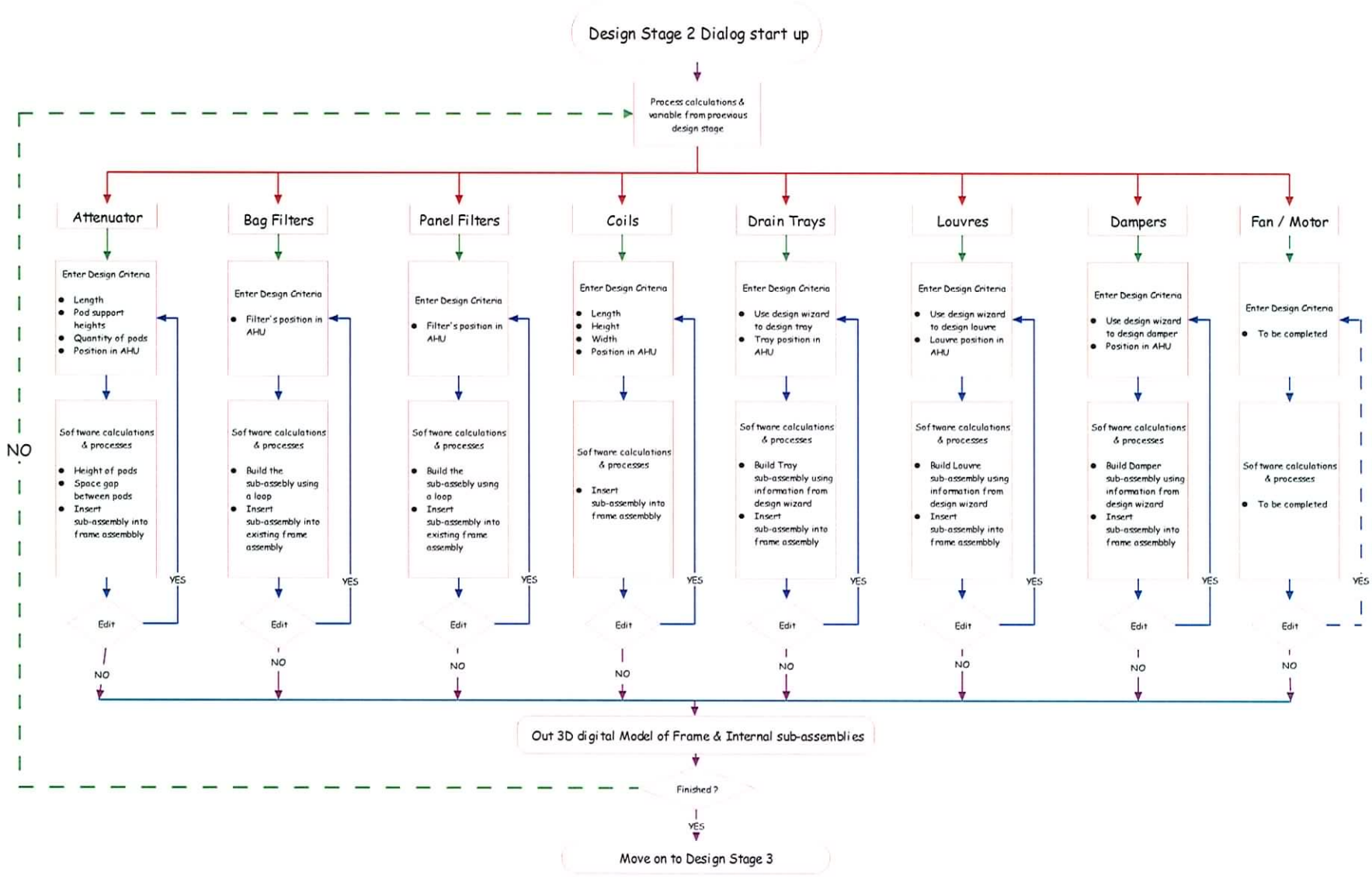


Fig. 53: Stage 2 Flow Chart

Fig. 53 shows the variety of sub-assemblies available to the design engineer. The technical processes for designing and inserting a sub-assembly into an air-handling unit may appear similar for all sub-assemblies in Stage 2.

- Choose a sub-assembly to enter into the air-handling unit.
- Specify key information so that the software can design the air-handling unit sub assembly
- The software cycles through its calculations and design procedures and then it builds and inserts the sub-assembly.

Once the sub-assembly has been inserted into the air-handling unit the user may place another sub-assembly into the air-handling unit or progress to Stage 3.

6.2.1 Attenuators

An attenuator is used at the rear of an air-handling unit to reduce the amount of noise emitted from the air-handling unit as air exits at high velocities.

Fig. 54 shows the graphical user interface from Stage 2. The attenuator section is the active option. An illustration of the attenuator can be seen in Fig. 55.

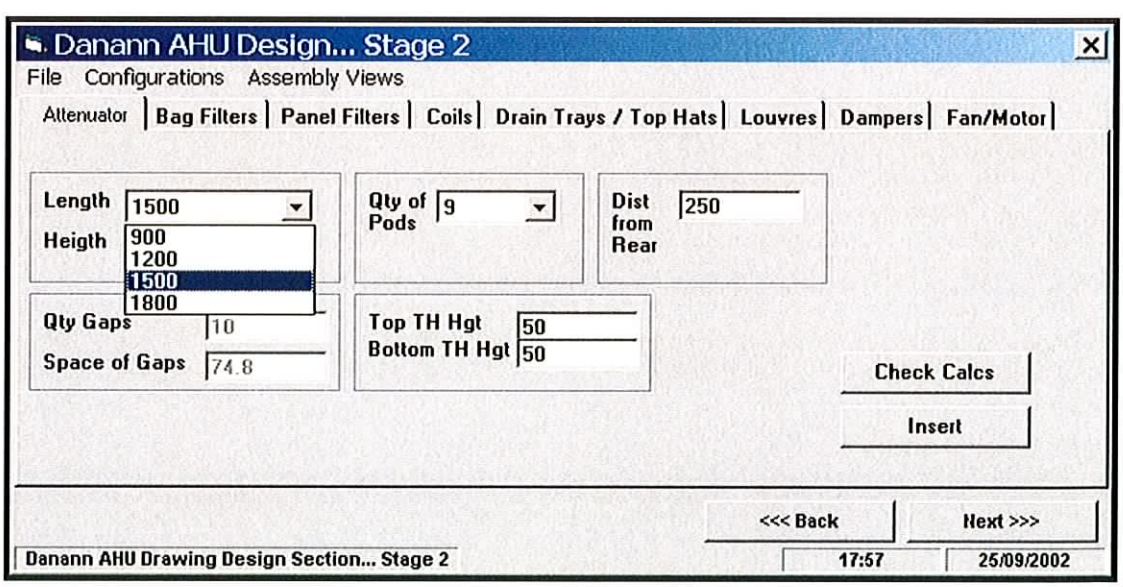


Fig. 54: Attenuators GUI

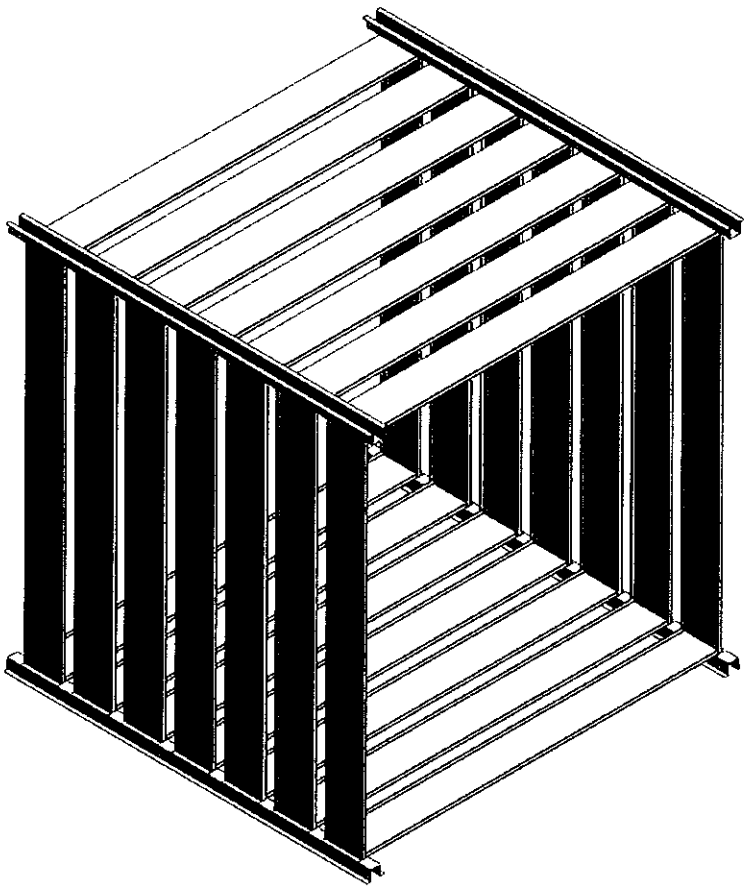


Fig. 55: Attenuator sub-assembly

The first step in specifying an attenuator into an air-handling unit is to supply the necessary information to the graphical user interface. From the information specified the software performs calculations that are required before the sub-assembly will be built. From the graphical user interface in Fig. 54 the design information that is required is:

- Length. There are four standard lengths that the attenuator can be made in. The user must choose one to proceed.

Height. The height of the attenuators is calculated from the exterior dimensions of the air-handling unit's frame and the aluminium cross-section's definition. This value is calculated in Stage 1 and it is held in the software's memory until it is required.

- Quantity of Pods. The quantity of pods that will populate the cross-section of the air-handling unit is important because if there are too many attenuator pods they will be too close together and not operate correctly. If there are too few attenuator pods then they will be spaced too far apart and they will not operate

effectively. When the user enters a value for the "Qty of Pods" the software automatically calculates the space that will be between each pod before they get inserted into the air-handling unit. The space between each pod is calculated from the exterior dimensions of the air-handling unit's frame and the aluminium cross-section's definition. After the space between the pods is calculated, it is displayed for the user on the graphical user interface prior to being inserted into the final assembly. It is displayed for the user so its value can be compared with the value for the optimum gap that should separate each pod in an air-handling unit.

- Distance from the Rear. The attenuators in an air-handling unit are typically situated at the rear of the unit. This input box allows the user to specify how far from the rear the attenuators will be fitted.
- Bottom Top Hat height. The bottom TopHats are fabricated sheet metal fitted across the bottom of the air-handling unit frame to mount the attenuator pods on them. The height of the TopHats must be known so that the heights of the attenuators can be calculated.
- Top Top Hat height. The upper TopHats are fabricated sheet metal fitted across the top of the air-handling unit frame to fix the attenuator pods to them. The height of the TopHats must be known so that the heights of the attenuators can be calculated.

The process for inserting an attenuator can be seen from the flow chart in Fig. 53. Once the design information has been supplied all the calculations can be reviewed by clicking the **"Check Calcs"** button. This will ensure that the values on view on the graphical user interface are reliable and can be compared with their optimum values.

Once the user is satisfied that everything is ready, the user can click on the **"Insert"** button and the software will proceed with creating the solid model of the attenuators. Once the solid model is ready the software will insert it into the air-handling unit's final assembly as described by the flow chart in Fig. 53.

By clicking on the **"Insert"** command button the software begins calculating the process of building the attenuators and inserting them into the air-handling unit.

Firstly the software opens the links to Solid Edge. Once the link has been established the software then opens up and attenuator assembly file and builds the attenuator as

required. Next the software uses a routine to place the required amount of attenuators into the assembly file and also into their correct positions. After the attenuator assembly has been built the file is saved and closed and the software activates the main air-handling unit assembly file. Once the main air-handling unit assembly is activated the software inserts the attenuator assembly file into the main air-handling unit assembly where all the attenuator pods will reside. When the attenuator pods were created they were created into their own file where they were positioned before the file was saved and closed. As the attenuators were positioned correctly in the file they were created in, it is unnecessary for them to be re-positioned after they were inserted into the main air-handling unit assembly.

```
Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_B")
    objOccurrence.Move (Positioning - IndividualSpaceGap) * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) - 0.0191,
(Positioning - IndividualSpaceGap) * 0.001

Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_C")
    objOccurrence.Move (Positioning - IndividualSpaceGap) * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) - 0.0191,
(Frame_Heigth * 0.001) - (Positioning - IndividualSpaceGap) * 0.001

Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_B")
    objOccurrence.Move (Positioning - IndividualSpaceGap) * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) -
(ValComboLength.Text) * 0.001) - 0.053), (Positioning - IndividualSpaceGap) * 0.001

Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_C")
    objOccurrence.Move (Positioning - IndividualSpaceGap) * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) -
(ValComboLength.Text) * 0.001) - 0.053), (Frame_Heigth * 0.001) - (Positioning - IndividualSpaceGap) * 0.001)

For I = 1 To Combo2.Text

  Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_A")
If Frame_A = 1 Then
    objOccurrence.Move Positioning * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) - (ValComboLength.Text) * 0.001),
(0.0545) - (ValText140.Text) * 0.001)
    Positioning = Positioning + PodWidth + IndividualSpaceGap
Else
    objOccurrence.Move Positioning * 0.001, (Frame_Length * 0.001) - (Text163.Text * 0.001) - (ValComboLength.Text) * 0.001),
(0.029) - (ValText140.Text) * 0.001)
    Positioning = Positioning + PodWidth + IndividualSpaceGap
End If
Next I
```

Fig. 56: Attenuator Positioning Code

The code used to position the attenuator pods correctly along side each other can be seen in Fig. 56. This routine contains formulas that are used to automatically add the correct sized attenuator pods to the assembly file as well as position them in the air-handling unit assembly file.

If a change is made to the configuration of the attenuators the user can delete the existing attenuator file from the main air-handling unit assembly and re-insert new attenuator pods from the software's graphical user interface. The software will again open up the attenuator assembly file. On opening the file it deletes its existing parts to

create a blank assembly. Therefore when the software adds more attenuator pods the correct configuration will be present within the file after the software's routine is complete.

Once the attenuators have been correctly placed into the air-handling unit assembly the user may now progress to Stage 3 or insert another sub-assembly from Stage 2.

6.2.2 Bag Filters

Bag filters are used in an air-handling unit to filter out larger particles that can be found in the air as it passes along through the air-handling unit.

Shown below in Fig. 58 is the graphical user interface that is used by the user to build a bag filter section for an air-handling unit assembly.

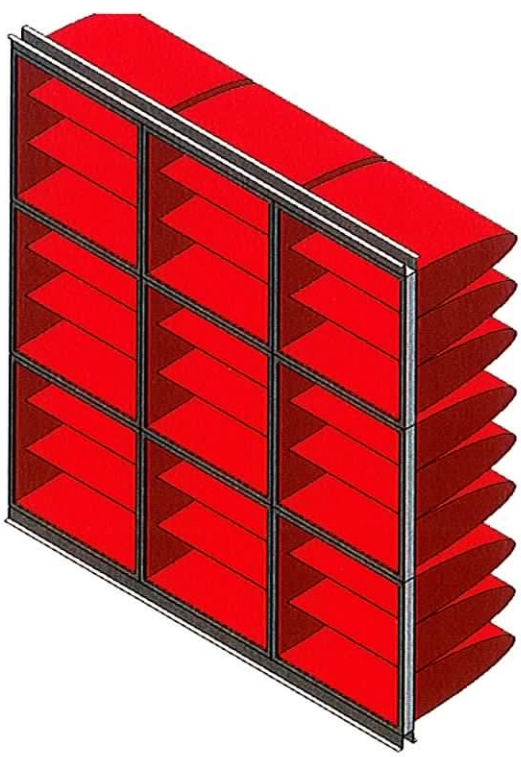


Fig. 57: Bag Filter sub-assembly

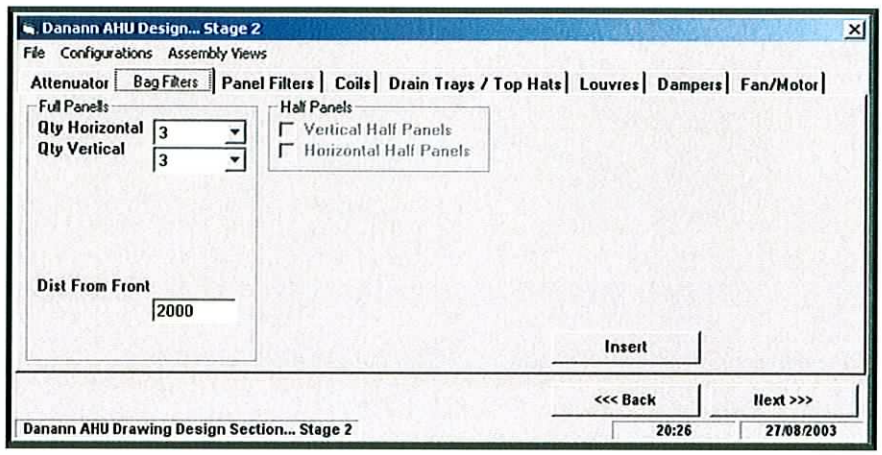


Fig. 58: Bag Filter Graphical User Interface

From Fig. 57 the finished bag filter sub-assembly can be seen. The bag filters, like the attenuators, are built and positioned in an assembly file of their own before they are inserted into the main air-handling unit assembly.

Bag filters are supplied in two sizes:

- 610 mm x 610 mm
- 610 mm x 305 mm

The process for inserting a bag filter sub-assembly into an air-handling unit assembly can be seen from the flow chart in Fig. 53. The user specifies the position that the bag filter section will take in the air-handling unit assembly. In Stage 1 the amount of bag filters to be used in the air-handling unit and the format in which they will be built is specified. In Stage 1 the information the user enters for the width and height of the air-handling unit is used to build the bag filter sub-assembly. The external frame dimensions such as the external width and the external height can be calculated from the bag filter sub-assembly because it is designed to fit flush inside the air-handling unit's external frame.

A section of bag filters in an air-handling unit are designed to completely incorporate the cross-section of the air-handling unit. This way no air will get through the air-handling unit without passing through the filters.

A bag filter sub-assembly is built in X Z matrix form. The amount of filters in the horizontal and vertical directions is determined from the exterior dimensions of the air-handling unit.

Once the software receives the positional information from the user it can automatically build the sub-assembly. To build the sub-assembly the software opens its ActiveX links with Solid Edge and creates a blank bag filter sub-assembly file. Using the information from Stage 1 the software builds the bag filter sub-assembly. Once the sub-assembly has been built it saves and closes the bag filter sub-assembly.

When the bag filter sub-assembly has been created the software opens or reactivates the air-handling unit assembly and inserts the bag filter sub-assembly and positions it accordingly using the positional information specified by the user in Stage 2.

A specific section of code was written to automatically build the bag filter sub-assembly once the user had supplied the correct information in Stage 1. It can be shown below in Fig. 59.

```
If Frame_A = 1 Then
InsideComponentSpace = Frame_Width - 109
   Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_D)
      objOccurrence.Move Positioning, Val(Text1(4").Text) * 0.001, 0.0545
For I = 1 To Val(Combo3.Text)
   Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_A)
   objOccurrence.Move Positioning, Val(Text1(4").Text) * 0.001, 0.0545 + BottomFilterTH * 0.001
   VerPositioning = 0.61
      For Z = 2 To Val(Combo4.Text)
         Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_A)
         objOccurrence.Move Positioning, Val(Text1(4").Text) * 0.001, 0.0545 + BottomFilterTH * 0.001 + VerPositioning
         VerPositioning = VerPositioning + 0.61
      Next Z
If Check4.Value = 1 Then
         Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_B)
         objOccurrence.Move Positioning, Val(Text1(4").Text) * 0.001, 0.0545 + BottomFilterTH * 0.001 + VerPositioning
End If
   Positioning = Positioning + 0.61
Next I
If Check3.Value = 1 Then
VerPositioning = 0.61
For I = 1 To Val(Combo4.Text)
         Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_C)
         objOccurrence.Move Positioning, Val(Text1(4").Text) * 0.001, 0.0545 + BottomFilterTH * 0.001 + (VerPositioning - 0.61)
         VerPositioning = VerPositioning + 0.61
Next I
End If
End If
```

Fig. 59: Nested For Loop

To build the bag filter sub-assembly the programming code situated the individual bag filters in sequence in the X and Z direction. The X routine was used to place one row of bag filters across the bottom of the air-handling unit frame. The Z routine was used to take the first row that was created by the X routine and copy it up in the Z direction to

the ceiling of the air-handling unit, hence completing the bag filter sub-assembly. Because every part is located relative to one another within the sub-assembly it was possible to derive a formula using the dimensions of the bag filters in association with the code to build the bag filters.

6.2.3 Panel Filters

Panel filters are used in an air-handling unit to extract finer particles that can be found in the air as it passes through the air-handling unit.

Fig. 61 shows the graphical user interface that was designed for the user to build a panel filter section for an air-handling unit assembly.

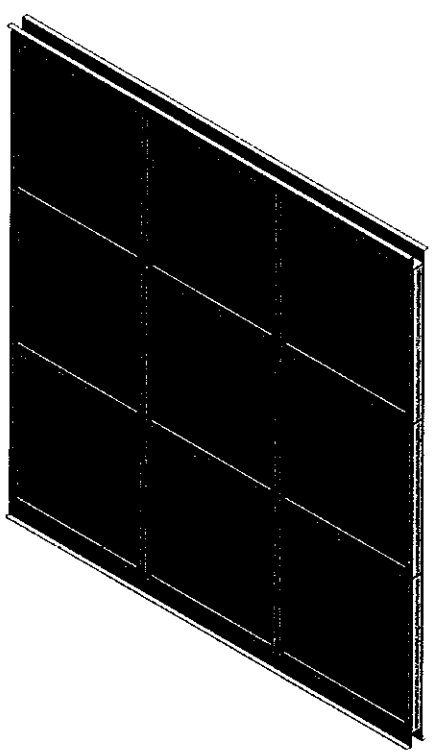


Fig. 60: Panel Filter sub-assembly

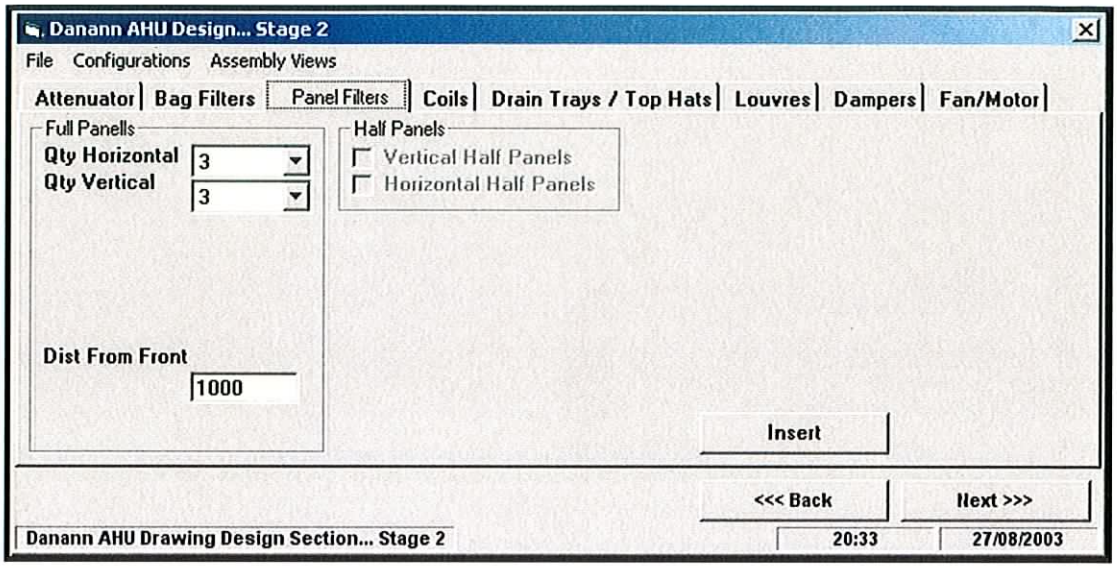


Fig. 61: Panel Filter Graphical User Interface

From Fig. 60 a finished panel filter sub-assembly can be seen before it was inserted into an air-handling unit external frame.

There were many similarities in automating the design of the panel filters and the bag filters. The primary difference being that the software uses a separate set of Solid Edge part and assembly drawings to build and locate the panel filter sub-assembly in the main air-handling unit assembly.

The difference between panel filters and bag filters used in air-handling unit manufacture is the material that is used to filter the air in each style of filter and the amount of material used to do so. The filters themselves have exactly the same dimensions and both styles of filter are assembled and fitted the same for each air-handling unit.

6.2.4 Coils

Coils are used to heat or cool air as it passes through the air-handling unit.

Fig. 62 shows the graphical user interface from Stage 2 that is used to insert coils.

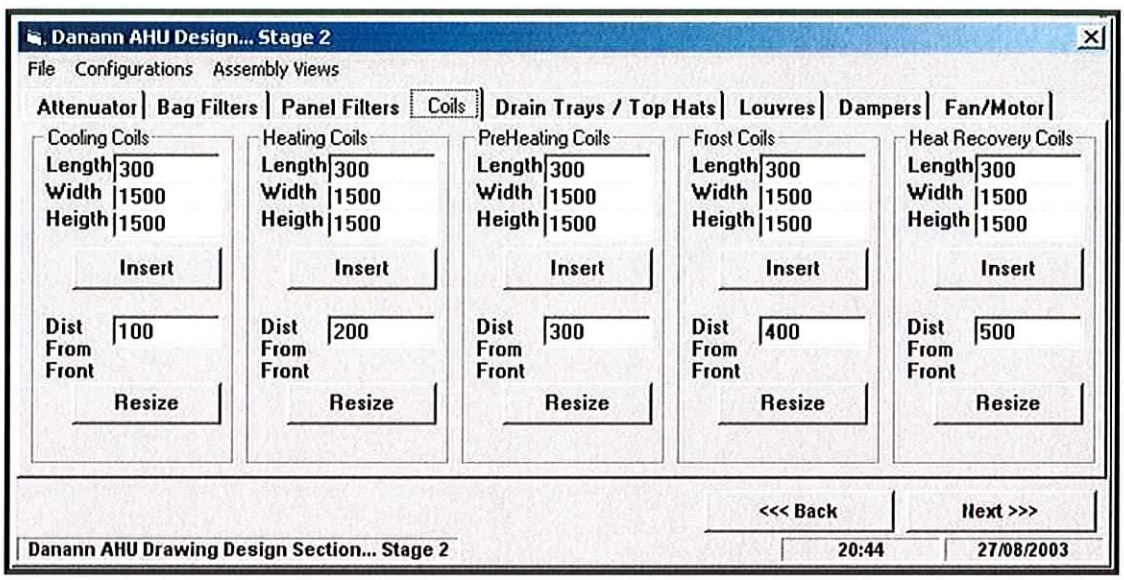


Fig. 62: Coil Graphical User Interface

From Fig. 62 the graphical user interface illustrates that there are five different types of coil that can be inserted into an air-handling unit.

- Cooling coils
- Heating coils
- Preheating coils
- Frost coils
- Heat recovery coils

It is not necessary to represent the coil with the same level of detail as the other sub-assemblies in the air-handling unit solid model. The important criteria for the coils is that when they are inserted into the air-handling unit assembly that their overall dimensions are correct and that they can visually be recognised as a coil in the drawings created for manufacture.

A coil is a component of the air-handling unit that is manufactured by a supplier of the air-handling unit manufacturer. It is more desirable for every coil to be inserted into the air-handling unit assembly as a single part rather than as a sub-assembly.

Fig. 62 shows that the software requires general information to be specified to insert a coil into the air-handling unit assembly.

- Overall length of the coil
- Overall width of the coil
- Overall height of the coil
- Position within the air-handling unit

The parameters for all five different types of coil that are used in their air-handling units are the same.

Once the design of the coils has been specified the **"Insert"** command button is provided to insert the coil into the air-handling unit assembly.

The software accesses the air-handling unit assembly via its ActiveX links with Solid Edge and inserts the coil. Once the coil has been inserted into the air-handling unit assembly the software updates the coils dimensions and positions it as per the information entered for Stage 2.

An algorithm was written for the coils section in the event that the dimensions of the coil had to be modified after it was inserted in to the air-handling unit. The algorithm was designed so the coils could be resized from the coils section on the graphical user interface. This facility would eliminate the process of deleting the coil first from the air-handling unit assembly and reinserting it later with new dimensions.

By specifying new dimensions on the graphical user interface and clicking on the **"Resize"** button the size of the coil would be changed and updated in the air-handling unit assembly file.

```
Public Sub Command4_Click()

  TESTFILE_A = App.Path & "\Drawings\Coils\Cooling_Coil.par"

' Create/get the application with specific settings
On Error Resume Next
Set ObjApp = GetObject(, "SolidEdge.Application")
If Err Then
Err.Clear
```

```
    MsgBox "The Correct File must be Open at this stage", vbInformation
    Else
    ObjApp.Visible = True
    Set objDoc = ObjApp.ActiveDocument
    End If
' adding a part document from a file to the assembly document
    Set objOccurrences = objDoc.Occurrences
    Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName:="TESTFILE_A)
If Frame_A = 1 Then
      objOccurrence.Move 0.0545, ValidText1(21).Text() * 0.001, 0.052
Else
      objOccurrence.Move 0.029, ValidText1(21).Text() * 0.001, 0.027
End If
'Access the Variables Collection
    Set objVars = objDoc.Variables

    For Z = 1 To objOccurrences.Count
    objOccurrences.Item(Z).Activate = True
    Next
'Change the Variables to their new Value
    Call objVars.Edit("CoolingCoil_Length", ValidText1(6).Text())
    Call objVars.Edit("CoolingCoil_Width", ValidText1(7).Text())
    Call objVars.Edit("CoolingCoil_Heigth", ValidText1(8).Text())
'Update all Links
    Call ObjApp.StartCommand(330681)'Update All Links
    Call ObjApp.StartCommand(40018)'Hide All Reference Planes
    Call objDoc.Windows(1).View.Fit 'Fit the View

    'USER DISPLAY

    'Release objects
    Set ObjApp = Nothing
    Set objDoc = Nothing
    Set ObjDocs = Nothing
    Set objOccurrences = Nothing
    Set objOccurrence = Nothing
    Set objVars = Nothing
    Set objVarList = Nothing
    Me.Show

End Sub
```

Fig. 63: Code for inserting a Cooling Coil

Fig. 63 shows a section of the code that was written to insert a cooling coil into an air-handling unit assembly. The second line of the code makes reference to a specific file on the computers hard drive where the Solid Edge drawing for the cooling coil is located.

Once the coil is directly inserted in to the air-handling unit assembly the code moves the coil into position. See Fig. 64.

```
If Frame_A = 1 Then
      objOccurrence.Move 0.0545, ValidText1(21).Text() * 0.001, 0.052
Else
      objOccurrence.Move 0.029, ValidText1(21).Text() * 0.001, 0.027
End If
```

Fig. 64:

After the coil has been positioned the code was then designed to size the coil to match the properties that were specified by the user. See Fig. 65.

```
'Change the Variables to their new Value
Call objVars.Edit("CoolingCoil_Length", ValiText1(6).Text)
Call objVars.Edit("CoolingCoil_Width", ValiText1(7).Text)
Call objVars.Edit("CoolingCoil_Heigth", ValiText1(8).Text)
```

Fig. 65:

6.2.5 Drain Trays/TopHats

Drain trays/TopHats are fitted between coils to accumulate condensate water after air becomes saturated while it passes through a coil and transforms to water.

Fig. 67 displays the graphical user interface that was created to enable the user to build a drain tray for an air-handling unit.

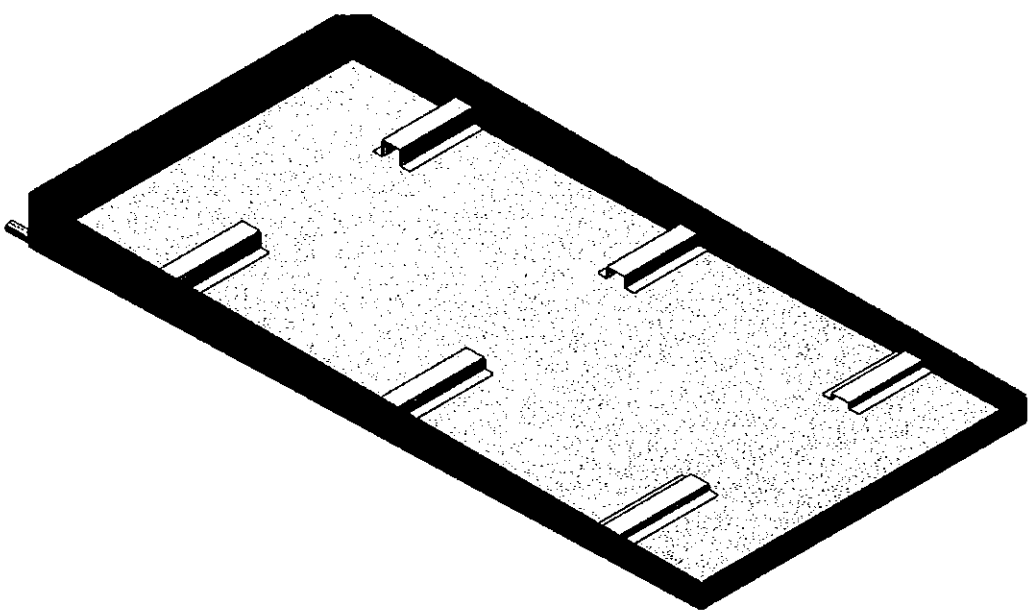


Fig. 66: Drain Tray sub-assembly

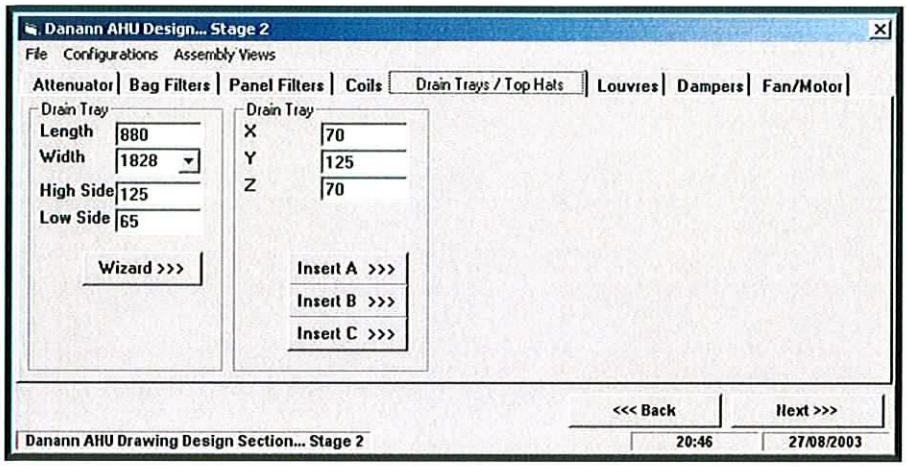


Fig. 67: Drain Tray Graphical User Interface

To design a drain tray the user must enter in basic information about it such as:

- Length. The overall length of the drain tray.
- Width. There are a variety of widths that the user can choose from by way of a drop down list located on the graphical user interface.
- XYZ co-ordinates. The XYZ co-ordinates are used to position the drain tray with the air-handling unit assembly.
- Height. The height of the drain tray at either side must be supplied as the drain tray slopes from one side to the other.

From Fig. 66 a typically finished Style C drain tray can be seen. The methodology for inserting a drain tray into an air-handling unit assembly can be seen from the flow chart in Fig. 53. A more in-depth procedure is shown below.

The procedure for inserting a drain tray sub-assembly is to specify the necessary design criteria into the graphical user interface and use either of the **"Insert A >>>"**, **"Insert B >>>"** or **"Insert C >>>"** command buttons. An alternative graphical user interface was designed and custom written to act as a **"drain tray Wizard"**. The drain tray wizard was designed so that more in-depth design information can be specified and used to help produce better drain tray models for the air-handling unit assembly. Located on the drain tray graphical user interface is a button labelled **"Wizard"**. By choosing the button labelled **"Wizard"** the button opens up another dialog called **"drain tray Wizard"**. The drain tray wizard was created to accommodate the variety of styles of drain tray that may be used in an air-handling unit.

See Fig. 68 below for an image representing the drain tray wizard.

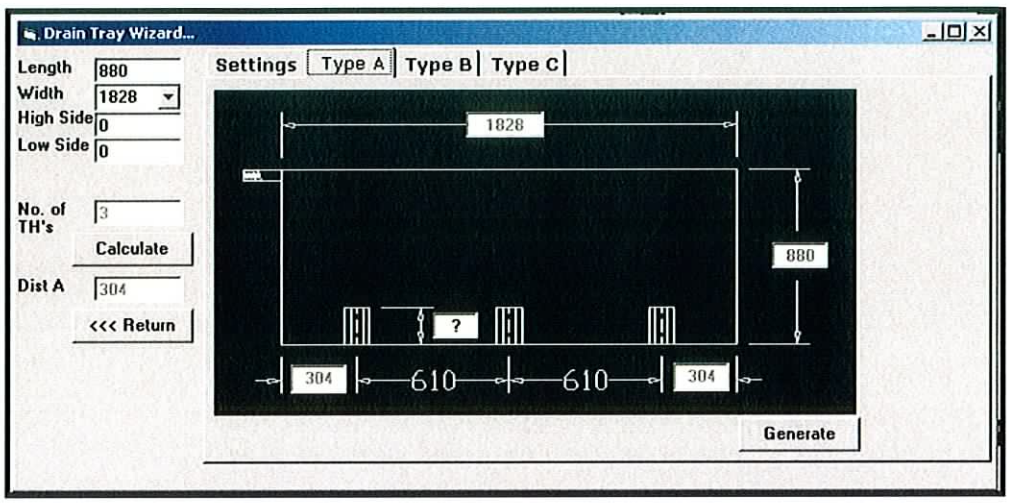


Fig. 68: Drain Tray Wizard

The drain tray wizard provides the user with a structured graphical user interface that requires specific information in a graphical nature. The graphical user interface was designed using two-dimensional images of the drain trays. Using these images the user can clearly make reference to the information the graphical user interface requires to design the drain tray and understand it as well.

The base of the drain tray is inclined as close to 2.5^0 as possible so that it can drain condensate water from itself. Inside the base of the drain tray there are top hats fitted so that a coil can be mounted on the drain tray. Due to the inclined base of a drain tray the TopHats are manufactured at different heights so that their upper surfaces are all in alignment. By fitting the TopHats correctly the coil can be mounted level on the drain. The top hats arrangement within the drain tray can be seen from

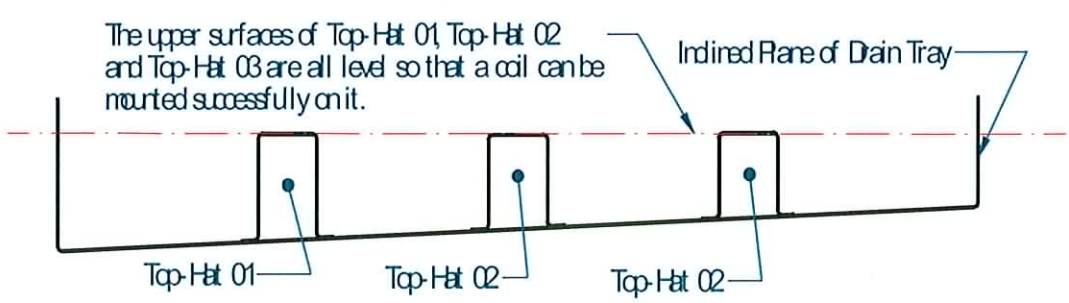


Fig. 69: Side view of TopHats fitted on a Drain Tray with a 2.5^0 gradient

After analysing the rules involved in designing a drain tray it was possible to write an algorithm to automate the process. From the specified information it is possible for the software to calculate the remaining information necessary to design the drain tray prior to accessing Solid Edge.

For the user to create a drain tray with the drain tray wizard it must click on a button labelled **"Generate"**. Once the drain tray has been created the user can return to the Stage 2 graphical user interface and insert the drain tray into the air-handling unit assembly. The user is presented with a choice of three styles of drain tray that can be inserted into an air-handling unit. By clicking on the **"Generate"** button, the software accesses Solid Edge and creates a drain tray according to the style chosen. The first step in building the chosen drain tray is to open a blank drain tray file. The software builds the drain tray from the users specifications. Once the drain tray has been built the software will save and close the drain tray sub-assembly file.

After the software saves and closes the drain tray assembly file it returns to the drain tray wizard dialog where a second and third style drain tray can be designed and built in Solid Edge. These drain tray assemblies will be saved separately to the first drain tray in assembly files of their own. It is possible to return straight to the Stage 2 graphical user interface after the software saves and closes the first drain tray assembly file. From the Stage 2 graphical user interface the drain tray can be inserted into the air-handling unit assembly directly, with out generating anymore drain trays.

6.2.6 Louvres

To facilitate the use of an air-handling unit outdoors it is necessary to include louvre(s) in its design. A louver gives protection to the air-handling unit system by preventing bodies such as twigs and leaves gaining entry to the air-handling system.

Fig. 71 shows the graphical user interface that is used to insert a louvre into an air-handling unit assembly. Form Fig. 70 a complete louvre with a right hand side panel can be seen after it was inserted into an air-handling unit external frame after using the louvre graphical user interface.

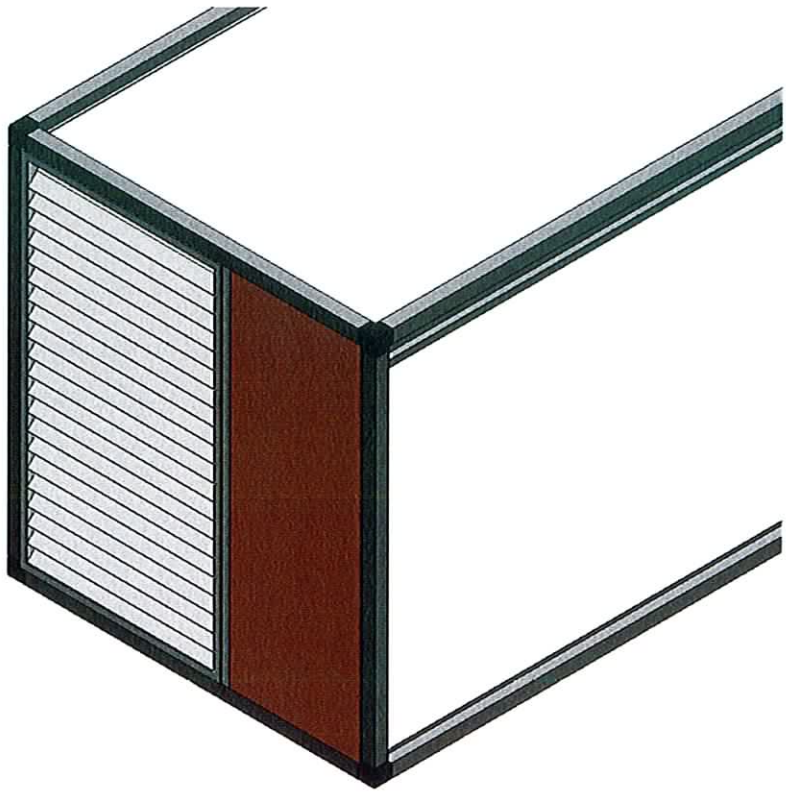


Fig. 70: Louvre sub-assembly

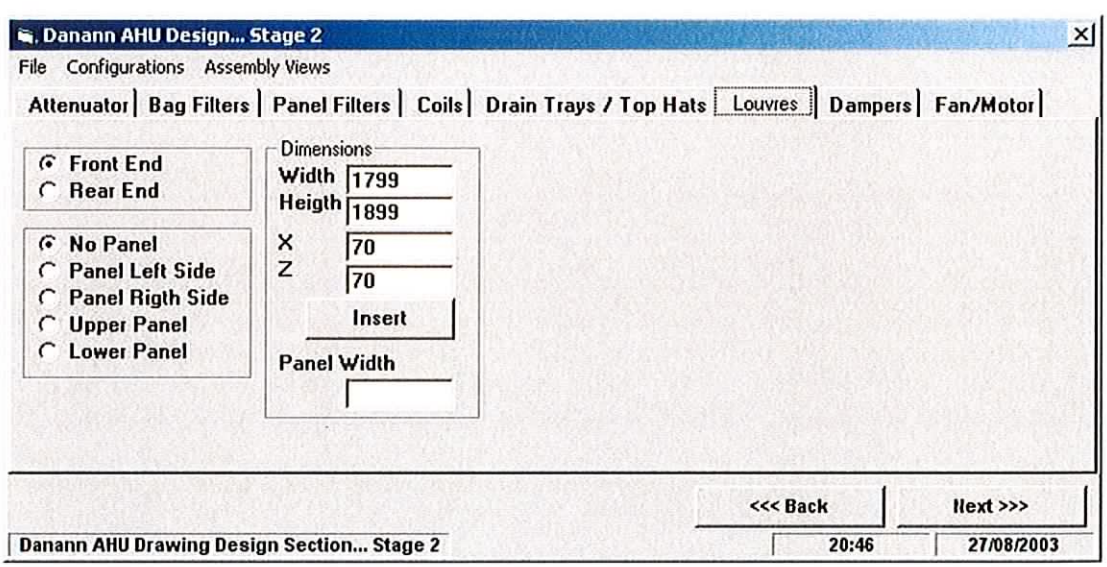


Fig. 71: Louvre Graphical User Interface

There are many features that must be considered when designing a louvre

- Is the louvre to be installed to the Front and/or Rear of the air-handling unit
- Must the louvre span the complete width and height of the air-handling unit

- Is an insulated panel required to the left or right of the louvre
- Is an insulated panel required above or below the louvre

When an insulated panel is required, implying that the louvre will not span the complete width or height of the air-handling unit, the graphical user interface has the capability to supply the software with the width that the inserted panel must conform to. Once the user supplies the width of the insulated panel the software will be able to automatically calculate the width of the louvre from the dimensions of the existing air-handling unit external frame and the width of the insulated panel.

In most cases the louver is not required to span the complete width and height of the front or rear of the air-handling unit.

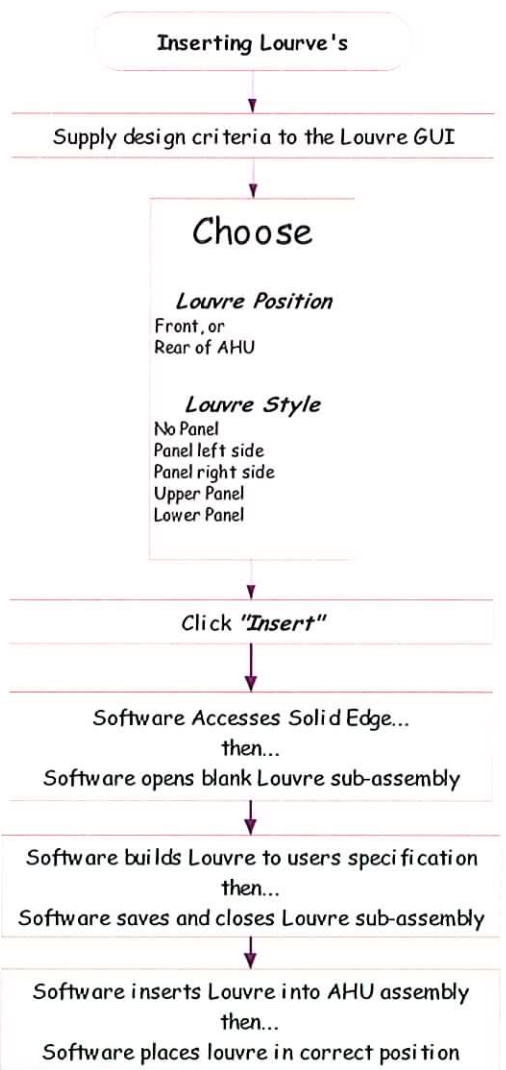


Fig. 72: Louvre sub-assembly design

The processes and procedures for inserting a louvre into an air-handling unit can be seen from Fig. 72. Firstly, design information must be supplied to the graphical user interface.

Louvre Position

- Front, or
- Rear

Louvre Style

- No panel
- Panel left side
- Panel right side
- Upper panel
- Lower panel

If an insulated panel must be included in the design of the louvre, its width must be specified in the text box labelled **"panel width"**. After the design information has been provided the user can click on the command button labelled **"Insert"** for the software to insert the louvre.

After the user clicks on the button labelled **"Insert"** the software begins to calculate the design of the louvre sub-assembly and insert it in to the air-handling unit main assembly.

Firstly the software accesses Solid Edge via its ActiveX links and opens up a blank louvre sub-assembly file. From the variety of options available to the user regarding the style of louvre sub-assembly the software is able to determine the parts and sub-assemblies required.

The software then inserts the necessary parts and sub-assemblies into the blank louvre sub-assembly file. After assembling the necessary parts and sub-assembly's in the correct fashion the software adjusts their dimensions to match the requirements of the air-handling unit assembly. The software then saves and closes the louvre sub-assembly file and reactivates the air-handling unit assembly file.

Upon activation of the air-handling unit assembly file the software would insert the louvre sub-assembly and position it as required.

```
If Option1(0).Value = True Then
   Set objDoc = ThisApp.Documents.Open(App.Path & "\Drawings\Louvres\SupplyLouvres.asm")
TESTFILE = App.Path & "\Drawings\Louvres\SupplyLouvre1.par"
Else
   Set objDoc = ThisApp.Documents.Open(App.Path & "\Drawings\Louvres\ReturnLouvres.asm")
TESTFILE = App.Path & "\Drawings\Louvres\ReturnLouvre1.par"
End If
Set objOccurrences = objDoc.Occurrences
If objOccurrences.Count > 0 Then
   For I = 1 To objOccurrences.Count
      objOccurrences(I).Activate = True
   Next
End If
For I = 1 To objOccurrences.Count
   objOccurrences(I).Delete
Next
Set objOccurrence = objOccurrences.AddByFilename(OccurrenceFileName, TESTFILE)
'From No Panel
      If Option1(0).Value = True And Option2.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, -0.006, ValText1(31).Text * 0.001
      End If
'From Left Panel
      If Option1(0).Value = True And Option3.Value = True Then
         objOccurrence.Move (ValText1(29).Text * 0.001) + ValText2.Text * 0.001) + 0.04, -0.006, ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreVerPanel1.asm")
         objOccurrence.Move (ValText1(29).Text * 0.001) + 0.0415, 0, ValText1(31).Text * 0.001 + 0.0415
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreTH-Large.par")
         objOccurrence.Move (ValText1(29).Text * 0.001) + (ValText2.Text * 0.001), 0, ValText1(31).Text * 0.001
      End If
'From Right Panel
      If Option1(0).Value = True And Option4.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, -0.006, ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreVerPanel1.asm")
         objOccurrence.Move (ValText1(29).Text * 0.001) + 0.0415 + (ValText2(2).Text * 0.001), 0, ValText1(31).Text * 0.001 +
0.0015
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreTH-Large.par")
         objOccurrence.Move (ValText1(29).Text * 0.001) + (ValText2(2).Text * 0.001), 0, ValText1(31).Text * 0.001
      End If
'From Top Panel
      If Option1(0).Value = True And Option5.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, -0.006, ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreHzPanel1.asm")
         objOccurrence.Move (ValText1(29).Text * 0.001), 0, ValText1(31).Text * 0.001) + 0.04 + ValText1(28).Text * 0.001)
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreTH-LargeHz.par")
         objOccurrence.Move (ValText1(29).Text * 0.001), 0, ValText1(31).Text * 0.001) + ValText1(28).Text * 0.001)
      End If
'From Bottom Panel
      If Option1(0).Value = True And Option6.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, -0.006, ValText1(31).Text * 0.001 + ValText2.Text * 0.001) + 0.04
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreHzPanel1.asm")
         objOccurrence.Move (ValText1(29).Text * 0.001), 0, ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\SupplyLouvreTH-LargeHz.par")
         objOccurrence.Move (ValText1(29).Text * 0.001), 0, ValText1(31).Text * 0.001 + ValText2.Text * 0.001)
      End If
'Rear No Panel
      If Option1(1).Value = True And Option2.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, Frame_Length * 0.001 - 0.006, ValText1(31).Text * 0.001
      End If
'Rear Left Panel
      If Option1(1).Value = True And Option3.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001, Frame_Length * 0.001 - 0.006, ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreVerPanel1.asm")
         objOccurrence.Move (ValText1(29).Text * 0.001) + (ValText1(27).Text * 0.001) + 0.0415, Frame_Length * 0.001,
ValText1(31).Text - 1.5) * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreTH-Large.par")
         objOccurrence.Move (ValText1(29).Text * 0.001) + (ValText1(27).Text * 0.001), Frame_Length * 0.001, ValText1(31).Text *
0.001
      End If
'Rear Right Panel
      If Option1(1).Value = True And Option4.Value = True Then
         objOccurrence.Move ValText1(29).Text * 0.001 + (ValText2.Text - 40) * 0.001), Frame_Length * 0.001 - 0.006,
ValText1(31).Text * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreVerPanel_1.asm")
         objOccurrence.Move (ValText1(29).Text - 1.5) * 0.001), Frame_Length * 0.001, ValText1(31).Text - 1.5) * 0.001
         Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreTH-Large.par")
```

```
        objOccurrence.Move (ValTextH(29).Text * 0.001) - (ValText2.Text * 0.001), Frame_Length * 0.001, ValTextH(31).Text *
0.001
    End If
'Rear Top Panel
    If Option1(1).Value = True And Option5.Value = True Then
        objOccurrence.Move ValTextH(29).Text * 0.001, Frame_Length * 0.001 - 0.006, ValTextH(31).Text * 0.001
        Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreHzPanel_1.asm")
        objOccurrence.Move (ValTextH(29).Text * 0.001), Frame_Length * 0.001, ValTextH(31).Text * 0.001 - 0.04 -
ValTextH(28).Text * 0.001)
        Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreTH-LargeHz.par")
        objOccurrence.Move (ValTextH(29).Text * 0.001), Frame_Length * 0.001, ValTextH(31).Text * 0.001 - ValTextH(28).Text *
0.001)
    End If
'Rear Bottom Panel
    If Option1(1).Value = True And Option6.Value = True Then
        objOccurrence.Move ValTextH(29).Text * 0.001, Frame_Length * 0.001 - 0.006, ValTextH(31).Text * 0.001 - 0.04 -
ValText2.Text * 0.001)
        Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreHzPanel_1.asm")
        objOccurrence.Move (ValTextH(29).Text * 0.001), Frame_Length * 0.001, ValTextH(31).Text * 0.001
        Set objOccurrence = objOccurrences.AddByFilename(App.Path & "\Drawings\Louvres\ReturnLouvreTH-LargeHz.par")
        objOccurrence.Move (ValTextH(29).Text * 0.001), Frame_Length * 0.001, ValTextH(31).Text * 0.001 - ValText2.Text * 0.001)
If objDoc.Occurrences.Count <> 0 Then
For Z = 1 To objOccurrences.Count
  objOccurrences.Item(Z).Activate = True
Call objDoc.Save
Call objDoc.Close
```

Fig. 73: Louvre Design programming code

The code displayed above in Fig. 73 represents the section of code that was written to analyse the user's specification for the louvre.

As the louvre will be positioned at the front or rear of the air-handling unit assembly an algorithm was written for the software to automatically calculate where exactly the louvre sub-assembly will be positioned. Therefore reducing the amount information the user must specify.

If an error is made and the louvre is required to be edited, it can be easily deleted from the air-handling unit assembly. After the louvre sub-assembly is deleted the user can begin once again with the louvre graphical user interface and re-insert another louvre. The software will not open a blank louvre sub-assembly file; it will open the existing one and delete all its existing parts. By deleting the parts in the assembly file before it rebuilds the louvre the correct configuration and number of parts will be present in the louvre assembly after the software has finished its modifications.

6.2.7 Dampers

Dampers are used on air-handling units to give the air-handling system the ability to regulate the amount of air intake into the air-handling system. Dampers can be fitted to the air-handling unit on the front, the rear, and the side or on the roof of the unit.

Shown below in Fig. 75 is the graphical user interface for inserting dampers into an air-handling unit assembly. Shown in Fig. 74 is a complete damper with a left hand side panel. This damper was inserted into an air-handling unit external frame using the damper graphical user interface.

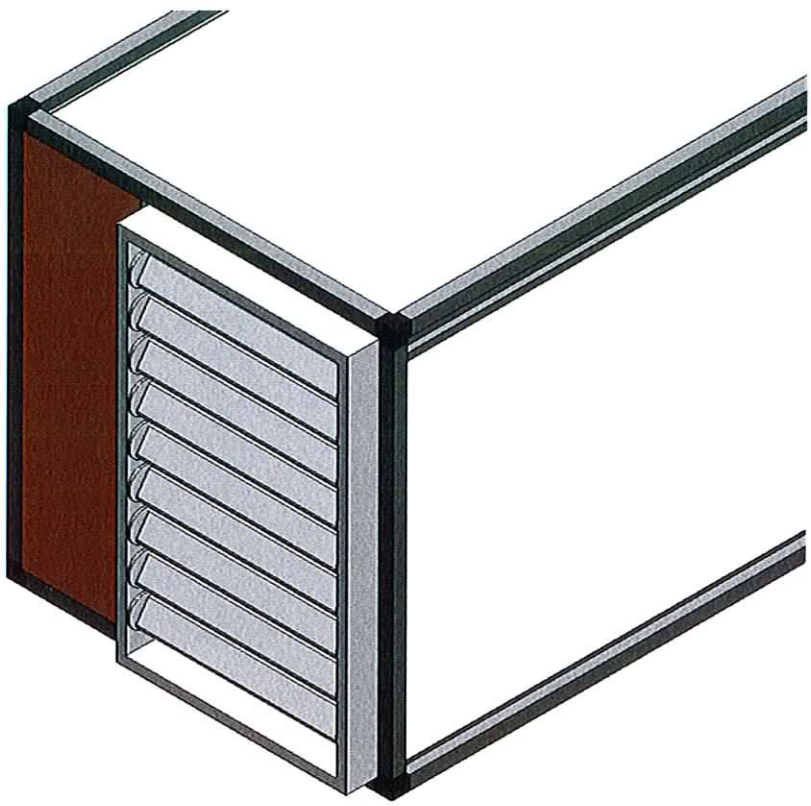


Fig. 74: Damper sub-assembly

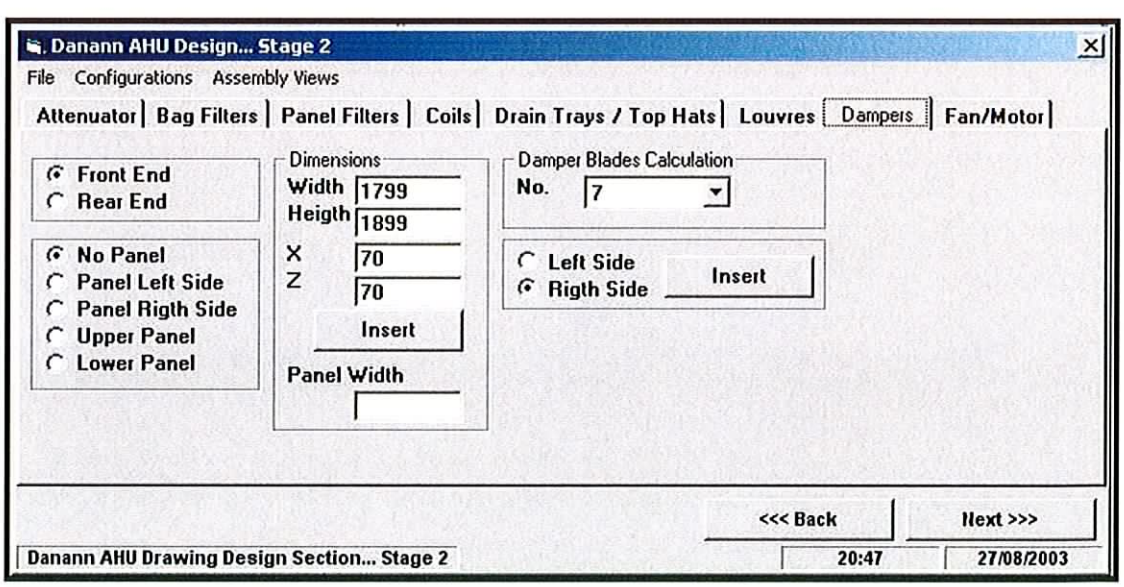


Fig. 75: Damper Graphical User Interface

Fig. 75 shows that the graphical user interface for the damper's section is very similar to the louvre's graphical user interface. The differences between the two graphical user interfaces are the:

- "Damper Blades Calculation" option available on the damper's graphical user interface, and the
- "Insert Left Side" or "Insert Right Side" options that are located on the damper graphical user interface also

The Solid Edge sub-assembly that is used contains identical parts for the insulated panel that can inserted into the air-handling unit assembly along with the damper itself. The only unique part in appearance is the sub-assembly is the damper itself.

The **"Damper Blades Calculation"** option on the graphical user interface for the damper allows the user to add more blades to the damper after it has been inserted into the air-handling unit main assembly

The **"Insert Left Side"** or **"Insert Right Side"** options give the software the capability to insert dampers inside the air-handling unit to the front or rear in the event that the manufactured unit will be installed outdoors. If the manufactured air-handling unit will be installed outdoors it may contain more than one louvre to the front and/or rear meaning that the damper will have to be installed just inside the air-handling unit frame opposite the louvre.

6.2.8 Global Variables

The software was designed so that the user would never have to key in the same information more than once to accomplish multiple tasks through out the course of the air-handling unit design.

A clear example of this was evident in Stage 2 when the user inserted a bag filter section and panel filter section into the air-handling unit. The graphical user interface in Stage 2 contained a great deal of information about the bag filter and panel filter sections when the graphical user interface for Stage 2 first opened. The information that was already supplied to the graphical user interface came from Stage 1. When the user keyed in the information during Stage 1 and then progressed to Stage 2 the software

saved all the information necessary from Stage 1 so it could be used in Stage 2 and Stage 3. When Stage 2 opens, it accesses the information in the global variables that was saved from Stage 1 and applies it in Stage 2. As a result the graphical user interface contained all the necessary information required to design the bag filters and panel filters.

Fig. 76 illustrates how the software saves information away from Stage 1, 2 and 03. Each piece of information saved away from each Stage is called a "Global Variable". Once a global variable has been saved its information can be accessed from any stage in the software i.e. Stage 1, 2 or 03.

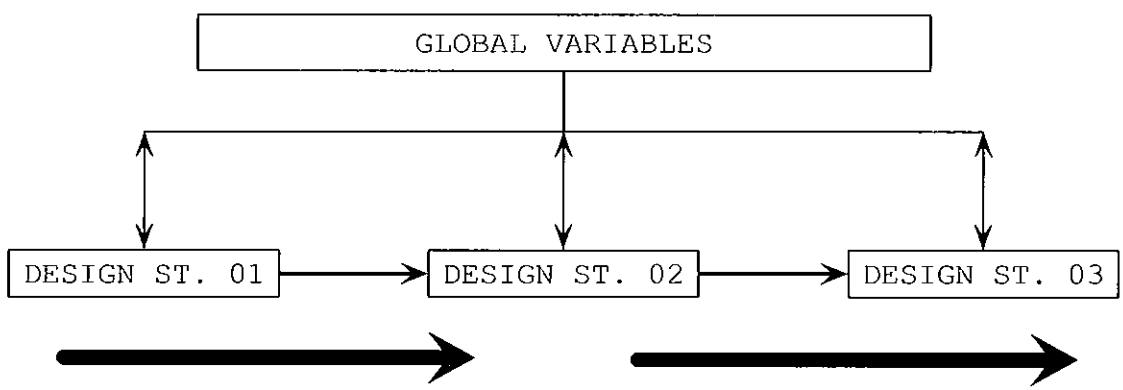


Fig. 76: Global Variables

As Stage 1 is the first stage of the design, it can save information as global variables but there is no information present for it to read. The software is designed to save any data as a global variable that is needed elsewhere by another stage of design. Therefore, when the software continues to Stage 2 it can fully access all the information that was saved by Stage 1. After Stage 2 has completed its necessary tasks it can save all the information that it has accumulated about the air-handling unit so that Stage 3 may have access to it. When Stage 3 opens it will have all the information at hand that was saved from Stages 1 and 2. The more information that is saved in the global variables means the less information that is required to be keyed in again by the user or re-calculated by the software as it progresses through the design of the air-handling unit.

Fig. 77 shows the start up code for Stage 2

```
Combo2.Text = 2
Combo".Text = "
'Panel-Filter & Bag-Filter Calculations
  Combo3.Text = QtyHzFilters
```

```
    Combo5.Text = QtyHzFilters
    Combo4.Text = QtyVrFilters
    Combo6.Text = QtyVrFilters
    Check1.Value = QtyHalfVrFilters
    Check2.Value = QtyHalfHzFilters
    Check3.Value = QtyHalfVrFilters
    Check4.Value = QtyHalfHzFilters
    Text1(4).Text = QtyGaps
    Text1(5).Text = IndividualSpaceGap
'Attenuator Calculations
    If Frame_A = 1 Then
        InsideComponentSpace = Frame_Width - 109
        QtyPods = Combo2.Text
        QtyGaps = QtyPods + 1
        PodWidth = 188
        PodSpaceRequired = QtyPods * PodWidth
        SpaceLeft = InsideComponentSpace - PodSpaceRequired
        IndividualSpaceGap = (SpaceLeft / QtyGaps)
        Positioning = 54.5 + IndividualSpaceGap
        Text1(1).Text = Frame_Heigth - (109 + Val(Text1(40).Text) + Val(Text1(41).Text))
    End If
    If Frame_B = 1 Then
        InsideComponentSpace = Frame_Width - 58
        QtyPods = Combo2.Text
        QtyGaps = QtyPods + 1
        PodWidth = 188
        PodSpaceRequired = QtyPods * PodWidth
        SpaceLeft = InsideComponentSpace - PodSpaceRequired
        IndividualSpaceGap = (SpaceLeft / QtyGaps)
        Positioning = 29 + IndividualSpaceGap
        Text1(1).Text = Frame_Heigth - (58 + Val(Text1(40).Text) + Val(Text1(41).Text))
    End If
'Louvre Settings
    If Frame_A = 1 Then
        Text1(27).Text = (Frame_Width - 140)
        Text1(28).Text = (Frame_Heigth - 140)
        Text1(29).Text = 70
        Text1(31).Text = 70
        Text1(33).Text = (Frame_Width - 140)
        Text1(34).Text = (Frame_Heigth - 140)
        Text1(32).Text = 70
        Text1(26).Text = 70
    End If
    If Frame_B = 1 Then
        Text1(27).Text = (Frame_Width - 100)
        Text1(28).Text = (Frame_Heigth - 100)
        Text1(29).Text = 50
        Text1(31).Text = 50
        Text1(33).Text = (Frame_Width - 100)
        Text1(34).Text = (Frame_Heigth - 100)
        Text1(32).Text = 50
        Text1(26).Text = 50
    End If
'Drain Tray Calculations
Combo1.Text = InsideComponentSpace - 2
    If Frame_A = 1 Then
        Text1(49).Text = 70
        Text1(37).Text = 70
    Else
        Text1(49).Text = 50
        Text1(37).Text = 50
    End If
```

Fig. 77: Stage 2 opening routine

The code shown in Fig. 77 displays some of the global variables that were accessed by the Stage 2 when it opened up. This information was saved during Stage 1. In relation to the bag filters and panel filters, Stage 1 saves the following information shown below in Fig. 78 as global variables.

```
Public TopFilterTH As Single
Public BottomFilterTH As Single
Public QtyHzFilters As Integer
```

```
Public QtyVrFilters As Integer
Public QtyHalfHzFilters As Integer
Public QtyHalfVrFilters As Integer
```

Fig. 78: Global Variables

The lines of code were written and saved in a module file called SE_Launch.bas. A module file is a file that is used to define global variables and write routines all of which can be used by any graphical user interface in the software project.

In the software the design procedures for the air-handling unit were defined into three individual stages. The module file, SE_Launch.bas, allowed the project to save information that was not specific to any of the three stages and to save it where the entire software project could access its information. By using module files and global variables meant that less programming code was required to be written for the software project and it was more manageable to carry information from one Stage to the next.

6.3 Stage 3: Exterior panel assembly and retrofit

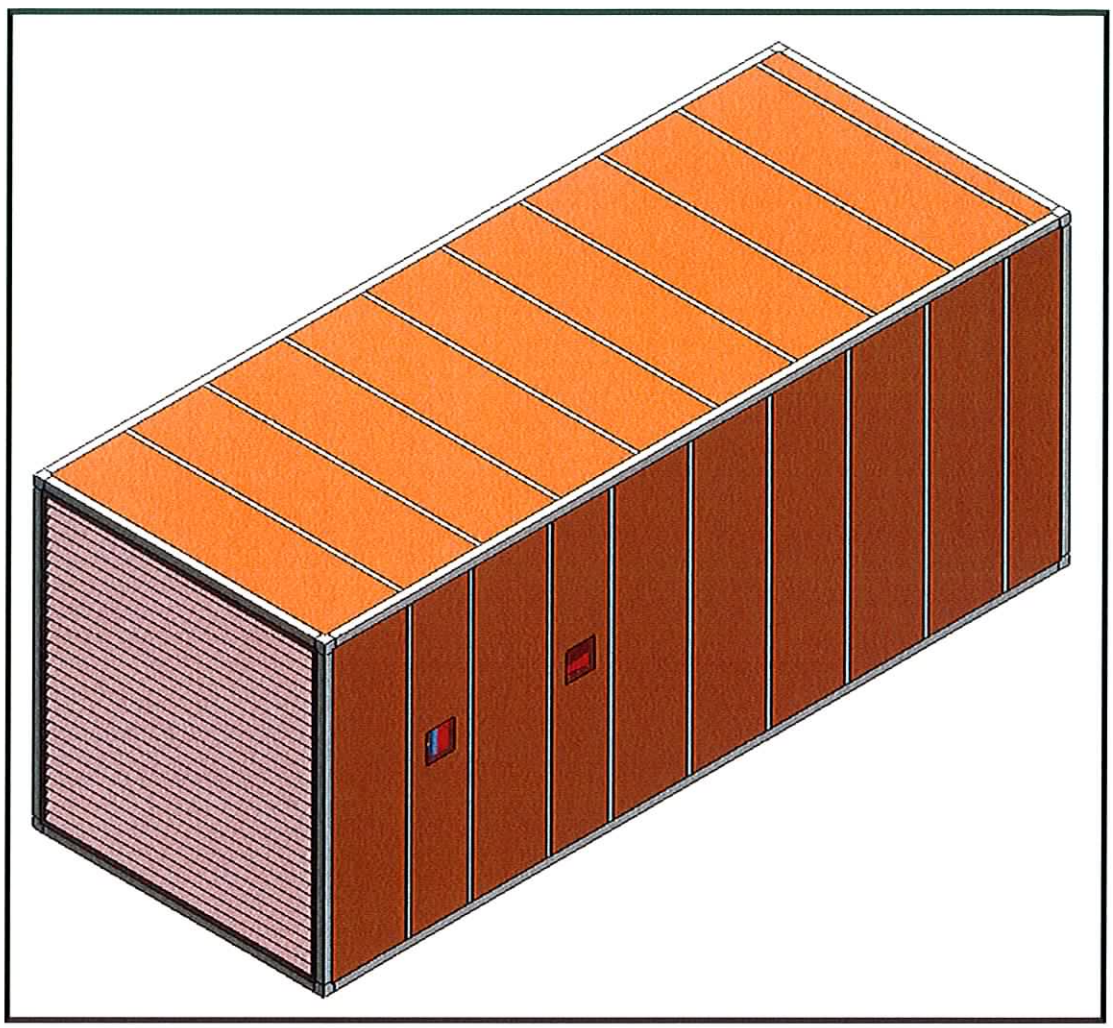


Fig. 79: Exterior view of a completed Air-handling Unit

Stage 3 is the air-handling unit's final stage of design and assembly. It is designed to deal with two main areas

- The exterior panelling arrangement of the air-handling units
- Generating working drawings from the air-handling unit solid model

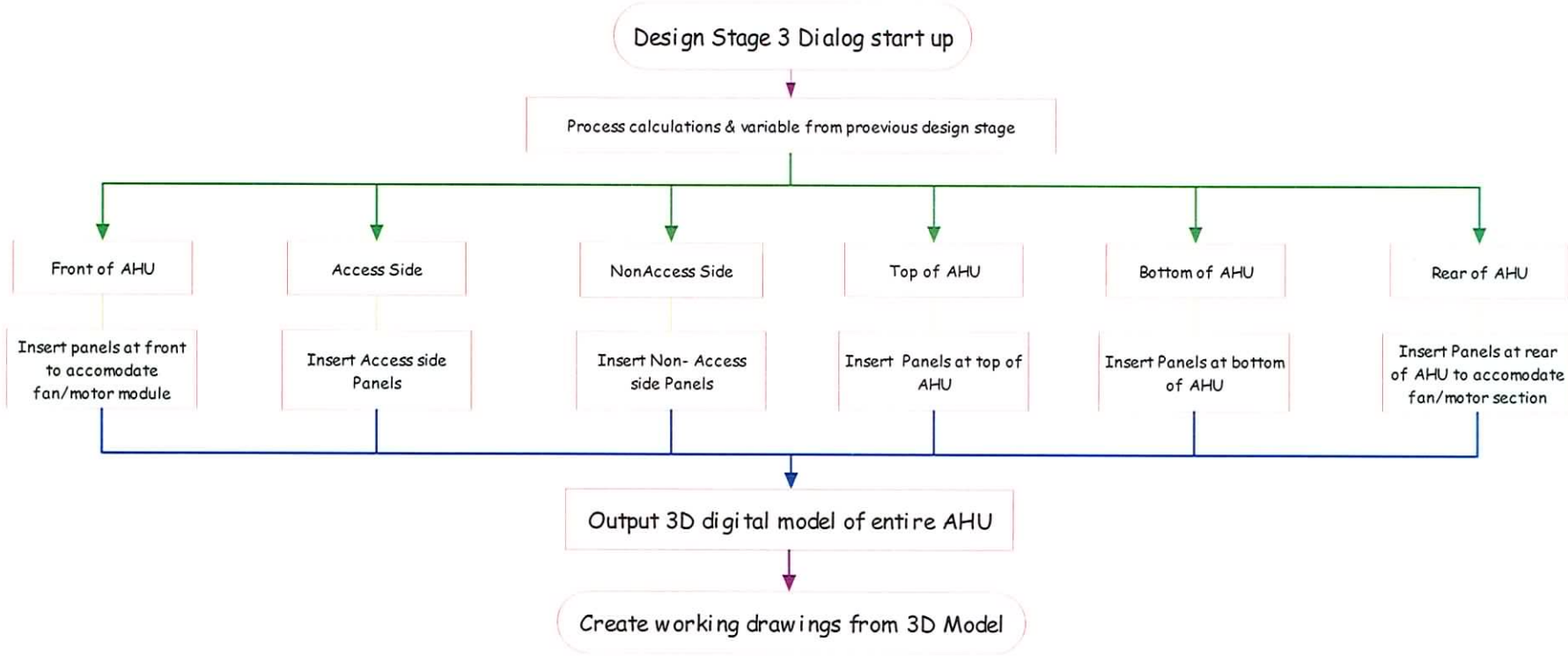


Fig. 80: Flowchart representing an overview of Stage 3

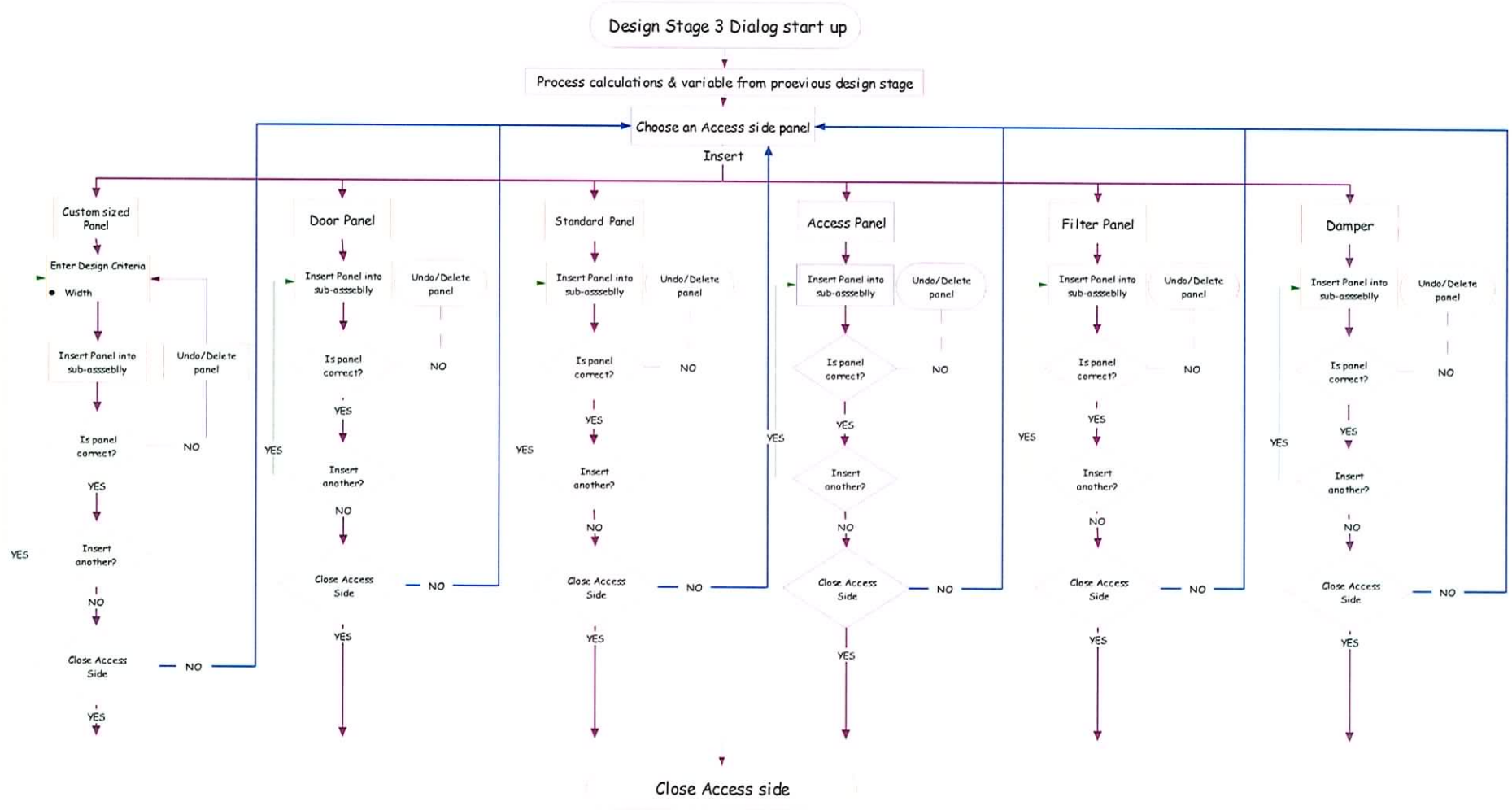


Fig. 81: A detailed look at placing a panel on the Access side of an AHU

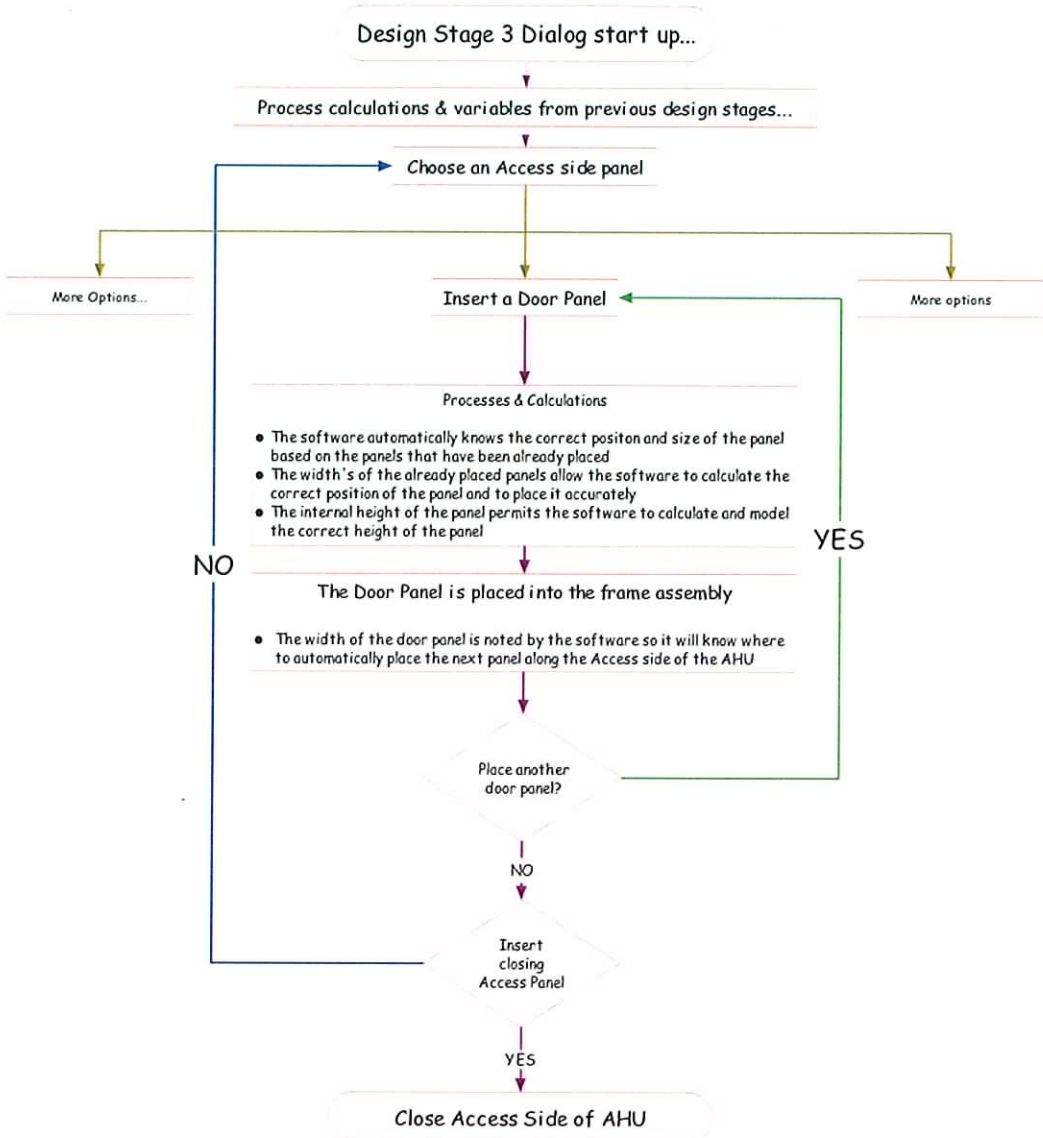


Fig. 82: A detailed look at placing an Access Door panel

6.3.1 The exterior panelling arrangement of the Air-handling Units

Panelling is required on all four sides of the air-handling unit and it may also be required at the front and back. Each panel itself is a sub-assembly and must be accurately designed. The dimensions of each particular panel are determined by the specifications of the subassemblies designed and inserted inside the air-handling unit during Stage 2. There are many different types of panel that can be used in an air-handling unit:

- Access panel. An access panel is 350 mm wide at standard dimensions. It has no window and is typically positioned to provide access to internal components such as filters. When maintenance is required on the filters the access panel can be removed to allow access to the filters.
- Door panel. A door panel is 500 mm wide at standard dimensions. It has a window to provide visibility into the internal components of the air-handling unit and it is mounted on hinges for easy access. A flow chart describing the processes involved inserting a Door panel into the air-handling unit assembly can be seen in Fig. 82
- Standard panel. A standard panel is 648 mm wide at standard dimensions. The dimension of 648 mm is governed by the size of the sheet metal provided by supplier. A standard panel is typically used to seal up the side of the air-handling unit and to make it airtight. Standard panels have no facility to provide access to the internal components of the air-handling unit.
- Panel filter panel. A panel filter panel is 140 mm wide at standard dimensions. A panel filter panel is designed to provide access to the panel filters inside in the air-handling unit to allow maintenance be carried out.
- Custom panel. Custom panels are used where spaces or gaps occur along the air-handling unit's side because the current arrangement of panels does not completely seal the air-handling unit. A custom panel has no fixed width as it is designed and built to fill space along the side of the air-handling unit.
- Damper. Dampers have no standard size. They can be custom built to meet any dimensional requirements. They are most often found at the front or rear of the air-handling unit, but they can be mounted on either side, or on the roof of the air-handling unit when required.

Fig. 79 shows a fully panelled air-handling unit, it illustrates that some of the panel sub-assemblies may even have windows. The panels can be seen in Fig. 79 are displayed in orange. The light grey strips illustrated between the panels in Fig. 79 are the supports (aluminium extrusions) that the panels are fixed to so that they are held in place. Each panel is fixed to the air-handling unit's frame with rivets. Rubber insulation is fitted between the back face of the panel and the air-handling unit frame to keep the panels air tight. The panel also has a 1.5 mm clearance around its edges to ensure that the only contact the panel has with the air-handling unit is with the rubber insulation. Inserting

an accurate panel into the frame assembly is essential because each panel contains two different fabricated pieces of sheet metal and an aluminium extrusion for support. Both pieces of sheet metal are of different thickness. To obtain advanced and accurate cut lists from the model, every feature of each component in the model must be accurate.

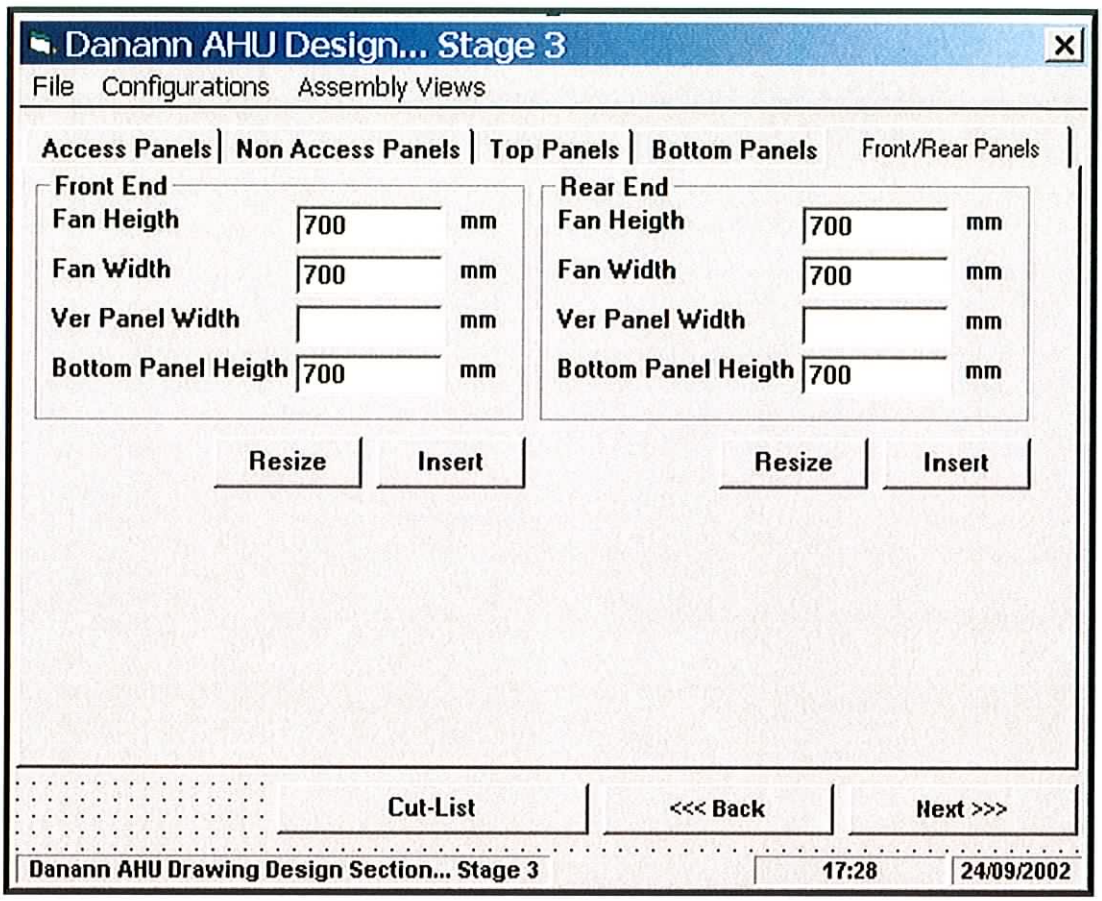


Fig. 83: Stage 3

Fig. 83 shows the graphical user interface designed for Stage 3. The graphical user interface has a "TAB" for each side of the unit where panelling is required. It also has a "TAB" labelled "Front/Rear Panels" to facilitate the software in closing the front and rear of the air-handling unit.

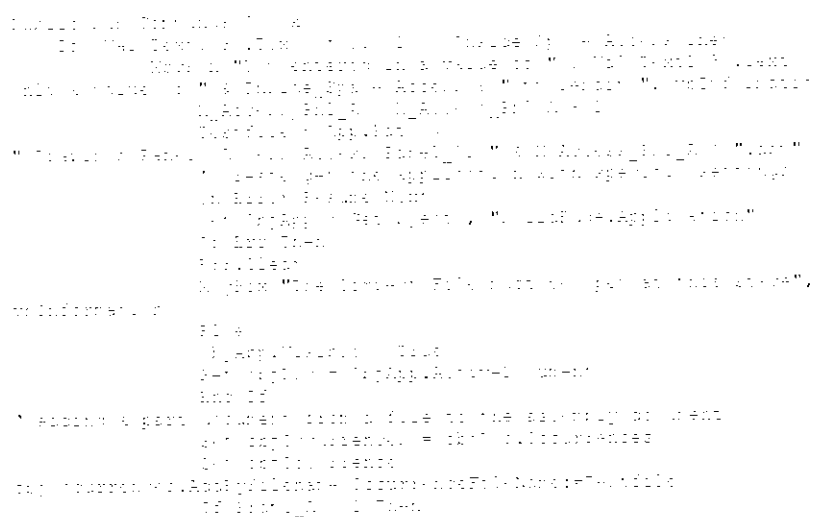
It was a requirement that Stage 3 had the capability to position each panel without requiring information from the user. The software was designed to take information from the geometry of the air-handling unit's external frame to position the first panel on any side of the air-handling unit. The software is designed such that after the first panel had been added to the air-handling unit it would take information from the air-handling unit and from the panel to calculate the next panel's location. The software is also

designed so that every time a panel is added to the assembly information from the panel is recorded by the software. The software is designed to allow the user place panels around the air-handling unit in any sequence. When there was not enough space left in any side of the air-handling unit's external frame to insert anymore panels the software is designed to calculate exactly how much space was left. From this value the software custom builds a panel to fill the space that is left to seal that particular side of the air-handling unit.

This is achieved as follows: every time a panel sub-assembly is placed into the existing assembly, the value of the combined width of the panel is added to a variable so that when the software places another panel subassembly the new value for the variable tells the software where to place the next panel.

Fig. 84: Example of incrimination calculations

As mentioned the software is designed with a feature to close off the air-handling unit automatically. The width of this panel is automatically calculated from the widths of the existing panel sub-assemblies and the interior length of the air-handling unit. This feature is unique because the panel that closes the air-handling unit is a different sub-assembly to the standard panel sub-assembly. The closing panel sub-assembly has no aluminium extrusion for support, it uses the inside edge of the air-handling unit frame along with the aluminium extrusion from the last panel sub-assembly inserted. Fig. 85 shows an example of code that places a closing panel.



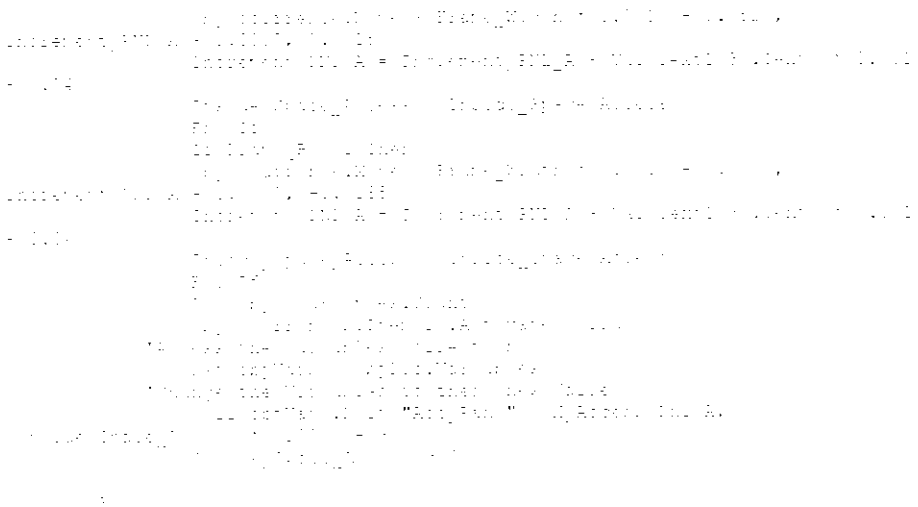


Fig. 85: Example code closing the Access side of an AHU

A feature incorporated into the design of the air-handling units is the ability of the panels to automatically resize if a change is made to the height of the air-handling unit. If the frame is modified the panel heights will update with the frame. This was achieved by using advanced solid modelling. The panels are modelled associatively to the interior of the frame assembly so that a change in the frame assembly would also mean a change in the panels.

The procedures involved in inserting the panel from the graphical user interface for Stage 3 can be seen from Fig. 80, Fig. 81 and Fig. 82. Fig. 80 displays the choice that the user is faced with and the range of panels that the user can insert into the air-handling unit assembly.

Fig. 81 shows in detail the processes and procedures involved in choosing a panel for the Access side of the air-handling unit assembly. The user can choose the following styles of panel:

- Custom width panel
- Door panel
- Standard panel
- Access panel
- Filter panel
- Damper

After specifying the appropriate style of panel the user must key in its design criteria and click on the **"Insert"** button. The software will insert the corresponding panel into the side of the air-handling unit assembly. The user then has the choice of entering another panel to the access side. If the user chooses to do so the software will insert the next panel along side the previously inserted one. If the user wishes to delete a panel after inserting it into the assembly it can be removed by clicking on the **"Delete"** button. When there is no more space remaining on the side that the user is inserting the panels on it can close off that particular side by inserting a custom panel. The software is designed to automatically close the side of the air-handling unit by inserting a panel on its own and no support that is inserted with every other panel.

Once Stages 1, 2 and 3 have been completed it is necessary to prepare the air-handling unit model to extract necessary manufacturing information from it. While Stage 3 is the final stage in building the air-handling unit assembly model, Stage 3 also contains a feature for producing a Solid Edge draft document containing a set of two-dimensional drawings from the air-handling unit assembly model.

6.3.2 Generating working drawings from an Air-handling Unit solid model

After the user has completed the air-handling unit solid model it can then progress with the preparation of manufacturing drawings from the air-handling unit. An automated routine was specially created to generate a set of specific two-dimensional drawings. See Fig. 88. Every air-handling unit can be described in a set of six two-dimensional views. Specially designed components such as drain tray's and fan frames require their own special drawings, but, the air-handling unit and the arrangement its internal components are can be described in six two-dimensional views on a series of A4 sheets.

Fig. 86 shows the menu on the graphical user interface for Stage 3 that generates the working drawings from the air-handling unit solid model. It is labelled **"Save and go to Draft"**. When the user chooses the **"Save and go to Draft"** from the file menu the user activates a series of events that creates a Solid Edge draft file from the air-handling unit solid model. The user then prints off the draft file.

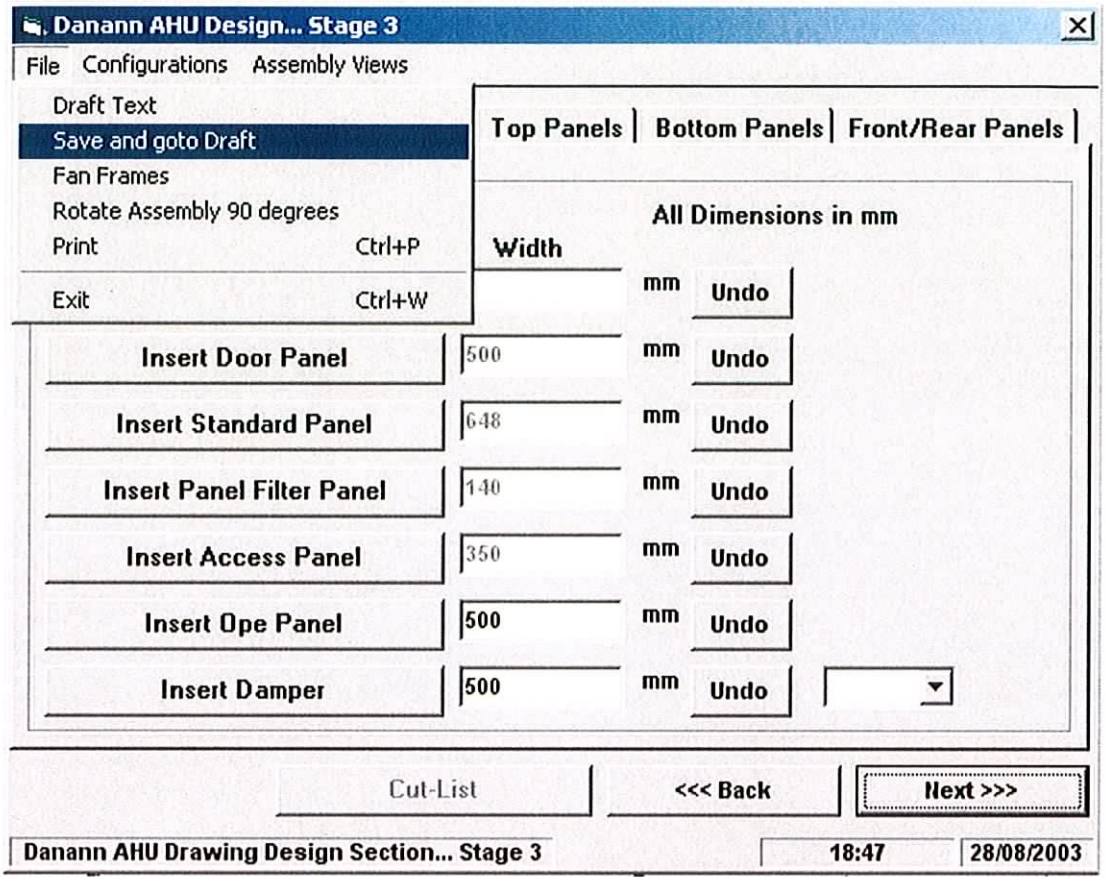


Fig. 86: Graphical User Interface

After the user has generated the draft file for the air-handling unit solid model it could possibly contain up to ten individual sheets. The sheets collectively will contain the necessary views of the air-handling unit and sub-assemblies that have to be custom made e.g. drain tray, base frame. Once the drawing views of the air-handling unit have been put in place the user must fill in the required text that must be printed on each sheet.

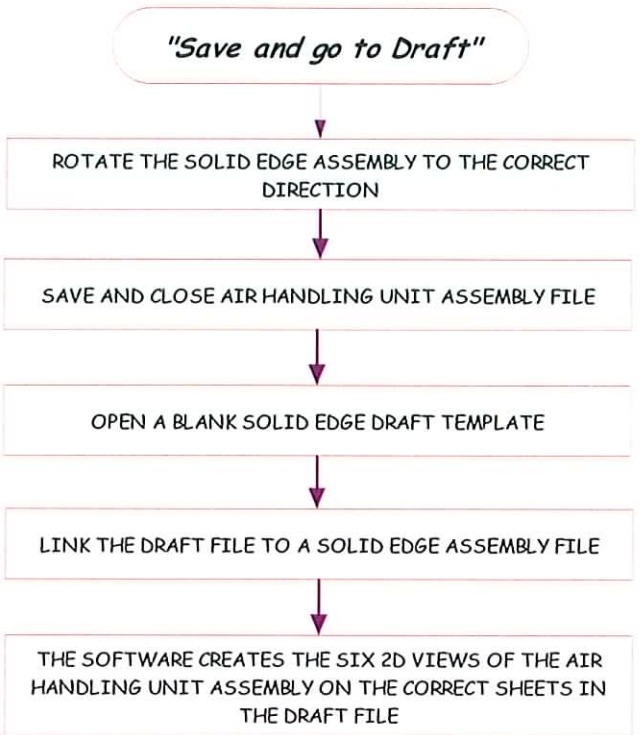


Fig. 87: Generating 2D drawings from the Air-handling Unit solid model

From the flow chart in Fig. 87 the routine used to create a Solid Edge draft file from the air-handling unit solid model can be seen. The first step in creating the draft file is to rotate all the parts in the Solid Edge assembly so that the air-handling unit is facing in the correct direction. It is necessary for the software to rotate the air-handling unit to face the correct direction so the plans and elevations are generated from the correct viewing point in the Solid Edge draft file.

Once the Solid Edge assembly file is facing the appropriate direction the software then saves and closes it and opens up a blank Solid Edge draft file from an appropriate template.

The software then links the draft file to the assembly file. Once the link has been made the software then generates the appropriate two-dimensional views of the air-handling unit and its accompanying sub-assemblies. See Fig. 88 for the programming code created to achieve this.

```
' Create/get the application with specific settings
On Error Resume Next
Set ObjApp = GetObject(, "SolidEdge.Application")
If Err Then
Err.Clear
MsgBox "The Correct File must be Open at this stage", vbInformation
```

```
Else
ObjApp.Visible = True
Set objDoc = ObjApp.ActiveDocument
End If

Set objOccurrences = objDoc.Occurrences

' rotating the placed part in the assembly
dblAxisX1 = 0
dblAxisX2 = 0
dblAxisY1 = 0
dblAxisY2 = 0
dblAxisZ1 = 0
dblAxisZ2 = 0.01

For I = 1 To objOccurrences.Count

Call objOccurrences.Item(I).Rotate(AxisX1:=dblAxisX1, AxisY1:=dblAxisY1, AxisZ1:=dblAxisZ1, _
AxisX2:=dblAxisX2, AxisY2:=dblAxisY2, AxisZ2:=dblAxisZ2, Angle:=(PI * 1.5))

Next

Call objDoc.Windows.Item(1).Activate
Call objDoc.Save
Call objDoc.Close

Set objDoc = ObjApp.Documents.Open(App.Path & "\Drawings\Draft Damam\Adit")
Set objDoc = ObjApp.ActiveDocument

Testfile = App.Path & "\Drawings\Top_Level\Assembly_H_V_Frame\Assembly.asm"

Set objModelLinks = objDoc.ModelLinks
Set objModelLink = objModelLinks.AddByFileName(Testfile)

objDoc.ActiveSection.Name = "Working Section"

Set objSheet = objDoc.ActiveSection.Sheets(1)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igFrontView, 0.02, 0.15, 0.15, seAssemblyDesignedView)
objDrwnView.Configuration = "07-Components"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

Set objSheet = objDoc.ActiveSection.Sheets(1)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddByFold(objDrwnViews(1), igFoldDown, 0.15, 0.085)
objDrwnView.Configuration = "07-Components"
objDrwnViews.CheckConfiguration (True)
objDrwnViews.Update

Set objSheet = objDoc.ActiveSection.Sheets(2)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igFrontView, 0.02, 0.15, 0.125, seAssemblyDesignedView)
objDrwnView.Configuration = "03-Access Side TH"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

Set objSheet = objDoc.ActiveSection.Sheets(3)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igBackView, 0.02, 0.15, 0.125, seAssemblyDesignedView)
objDrwnView.Configuration = "04-Non Access Side TH"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

Set objSheet = objDoc.ActiveSection.Sheets(4)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igTopView, 0.02, 0.15, 0.16, seAssemblyDesignedView)
objDrwnView.Configuration = "05-Top Side TH"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

Set objSheet = objDoc.ActiveSection.Sheets(4)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igBottomView, 0.02, 0.15, 0.095, seAssemblyDesignedView)
objDrwnView.Configuration = "06-Bottom Side TH"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

Set objSheet = objDoc.ActiveSection.Sheets(5)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igLeftView, 0.02, 0.095, 0.125, seAssemblyDesignedView)
objDrwnView.Configuration = "02-Frame Only"
```

```
objDrwnView.CheckConfiguration (True)
objDrwnView.Update
Set objSheet = objDoc.ActiveSection.Sheets(5)
Set objDrwnViews = objSheet.DrawingViews
Set objDrwnView = objDrwnViews.AddAssemblyView(objModelLink, igRightView, 0.02, 0.205, 0.125, seAssemblyDesignedView)
objDrwnView.Configuration = "02-Frame Only"
objDrwnView.CheckConfiguration (True)
objDrwnView.Update

' Release objects
Set ObjApp = Nothing
Set objDoc = Nothing
Set ObjDocs = Nothing
Set objOccurrences = Nothing
Set objOccurrence = Nothing
Set objVars = Nothing
Set objVarList = Nothing

Me.Show
```

Fig. 88: Programming code for generating a set of draft views from an Air-handling Unit solid model

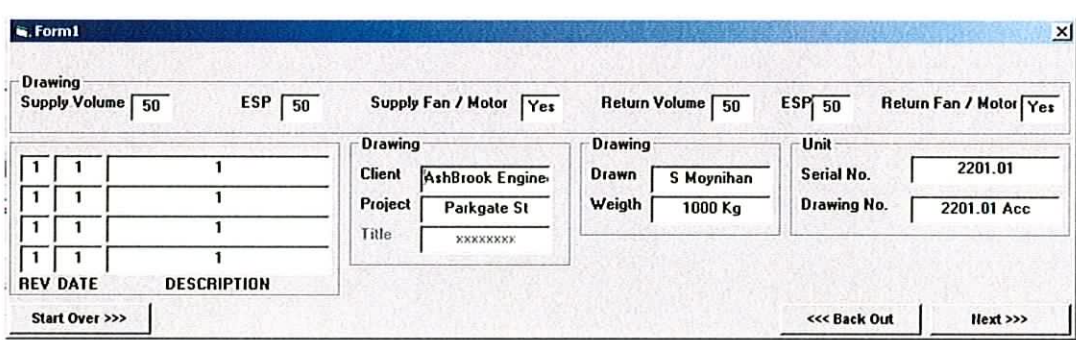
Firstly the software has to make sure that the air-handling unit assembly solid model is facing the correct way. All of the draft views are then generated from the air-handling unit assembly sitting in one location positioned one way. If the orientation of the air-handling unit assembly is incorrect all of the draft views in the draft file will be incorrect also. Shown in Fig. 89 is the section of the routine that is responsible for rotating the parts in the assembly so they are facing the correct direction.

```
For I = 1 To objOccurrences.Count

Call objOccurrences.Item(I).Rotate(AxisX1:=dblAxisX1, AxisY1:=dblAxisY1, AxisZ1:=dblAxisZ1, _
AxisX2:=dblAxisX2, AxisY2:=dblAxisY2, AxisZ2:=dblAxisZ2, Angle:=(PI * 1.5))

Next
```

Fig. 89: Code for giving the Air-handling Unit assembly the correct orientation

6.3.3 Generating text on Solid Edge draft documents

Once the Solid Edge draft file has been completed it is necessary to fill it in with the appropriate text before it is printed off. Shown in Fig. 90: Graphical User Interface



is the location on the Stage 3 graphical user interface where the facility for creating text on the Solid Edge draft file can be found.

Once the user chooses the **"Draft Text"** from the file menu the **"Draft Text"** graphical user interface opens up and the user is prompted to enter text in to the required fields. It can clearly be seen from the **"Draft Text"** graphical user interface in Fig. 91 that the labels on the graphical user interface correspond to the labels on the Solid Edge draft template that was created for the air-handling unit manufacturing drawings.

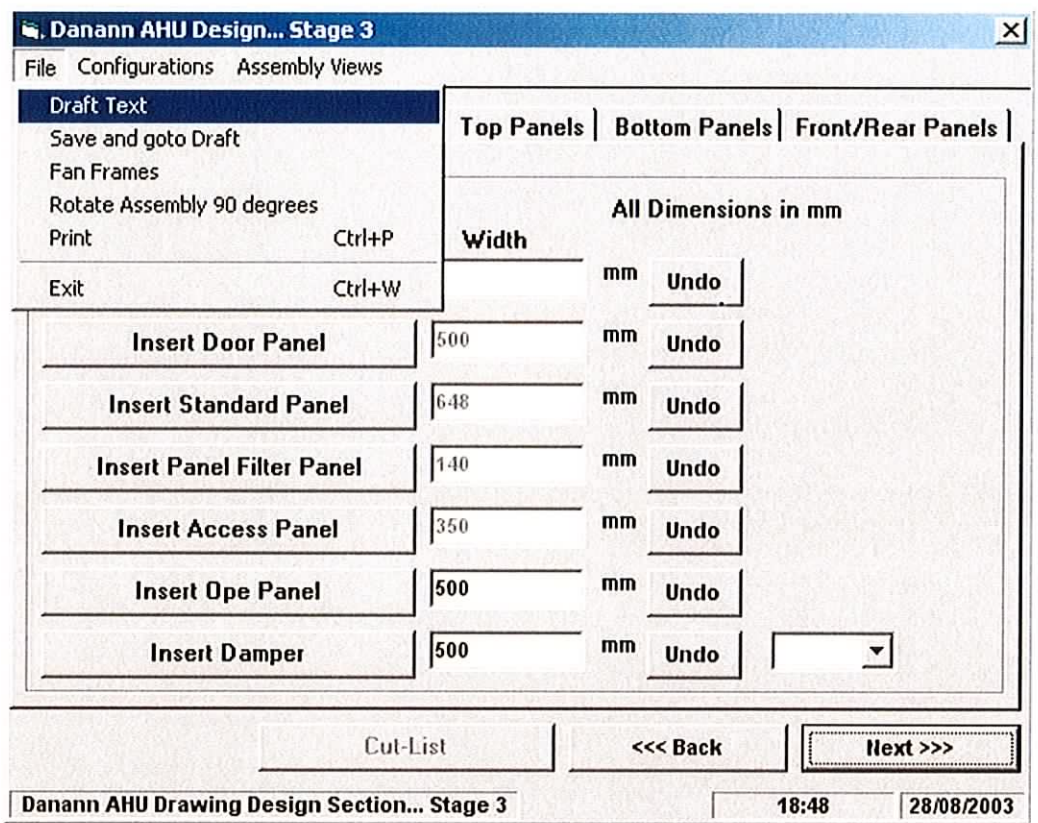


Fig. 90: Graphical User Interface

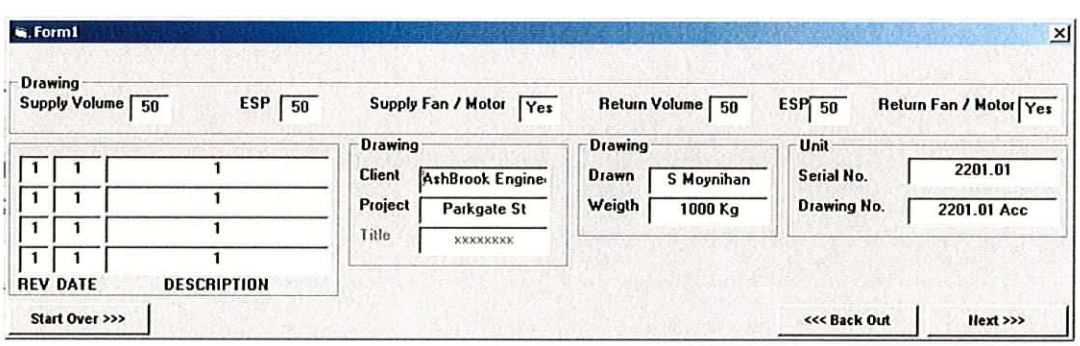


Fig. 91: Draft Text

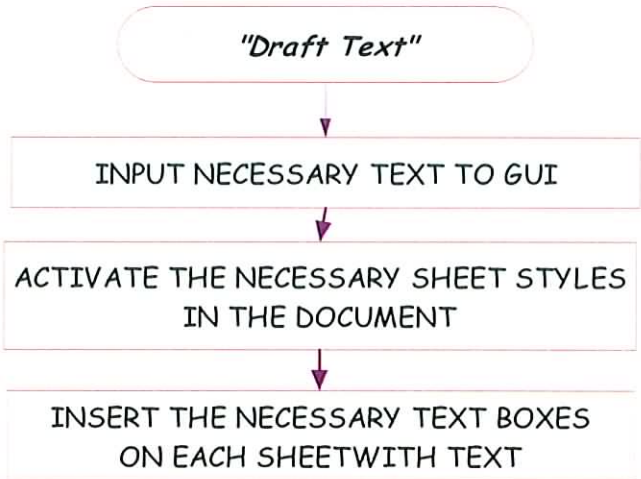


Fig. 92: Draft Text Process

Fig. 92 shows a flow chart that describes the procedure involved in placing text onto the Solid Edge draft file. The routine created for this procedure is designed to place the same text onto multiple sheets in the same Solid Edge draft document.

The first stage in the procedure is to fill in the required text into the graphical user interface shown in Fig. 91. After the text has been keyed in, the user must click on the **"Next"** button. The software then cycles through the routine in Fig. 93.

Firstly the software activates the appropriate Solid Edge draft file. The software then concentrates on the '**Working Section**' of the draft file. The working section of the draft file is the section where the two-dimensional views are situated. It is this section the text the routine must place text on the Solid Edge draft file.

After the working section has been activated the routine inserts all the appropriate text boxes along with the necessary text on to each sheet. When the routine has concluded all the appropriate draft views and text will have been added to the draft file where it will be ready to be printed.

```
' Create/get the application with specific settings
    On Error Resume Next
    Set ObjApp = GetObject(, "SolidEdge.Application")
    If Err Then
    Err.Clear
    MsgBox "The Correct File must be Open at this stage", vbInformation
    Else
    ObjApp.Visible = True
    Set objDoc = ObjApp.ActiveDocument
    End If

    objDoc.ActiveSection = 0 'Working Section
```

```
Set objSheets = objDoc.ActiveSection.Sheets

For I = 1 To objSheets.Count

Set objSheet = objSheets.Item(I)
Set objTxtboxs = objSheet.TextBoxes
Set objTxtStls = objDoc.TextStyles

objTxtStls.Active = "DamannA"

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.13006, 0.03494, 0, 0.00831, 0.00069, 0)
objTxtbox.Text = txtDft(0).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.13006, 0.02647, 0, 0.00831, 0.00069, 0)
objTxtbox.Text = txtDft(1).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

objTxtStls.Active = "DamannB"

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.20039, 0.02982, 0, 0.00486, 0.00265, 0)
objTxtbox.Text = txtDft(4).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.20039, 0.03494, 0, 0.00486, 0.00265, 0)
objTxtbox.Text = txtDft(3).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

objTxtStls.Active = "DamannC"

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.22706, 0.03244, 0, 0.00995, 0.051, 0)
objTxtbox.Text = txtDft(5).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.22706, 0.02006, 0, 0.00995, 0.051, 0)
objTxtbox.Text = txtDft(6).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

objTxtStls.Active = "DamannD"

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.05, 0.2025, 0, 0.0065, 0.023, 0)
objTxtbox.Text = txtDft(7).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.06102, 0.2025, 0, 0.0065, 0.02394, 0)
objTxtbox.Text = txtDft(8).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.20905, 0.2025, 0, 0.0065, 0.02394, 0)
objTxtbox.Text = txtDft(9).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.25997, 0.2025, 0, 0.0065, 0.01439, 0)
objTxtbox.Text = txtDft(10).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.191, 0.2025, 0, 0.0065, 0.01108, 0)
objTxtbox.Text = txtDft(11).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.239, 0.2025, 0, 0.0065, 0.02, 0)
objTxtbox.Text = txtDft(12).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

    objTxtStls.Active = "DamannE"

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.01, 0.035, 0, 0.005, 0.006, 0)
objTxtbox.Text = txtDft(13).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.01, 0.03, 0, 0.005, 0.006, 0)
objTxtbox.Text = txtDft(16).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.01, 0.025, 0, 0.005, 0.006, 0)
objTxtbox.Text = txtDft(19).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.01, 0.02, 0, 0.005, 0.006, 0)
objTxtbox.Text = txtDft(22).Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter
```

```
Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.016, 0.035, 0, 0.005, 0.01, 0)
objTxtbox.Text = txtDft014.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.016, 0.03, 0, 0.005, 0.01, 0)
objTxtbox.Text = txtDft017.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.016, 0.025, 0, 0.005, 0.01, 0)
objTxtbox.Text = txtDft020.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.016, 0.02, 0, 0.005, 0.01, 0)
objTxtbox.Text = txtDft023.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.026, 0.035, 0, 0.005, 0.035, 0)
objTxtbox.Text = txtDft015.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.026, 0.03, 0, 0.005, 0.035, 0)
objTxtbox.Text = txtDft018.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.026, 0.025, 0, 0.005, 0.035, 0)
objTxtbox.Text = txtDft021.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Set objTxtbox = objTxtboxs.AddByHtWd(Ang(0.026, 0.02, 0, 0.005, 0.035, 0)
objTxtbox.Text = txtDft024.Text
objTxtbox.VerticalAlignment = igTextHzAlignVCenter

Next
'Release objects
Set ObjApp = Nothing
Set objDoc = Nothing
Set ObjDocs = Nothing
Set objOccurrences = Nothing
Set objOccurrence = Nothing
Set objVars = Nothing
Set objVarList = Nothing

Me.Show
```

Fig. 93: Draft Text automated routine

6.4 Critical assessment of the bespoke software

To evaluate the bespoke software post programming and implementation, a critical assessment was performed on it to determine:

- did the bespoke system achieve the defined project goals set out at the beginning of the project
- the feasibility of customising and reconfiguring the software

6.4.1 Testing the bespoke system

In order to assess if the bespoke system achieved the goals set out at the beginning of the project three tests were carried out to test its performance. These tests were

necessary in order to measure how successful, or not, the new system is. A number of existing design and drafting techniques used by the company were compared to the new system to create automated designs. The tests were designed to analyse the performance of the system and ascertain how successful it is in achieving its goals i.e. in being a more productive and efficient system than the existing one. The tests were performed in three stages, each stage consisting of two parts.

6.4.1.1 The first test

Phase 1 of the first test was to complete stage 1 of the air-handling unit design. The first test was designed to measure the time taken by both systems to develop manufacturing drawings for an air-handling unit external frame. Phase 2 of the first test was then to implement one design revision and to measure the time necessary to implement the design revision. See the results in Table 2.

	Current 2D system	**Custom designed system**
Phase 1	16 minutes	1 minute
Phase 2	10 minutes	1 minute
Total Time	26 minutes	2 minutes

Table 2: Test 1 results

The results from Table 2 clearly show that the customised system completed both of its tasks much quicker than the current 2D system. The major advantage that the new system had over the current system was that to implement the design revision it was only necessary to re-key the new dimensions of the external frame of the air-handling unit and then click on the "Resize" command button located on the graphical user interface. To implement the design revision with the current 2D system involved manually editing three drawing views from one drawing file and then manually revising them to ensure that all views were consistent and free from errors. To implement the design revision it was imperative that the user had comprehensive knowledge and experience of the commands necessary to edit the drawing views.

6.4.1.2 The second test

Phase 1 of the second test was to complete stage 2 of the air-handling unit design. This test was designed to measure the time taken by both systems to further develop the manufacturing drawings from stage 1 to include all the internal components and sub-assemblies. Phase 2 of the test was to measure the time taken to implement one major design revision to the drawings and update them. See the results in Table 3

	Current 2D system	Custom designed system
Phase 1	15 minutes	3 minutes
Phase 2	15 minutes	3 minutes
Total Time	30 minutes	6 minutes

Table 3: Test 2 results

The results from Table 3 clearly show that the new system again completed both of its tasks much quicker than the current 2D system. The process for implementing design revisions with the custom designed system involved a combination of resizing components as well as removing them from the assembly and reinserting them depending on the complexity of the design revision. With the current 2D system the process never changes for implementing the design revisions. Depending on the revisions complexity it can be implemented by simply editing the necessary components of the drawing, or, the drawing must be deleted and redrawn. As the current system is purely 2D, every drawing view that is associated with the design revision must go through the same analysis and editing procedures. Once the revisions have been implemented they must then be revised for consistency and errors.

6.4.1.3 The third test

Phase 1 of the final test was to complete stage 3 of the air-handling unit design. This test was designed to measure the time taken by both systems to further develop the manufacturing drawings from stage 2 to include all the external panel subassemblies that are used to seal off the air-handling units from the outer atmosphere. Phase 2 of the test was to measure the time taken to implement one major design revision to the drawings and update them. See the results in Table 4.

	Current 2D system	Custom designed system
Phase 1	15 minutes	4 minutes
Phase 2	15 minutes	4 minutes
Total Time	30 minutes	8 minutes

Table 4: Test 3 results

The results from Table 4 clearly show that the customised system again completed both of its tasks much quicker than the current 2D system. In test 3, the current system again suffered in the same manner as outlined in test 1 and test 2. The final test illustrated that the current system could match the custom developed system for functionality, productivity and efficiency.

After all three tests were completed it became clear that the custom developed system out-performed the current 2D system in each phase of each test. The overall results can be seen in Table 5.

	Current 2D system	Custom designed system
Test 1	26 minutes	2 minutes
Test 2	30 minutes	6 minutes
Test 3	30 minutes	8 minutes
Total Time	86 minutes	16 minutes

Table 5: Final results

The advantages of the new system were evident when a design revision was performed once the drawings were completed. The new system was able to manage the design revision with ease and it was able to implement it almost immediately. To implement the modifications with the current 2D system was tedious due to the non-associative nature of the software. In some cases more than one drawing had to be updated manually as the 2D system has no facility in place to monitor the revision status of the drawings resulting in the process where by a revision to one drawing requires a manual revision of all related drawings to ensure consistency.

Another key advantage of the bespoke system was the level of skill necessary to operate it. With the 2D system strong computer literacy, intense training and usage of system

and the ability to understand 2D engineering drawings is necessary before acceptable levels of competency can be achieved to produce manufacturing documentation for the air-handling units. In contrary the bespoke system can be used with minimal levels of computer literacy and knowledge of engineering drawings. This allows design engineers to concentrate on other aspects of manufacturing while less skilled personnel can produce the necessary manufacturing drawings and documentation using the bespoke system.

6.4.2 Updating the existing bespoke software

In order to assess feasibility of customising and reconfiguring the software, an assessment into the time and effort involved in implementing a modification to stage 1 of the software would be performed. The modification would include facilitating the software with an additional choice of external frame type for designing air-handling units.

The new choice of frame was chosen for the software because it is the most difficult modification to implement. The frame is the first element of the air-handling unit to be designed and all the dimensions of the other components are calculated from its characteristics.

6.4.2.1 Discussion of factors to be considered

Each internal and external component of an air-handling unit is programmed to understand which type of the air-handling unit has been chosen. Therefore, if a new frame profile is introduced for future projects, each component that could be included in an air-handling unit, with the new frame type, must be updated with a new section of code so the software can make the same calculations as are made when an existing frame type is chosen.

After a frame is chosen in stage 1 of the design, the software calculates numerous reference points around it to facilitate automatic functions, such as dimensioning and positioning of components that will be inserted to the frame at later stages. After an additional component is chosen in stage 2 or 3, using the reference points from stage 1, the software has calculated the new components dimensions and its X and Y

coordinates. The outstanding information to be supplied to the software is the Z coordinates i.e. the distance from the front face of the air-handling unit to the front face of the new component. This information is supplied by the user prior to commanding the software to insert the component. Therefore, the choice of frame made in stage 1 of the design has a direct effect on every part consequently inserted in to the air-handling unit.

Therefore, to facilitate all the components that may be specified during stages 2 and 3 of the design process to be correctly referenced from the new style of frame; it is first necessary to have them reprogrammed with the necessary information to do so.

6.4.2.2 Assessment of required programming and development

The amount of programming hours required to introduce the new frame profile and associated modifications would be dependant on the programmer's programming and solid modelling skills. At a minimum the programmer would need familiarity with the custom applications architecture and design to understand the relationships that were previously programmed between the frame profiles and the air-handling unit's internal and external components. Although a lot of new code would have to be written to completely update the software, the greater familiarity the programmer would have would imply the faster the software could be updated and prepared.

If the programmer had no previous exposure to the custom application during its design and implementation phase it would be difficult to assess how many programming hours it would take to successfully apply the necessary variations to its functionality. It is most likely that a successful implementation could not be achieved.

It should be noted that the bespoke system was written with a standard programming interface, Microsoft Visual Basic, which should aid the customisation and reconfiguration of the software. It should also be noted that each stage of the bespoke system (stage 1, stage 2 and stage 3) had a standard template that was used to prepare the required code for each component associated with that stage. The benefit being, that a change to any component could easily and quickly be implemented to the other components in that stage by referring to the necessary section and making the required modifications.

6.4.2.3 Process for updating the bespoke system

The process for updating the existing bespoke software to accommodate the additional choice of external frame should be performed using the following steps

- Fabrication of a solid model assembly of the new frame type with Solid Edge
- Redevelopment of stage 1 of the bespoke software
- Redevelopment of stage 2 of the bespoke software
- Redevelopment of stage 3 of the bespoke software
- Testing and debugging of the bespoke software
- Implementation of the new bespoke software

For the first step in the process, a solid model assembly of the new frame type must be created using the same procedures as the existing assemblies. It is important that the variable parameters of the new assembly are exactly the same as those for the existing assemblies to facilitate a smoother integration process with the bespoke software. As the assembly will feature a new type of frame profile previously unused, existing frame assembly information cannot be used as a staring point to commence the assembly. Therefore a good working knowledge of manual Solid Edge processes and procedures will be necessary to complete the assembly, necessary relationships and variable parameters as efficiently as possible.

Once the new assembly is constructed, the next step of the redevelopment is reprogramming of stage 1 of the bespoke system so that a user of the software can select and insert the new frame profile for a project. The existing code for stage 1 can be used as a template to generate the new code and should facilitate faster development. It is estimated that the combined time necessary to complete steps 1 and 2 of the redevelopment should take an experienced Solid Edge user and programmer approximately 4 programming hours.

The next step of the redevelopment is to update the internal and external components from stage 2 and stage 3 of the bespoke software. Reprogramming these components of the air-handling unit would involve adding a new section of the code to each component providing it with critical design information allowing it to resize and position itself in order to become compatible with the new frame profile. As with the previous step, each component must be individually updated and existing code can be used as a template for

the new code to be written. It is estimated that with the software in its existing state, the time that should be necessary to update both sections should be approximately 16 programming hours in total for an experienced programmer with a good knowledge of the existing bespoke software.

Following the bespoke software's update it must be tested and debugged prior to implementation. The software must also be tested for solid modelling integrity to ensure that all produced drawings have the correct dimensions and positions. The estimated time for all of the testing, debugging and implementation process is 8 programming hours bringing the cumulative time for the total redevelopment and implementation of the software to 28 programming hours i.e. 2.5 working days.

Based on the above estimations, a company considering modifications to their software must decide on the following:

- Should the software redevelopment be performed internally as an internal project, or
- Should the software redevelopment be contracted out to a third party

If the skills to execute the project successfully are available internally, it would be the recommendation of the author to proceed with the first option and redevelop the software internally as it may be possible to implement it quicker and at a lesser cost. Also, if an existing project is paused awaiting implementation of the software it would be possible to develop only the immediately necessary sections in the short term in order to facilitate the project and complete the rest of the redevelopment afterwards.

During the software's redevelopment, it would still be possible to use it in its existing state, however, if it was necessary to start a project with the new frame, it would be necessary to complete the project using manual solid modelling techniques or else wait for the new release of the software.

Chapter 7: Conclusions

This thesis investigates the application of ActiveX technology to customise mid range solid modelling software, with a view to exploring the benefits small to medium sized manufacturing companies can experience from its successful implementation.

Previous independent articles (Shepherd, R., 2003) indicated that companies could gain considerable savings by effectively progressing from 2D CAD systems to 3D solid modelling systems. This thesis investigates also if further productivity gains could be achieved by customising them using ActiveX technology.

The focus of the research concentrated on successfully integrating the necessary technologies to create a custom software application that would automate the design (for manufacturing purposes) of precision air-handling units. Some of the goals set for the customised system include:

- Reduce the time to create manufacturing information by automating its process within the system thus improving efficiency and productivity
- Create solid models of the air-handling units components and assemblies
- Make all solid models associative to one another, thus simplifying the process of design revisions
- Derive all manufacturing documentation from the air-handling unit solid model
- Increase the quality of manufacturing documentation by eliminating common drafting errors associated with 2D CAD drawings
- Investigate the links to a PDM system to manage the design intent of revisions

After researching the various 3D CAD technologies a case study was undertaken to investigate the above goals. Most Irish manufacturing companies in general would have less than two hundred seats of CAD software. Therefore the research concentrated on the custom development of so called mid-range solid modelling systems. A number of the most popular mid-range systems were researched and from them Solid Edge from UGS PLM Solutions was chosen for this research. The rationale for choosing Solid Edge can be summarised as follows.

- The key features and level of functionalities of Solid Edge can be found in most mid range solid modelling systems.
- Solid Edge is customisable and using ActiveX technology, algorithms can be written in either C++ or Visual Basic that facilitate the automation of design tasks
- Solid Edge can manage large assemblies i.e. over 200,000 parts. This capability is more than adequate for the majority of Irish manufacturing companies
- The software contains advanced tools within its own application including the Engineering Handbook, Xpres Route, modern parts libraries and Finite Element Analysis (FEA)

The goals set previously were achieved by successfully integrating a custom written software application using three major technologies

- ActiveX technology
- Data access technology
- Solid modelling technology

By integrating the custom written software application with solid modelling technology, the software was able to access Solid Edge and by using algorithms it could rapidly calculate and construct the air-handling unit solid models. By integrating the custom written software application with Data Access technology, the software was enabled to draw on database information as necessary and apply it to the algorithms that are used to control the process of calculating and constructing the solid models. ActiveX technology was the tool that was used to allow both the solid modelling and Data Access technologies integrate together.

To ensure the software application had a user friendly design and screen flow it was designed with Visual Basic graphical user interfaces. The graphical user interfaces had been primarily designed to perform the following tasks

- Simplify the complex nature and terminology based menus and commands found in the solid modelling software, thus reducing the level of skill necessary to perform design tasks

- Automate the design of manufactured products, thus further improving efficiency and productivity

The software application is designed to construct the air-handling unit solid models in three stages. Each stage of the design and construction concentrates on a specific area of the air-handling unit. The first stage starts with the air-handling unit frame and then progresses onwards to the second stage. The second stage of design and construction concentrates on the air-handling unit's internal components e.g. filters and coils etc. Once the second stage has been completed the third and final stage of the system's design and construction is completed by fitting the exterior panels that seal the air-handling unit to the outer atmosphere.

The resulting system is a flexible automated design system for air-handling units that allow its users to quickly produce air-handling unit designs and solid models. From analysis of the new system its immediate first impressions are

- The speed and flexibility at which it creates solid models of the air-handling unit's components and sub-assemblies
- The software's ability to build a component/sub-assembly outside the main air handling unit file and then insert it into the main file upon its completion, thus improving the speed at which it can create large air-handling unit solid models

After the new system was developed to an acceptable level of functionality, it was necessary to measure the productivity and efficiency of the new system when compared that of the existing one. The results of the tests that were performed are illustrated in Table 5.

7.1 Discussion of findings made in section 6.4.1

7.1.1 Testing the bespoke system

From section 6.4.1, it can be seen that the bespoke system achieved its goals by proving to be superior to the 2D system for producing design and manufacturing information. The advantage the bespoke system has over the 2D system relates to the manner in

which the bespoke system is designed and programmed to automate defined and repetitive functions for generating solid models and manufacturing drawings.

Other key advantages that the bespoke system has over the 2D system are the low level of CAD skills and training necessary to competently and quickly produce air-handling unit designs for manufacture. In contrast, for a user of the 2D CAD system, intensive training and usage of the 2D system is first necessary before a level competence can be reached whereby designs can be produced efficiently and effectively.

7.1.2 Updating the existing bespoke software

From the assessment performed in section 6.4.2, it can be seen that it is feasible to customise and reconfigure the bespoke system. The standard programming interface used and the architecture of software lends itself to aiding this process. The time necessary to perform any modification is always dependant on the amount of programming and reconfiguration necessary to implement the modification and the experience and technical capability of the person(s) implementing the modification.

The estimations made in section 6.4.2, concerning the time required to redevelop the software, are based on the author's knowledge and experience of developing and implementing the bespoke software.

Based on the estimations it is important that a company considering investing in bespoke software understands its boundaries and limitations. It is also important to be informed of the implications when redevelopment and modifications may become necessary. Therefore it is the author's conclusion that a company who invests in a custom software application must develop and retain the following in house:

- The resources used to develop and implement it in order to facilitate future software modifications and upgrades
- A knowledge transfer and training system whereby new staff can be trained to undertake such roles in the event that key resources should become unavailable after the application of the software into operation

7.2 Recommendations for future work

This research has showed that it was indeed possible to automate the design process by linking ActiveX technology with solid modelling systems. This would have considerable benefits to offer a multitude of companies still operating in 2D. The advanced nature of the systems is such that it offers companies flexibility by allowing users to specify a wide range of options about multiple products and at the same it is possible to return immediate manufacturing drawings for the products specified.

With further development algorithms could be written to completely automate all costing aspects of the new system. The author recommends research into the existing MRP systems and the development of algorithms that would link customisable systems to such existing MRP products.

Initial research that was carried out into PDM systems and the conclusion was reached that Solid Edge with its new entity, Insight, could manage the complex number of parts, assembles and subassemblies generated by the custom system. In particular such a system could be used to manage the large number of documents that would be created from revisions generated by the automated system.

7.3 Conclusion

This research has proven that the application of ActiveX technology to developing a customised system for automating a solid modelling software system can provide great advantages to small to medium manufacturing companies in Ireland. By automating the design of their products considerable savings in time can be achieved.

The author hopes that this research will lend itself to a greater understanding of the use of ActiveX technology (OLE programming techniques) to automate design applications using mid range solid modelling systems. It is further hoped, that the systems and solutions developed by this research will offer an insight into possible solutions that can be investigated by other manufacturing companies for their benefit.

References

Andriulli, M., 2002, Solid Edge (Parasolid) vs. Inventor (ACIS/Shape Manager) Kernel Test, UGS PLM Solutions

Cohn, D. (2003), Solid Edge V15 Continues the Design With Insight Vision, Cyon Research Engineering Report

Connell, J., 1998, Beginning Visual Basic 6 Database Programming, Wrox Press Ltd

Date, C. J., 1986, Relational Database: Selected Writings, Addison – Wesley Publishing Company

Date, C. J. and Darwen, H., June 1993, A Guide to the SQL Standard, Third Edition, Addison-Wesley Publishing Company

EDS, 2002, Solid Edge Fundamentals Volume 1, MT01413-120, Version 12

Gott, B., 2003, A white paper, Evolution from 2D to 3D, A Design Engineer's Perspective, Cambashi Limited

Gott, B., 2003, A white paper, Evolution from 2D to 3D, A Senior Managers Perspective, Cambashi Limited

Groff, J. R. and Weinberg, P. N., 1999, SQL: The complete Reference, McGraw-Hill Osborne Media

Kutz, M., 1998, Mechanical Engineers' Handbook, 2nd edition, ISBN 0-471- 13007-9 John Wiley & Sons, Inc

Papazoglou, M. and Valder, W., 1989, Relational Database Management: A Systems Programming Approach, Prentice Hall International (UK) Ltd

Ritchie, C., 2002, Relational database principles, Second Edition, Thomson Learning, ISBN: 0826457134

Roff, J. T., 2001, ADO: ActiveX Data Objects, O'Reilly UK

Roman, S., 1999, Access Database Design and Programming, Second Edition, O'Reilly UK,

Roman, S., 2002, Access Database Design and Programming, Third Edition, O'Reilly UK

Shepherd, R., 2003, Research and Analysis of Engineering Design Productivity: A Research Report

Smith, R. and Sussman, D., 2003, Beginning Access 2000 VBA, Apress

Spatial Technology, Inc., 2000, ACIS 3D Toolkit Technical Overview, Version 6.0, Issue 1.0, ACSTOV060

Sun, T. Y., 1994, Air handling System Design, McGraw-Hill, PART 1 - General, Chapter 7 - Categories of Central Air Handling Systems

Thomsen, C., 2002, Database Programming with Visual Basic.NET, Second Edition, Apress

Unigraphics Solutions Inc., 1999, Overview of PARASOLID

Unigraphics Solutions Inc., 2001, Programmers guide, Customising Solid Edge, MU28000-ENG Version 11

Unigraphics Solutions Inc., 2001, Getting started with Solid Edge, MU28900-ENG, Version 11

Walker, J., 1986, Computer Aided Design: Vertical Market Application, General Purpose Productivity Tool, or the Heart of Computer Science? Revision 1

Williams, C., 1999, Professional Visual Basic 6 Databases, Multi-tier Database Programming, Wrox Press Ltd

WWW References

[1]www

Greco, J., 2003

More than just Mechanical Design?, Desktop Engineering Magazine, Cover Story, http://www.deskeng.com/articles/03/aug/cover/index.htm [Accessed 31 May 2005]

[2]www

Hess, C., 2003

CADserver, Making the Move to 3D - with the marked productivity gains it offers, 3D should no longer be viewed as a luxury but as a necessity, http://www.cadserver.co.uk/common/viewer/archive/2003/Jun/12/feature1.phtm [Accessed 31 May 2005]

[3]www

Mirman, I., June 2003

CADserver, Making the Move to 3D - with the marked productivity gains it offers, 3D should no longer be viewed as a luxury but as a necessity, http://www.cadserver.co.uk/common/viewer/archive/2003/Jun/12/feature1.phtm [Accessed 31 May 2005]

[4]www

CIMdata, 2003

May 2003 version of the NC Software and Related Services Market Assessment, http://www.cimdata.com/press/PR04-0513.htm [Accessed 31 May 2005]

[5]www

Lacey, R., 2000

Customising SolidWorks, CADserver

http://www.cadserver.co.uk/common/viewer/archive/2000/Nov/21/feature3.phtm [Accessed 31 May 2005]

[6]www
UGS PLM Solutions Inc., 2002
Integration and Customisation
http://www.solid-edge.co.uk/pages/sect01_p01.htm [Accessed 31 May 2005]

[7]www
CAD/CAM Publishing, Inc., 2003,
EDS overhauls Insight PDM aid
http://www.cadcamnet.com/ [Accessed 31 May 2005]

[8]www
UGS, 2004
Solid Edge Overview
http://www.solid-edge.com [Accessed 31 May 2005]

[9]www
Microsoft Corporation, 1999
How to Write and Use ActiveX Controls for Microsoft Windows CE 2.1.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce21/html/activexce.asp [Accessed 31 May 2005]

[10]www
Pizzo, M. and Cochran, J., 1997
OLE DB for the ODBC Programmer, Microsoft Corporation, http://msdn.microsoft.com/ [Accessed 31 May 2005]

[11]www
Rauch, S., 1997
Manage Data from Myriad Sources with the Universal Data Access Interfaces, Microsoft Systems Journal
http://www.microsoft.com/msj/0997/universaldata.aspx [Accessed 31 May 2005]

[12]WWW

Microsoft, 2004

MSDN Library Home, Data Access, Microsoft ActiveX Data Objects (ADO), ADO Programmer's Guide

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/pg_introduction.asp [Accessed 31 May 2005]

[13]WWW

Microsoft, 2004

MSDN Library Home, Visual Basic 6.0, ADO, DAO and RDO in Visual Basic,

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconusingadodaordoinvisualbasic.asp [Accessed 31 May 2005]