AN INTERFACE TO FACILITATE DATA FLOW IN THE
INTELLIGENT MACHINING WORKSTATION

A Thesis Presented to
The Faculty of the
Fritz J. and Dolores H. Russ
College Of Engineering and Technology
Ohio University
In Partial Fulfillment
Of the Requirement for the Degree
Master of Science

By
Dilip Viswanath

June, 2000
# Table of Contents

Table of Contents ...................................................................................................................... iv

Table Of Figures ...................................................................................................................... vii

1 Introduction ............................................................................................................................. 1

1.1 Tools to Support Concurrent Engineering ............................................................................. 2

1.1.1 PDM Systems ..................................................................................................................... 2

1.1.2 IDS ....................................................................................................................................... 5

1.1.3 IMW ..................................................................................................................................... 7

1.2 Description of the Problem .................................................................................................... 10

1.3 The Thesis Statement .......................................................................................................... 11

1.4 Layout of This Thesis Report ............................................................................................... 12

2 IMW: The Background .......................................................................................................... 13

2.1 The IMDE Architecture ......................................................................................................... 13

2.2 Tools in the IMW ................................................................................................................... 15

2.2.1 PART ................................................................................................................................. 16

2.2.2 Unigraphics ........................................................................................................................ 18

2.2.3 Virtual NC .......................................................................................................................... 19

2.2.4 COST ADVANTAGE .......................................................................................................... 21

2.3 The Translation Environment ............................................................................................. 22

2.3.1 The STEP Standard ......................................................................................................... 22

2.3.2 TNR – A Tool to Facilitate Data Translation .................................................................... 24
5.4 Exporting Data from Unigraphics ......................................................... 65

6 Conclusions And Recommendations .................................................... 67

Bibliography .................................................................................................. 69
List Of Figures

Figure 1. PDM Systems [5] ................................................................. 5
Figure 2. IDS System [27] ................................................................. 7
Figure 3. Tools and Components in the IMW [1] ............................... 9
Figure 4. The IMDE Architecture [8] ........................................... 15
Figure 5. The TNR Translation Environment [9] ............................. 25
Figure 6. The IMW GUI Main Window ........................................... 43
Figure 7. File and Utilities Pull Down Menu Options ....................... 45
Figure 8. The IMW Main Window at the Time of Invocation ............. 55
Figure 9. The Sample Part ............................................................. 56
Figure 10. The Main Window for PART ......................................... 58
Figure 11. The Main Window at the Time of Data Translation .......... 59
Figure 12. The Main Window After Data Has Been Successfully Translated ... 59
Figure 13. The Main Window for CA ............................................. 60
Figure 14. New CA Library being Created .................................. 61
Figure 15 The Confirm Dialog Box .............................................. 62
Figure 16. The Main Window After CA Data Translation .................. 62
Figure 17. Cost Estimates For “steped_hole” as Seen In the CA Window .... 63
Figure 18. The Main Window for VNC ........................................... 64
Figure 19 The Select Dialog Box .................................................. 64
Figure 20. The VNC “Ask” Dialog Box ................................................................. 65

Figure 21. The GUI Main Window for UG ............................................................ 66
1 Introduction

Today's highly competitive global market economy, fueled by rapid advances in technology and an ever-changing customer needs, demands high quality products at a low price in the quickest time from companies competing in the market. These companies are faced with shrinking "time-to-market" for their new products, the emphasis being on reducing the elapsed time between product conception to its actual availability in the market for sale. A product goes through several stages of development that collectively define the product life cycle. To maintain its competitive advantage in the global market, a company should be able to continuously innovate, both the product and the processes in the life cycle of the product, in order to respond quickly to the market changes and reduce the risk of failure. Concurrent engineering is a stated tool in achieving this objective.

Concurrent engineering can be defined as the earliest possible simultaneous work of experts from various functions in an enterprise, concerned with producing a specific product, in order to achieve high quality, functionality and manufacturability in the shortest time, for a minimum cost. Concurrent engineering is primarily an expression for the desire to increase the competitiveness by decreasing the product lead time, while improving the product quality and cost.

To support and improve the product development lifecycle, a large number of different software tools have come into existence to facilitate particular tasks in the product life cycle. However, these different software tools are often restricted to specific
tasks and have difficulty in communicating with one another. In order to realize products that comply with customer demands and can be manufactured at reasonable cost, it is important that these computer tools are provided with the ability to communicate the right kind of information. The success of a concurrent engineering strategy depends on the coordination, cooperation and communication within and between the computer tools used in the different stages of the product life cycle. Efforts are, thus, being made, through the development of integrated systems, to implement a concurrent engineering environment able to cover all phases of product development.

1.1 Tools to Support Concurrent Engineering

Tools have been made available to manage the heterogeneous data of different tools so as to provide an integrated product development system. A few have been described in some detail below.

1.1.1 PDM Systems

The process of product development involves many different computer systems creating different types of product data. Though the systems often form very good tools for their specific tasks, the lack of integration between the systems results in a weak exchange of information and an ineffectual data management. The failure to manage information effectively can have a devastating effect on a company's competitiveness, as most of the product life cycle is spent not on developing the product but on looking for and verifying data [10]. Thus, the desire to improve the product development process and as a result, the engineering productivity by the efficient integrated management and
exchange of product information has led to the introduction of Product Data Management (PDM) systems.

“A PDM system is a software framework that enables manufacturers to manage and control engineering information, specifically, data surrounding new product designs and engineering processes”[5]. PDM systems aim to provide an integrated product development method so as to produce products with higher quality and lower costs in a shorter lead time.

It is evident that for the success of a concurrent engineering approach in manufacturing, cross-functional design teams with the associated data must be brought together. A PDM system gives product design team members the opportunity to work with an integrated product and project data model containing all product and project documentation, independent of the system it was created in and a faster and earlier release of information. These features of a PDM system are assumed to contribute to a faster product development cycle and thereby a reduced time-to-market.

PDM systems support the product development process at two levels – the task level and the project level. From the task perspective, a PDM system can help organize design revisions, track versions of an evolving design concept and retrieve archived data and other product-specific information. From a process perspective, a PDM system can orchestrate procedural events such as design reviews, approvals, and product release. PDM functions can be summarized into user functions and utility functions. User functions include tasks such as design release management, change management, product structure management, program management, and classification. Functions such as
communications and notifications, data transport, data translation, image services, and administration constitute the utility functions. By organizing, tracking and controlling access to product information as it is created, a PDM system facilitates a team-oriented approach to product development, because it manages data as it is reviewed, modified, approved and archived.

A PDM system integrates data from different types of hardware and software applications in a manner such that the product and project information can be located and accessed quickly. The PDM system stores product and associated information in a database. The system can be easily customized to define various ways of addressing data, configurations, audit trails and computer interfaces through the centralized data repository. The management of the entire process of product development is improved by providing an advanced query capability for accessing data via common attributes and increasing data security. By overseeing application-level tasks e.g. checking and tracking design revisions, authorizations, and drawings, a PDM system provides procedural control over the design methodology by facilitating approvals and notifying the current project status. As a result, a PDM system provides the improved coordination in a project necessary to implement concurrent engineering and compress the overall product development cycle. Figure 1, provides a functional view of a typical PDM system.
1.1.2 IDS

Many problems in the product development process are accentuated by a poor flow of information among the various stages in the product life cycle. In the design of a product, the design engineer is concerned with meeting the functional requirements of the product. The manufacturing engineer is left with the task of producing the product as designed, or negotiating the changes in the design when the part cannot be manufactured according to design specifications. The result is a redesign process, which not only results in longer lead times but also escalation in product and project costs. An approach in overcoming these shortcomings and producing "manufacturable" designs is the Intelligent Design System (IDS) [27], developed at the Georgia Institute of Technology.
In its simplest terms, IDS merges a CAD system, an expert system, and a database management system. The various components of the IDS system are linked through a central controller program. The IDS represents information in a standard form, so as to assure efficient and accurate data transfer between the system components.

The information requirements of the IDS can be divided into three broad categories -- CAD data, the design catalog and the knowledge base. CAD data pertains to specific information concerning the geometric, the material, and the manufacturing properties of the object being designed. Design catalog is the information from the vendor catalogs and handbooks that are an established part of the engineering design process. The rules and heuristics concerning the design and the manufacturing methodologies constitute the knowledge base of the IDS. The IDS system provides the product designer with the ability to select items for design from the design catalog, as well as to design custom items using the CAD system. The designs are then stored in the IDS database and checked by the expert system for manufacturability and assembly.

The layered object model of the IDS controller provides a standard design information model to reduce the difficulties in integrating CAD systems, expert systems, and databases. The model captures design information in a structure that interacts with the design process and the expert system's heuristic modules. An interface, consisting of a set of expert system functions, between the expert system and the database, facilitates the extraction of data from the database by the expert system. The interface also provides the functionalities of checking, deleting and writing of information to the database. The
majority of the expert system rules are placed in the database providing extensibility to the knowledge base.

The IDS, through its components (shown below in Figure 2), seeks to bridge the gap between manufacturing and design, and increase a designer's productivity and creativity.

![Diagram of IDS System](image)

**Figure 2. IDS System [27]**

1.1.3 IMW

The product development life cycle is complex and problematic as it involves the integration of multiple systems designed and optimized by different designers and because it is “interdisciplinary in approach, asynchronous in operation, and constantly
evolving” [8]. This results in long product development times, as the product requirements are not well understood in the beginning, and also as the requirements may constantly change in the product life cycle. The Intelligent Machining Workstation, or the IMW, seeks to provide an extensible integration technology that enables the systematic integration of new systems and coordinates the design efforts of multidisciplinary design teams, thus providing technology for rapid prototyping of new products.

The IMW, developed at Ohio University (OU), provides a powerful concurrent engineering environment for design and manufacturing engineers [1]. The approach adopted in developing the IMW was to integrate the best commercially available tools onto a single platform; the actual integration itself involving the development of translators to provide for integration of the tools. Figure 3, below, is a diagrammatic representation of the tools and the data flow among these tools in the IMW.
In its current form, the IMW provides for the generation of process plans and simulation and cost estimation of the machining process. The IMW uses Tecnomatix’s PART to generate the prismatic process plans, Unigraphics (UG) to develop solid models and turning and milling process plans, Cognition’s Cost Advantage (CA) as a cost estimation software, and Deneb Robotic’s Virtual NC (VNC) as a CNC simulation package. The above four tools were chosen for integration into the IMW to provide for a reasonable cross-section of fairly complex tools in a typical manufacturing organization.
A typical concurrent engineering approach in utilizing the IMW would be to first generate the solid model of a product using UG. The process plans for the machining process could be made available either from UG itself or from a typical run of PART to identify the machining features, to create proper setups, to determine the machining operations, to select the appropriate tooling, and to generate the tool paths. The IMW allows the user to move the data previously generated into VNC so as to execute the machine model for each setup and to verify the process plans. Reasonably accurate cost estimates can be developed in CA to arrive at the most competitive design and process plans.

1.2 Description of the Problem

The tools in the IMW environment depend on information generated by other tools in the environment. A process plan cannot be generated by PART without a solid model being made available to it by UG. Similarly, VNC utilizes the machining operations, tooling and tool paths described by PART or UG to simulate the machining process. Cost estimates are arrived at by CA; on basis of machining features identified in the process plans. The IMW supports the tools necessary to translate data between the different tools in its environment.

However, the success of concurrent engineering in IMW depends, in a large measure, on a logically organized flow of information between the various tools in the platform. It is imperative that right information is made available to the right tool at the right time and in the right form. The information flow in the IMW, thus, necessitates the
development of a mechanism to organize access and then control the flow of data in its environment.

1.3 The Thesis Statement

The aim of this thesis is to design and develop a Graphical User Interface (GUI) for the IMW to:

- Coordinate the data flow among the various tools in the IMW platform.
- Provide the user with a “handy” tool to communicate requests and to review the results of a request of the IMW.

The thesis will accomplish these aims through the steps enumerated below.

1. Identify the issues in the development of a GUI for the IMW. The GUI should provide the user, the appropriate means to access the various tools and their respective data in the IMW environment. The flow of information should be coordinated by the GUI in a manner that facilitates a concurrent engineering design.

2. Identify an appropriate tool for interface development. As the operating environment of the IMW is UNIX, the evaluation and the subsequent selection of an appropriate tool for building the IMW interface is restricted to those available in the UNIX operating system. A few factors that could influence the evaluation methodology are ease of availability, ease of learning and ease of use.

3. Implement the graphical user interface. The development of the user interface should be followed by a thorough evaluation of the system for its user friendliness and robustness.
1.4 Layout of This Thesis Report

Chapter 2 provides a background on the research leading to this thesis. In particular, a discussion on the IMW is presented. The implementation architecture for the IMW is outlined and a synopsis of each of the four tools currently integrated into the IMW environment is also presented. General issues in the development of a graphical user interface are enumerated in Chapter 3. The chapter also looks at issues specific to the development of the IMW GUI. Chapter 4 describes the actual implementation of the IMW GUI. The chapter provides an overview of the GUI and describes the modules and procedures in the GUI implementation. In chapter 5, the usage of the IMW GUI is demonstrated with the help of a sample part. In chapter 6, conclusions and recommendations are made.
2 IMW: The Background

This chapter expands on the IMW system outlined in some detail in chapter 1. The idea is to familiarize the reader with the integration environment that provides the basis for this thesis. The major sections in this chapter describe the IMW IMDE architecture, the tools presently integrated into IMW and the translation environment of the IMW.

2.1 The IMDE Architecture

Typical manufacturing design is a complex and a serial process with long development times and often conflicting and non-functional requirements being furnished to design engineers. A concurrent approach to product design process has been suggested to facilitate coordinated multi-disciplinary design efforts with the aim of making the design process less serial. The IMDE model was proposed as a new concurrent engineering technology to enable prototyping of complex products as well as “systematic integration of new tools” [8] in an extensible environment.

The IMDE architecture has three components – the unified data meta-model (UDMM), intelligent interfaces and message board. The approach in integrating these components is similar to a distributed database [8]. The integration is between relational and non-relational data in an environment where there are no links previously defined
between the entities in the multiple databases. The functioning of the three components of the IMDE model are briefly discussed as below:

**UDMM.** A UDMM contains the types of entities, relationships among the entities, and constraints that are commonly shared among the methods used for a product design. A UDMM defines a standardized data format for translation, such that it captures all data common to more than one tool or method in the IMDE. Individual tools will have data that is not specified in a UDMM, as necessary, to perform their particular functions. There will be a different UDMM for each implementation of the IMDE model capturing all information that overlaps between the independent tools in that particular implementation.

**Intelligent Interfaces.** Each tool in the IMDE model must have its own intelligent interface. The intelligent interface facilitates the exchange of data between various tools in the model. To this end, each intelligent interface “understands its tool’s local data and the tool’s relationship to the UDMM” [3]. The intelligent interface with its knowledge of relationship between local objects and the UDMM translates from the local format to a neutral format specified by UDMM. When information from a remote tool is desired, a link is established and data from the remote tool is received in the neutral format. The intelligent interface then translates the data into the format of the local tool.

**The Message Board.** Any irresolvable constraints, links or errors are posted on the message board providing the designer with the ultimate control over the domain and model.
Notable features of the IMDE architecture are as follows:

1. The IMDE architecture coordinates all activities in a hierarchical environment that allows a designer to work with familiar tools.

2. All data is allowed by the architecture to be stored by the creating tools in the format specified for that particular tool.

3. The IMDE architecture is flexible in the sense that tools can be added or removed.

The IMW is an implementation of the IMDE architecture.

Figure 4 presents the IMDE architecture along with its components.

![Figure 4. The IMDE Architecture [8]](image)

### 2.2 Tools in the IMW

Using the IMDE model as the basis for the integration architecture, four tools have been integrated into the IMW platform. The following sub-sections examine each of the tools and describe the working methodology of these tools.
2.2.1 PART

PART, by Tecnomatix, is an automatic process planning system, generating process plans for previously defined products. A product is a basic entity for which process planning can be performed. Within PART, a product is considered to consist of two geometric 3D models (one describing the blank; i.e. the initial state of the product before machining, the other describing the part; i.e. the final state of the product after all machining operations have been performed), and a material specification [20].

The part and blank models, which are used in the products, are 3D ACIS solid models (ACIS is the kernel modeler used within PART). Before the user can perform process planning, 3D models of both – the actual part and the blank – are needed. Such models have to be created outside PART and can subsequently be imported. PART provides support for the import of 3D solid models from standard ACIS based solid modelers such as PRO/Engineer and ISO STEP.

PART allows the user to formulate a process planning strategy by defining several process-planning scenarios. A process-planning scenario, in PART, consists of several sub-scenarios, each of which constitutes a part of the complete process plan [20]. Examples of sub-scenarios include basic process planning activities like Setup Selection, Machine Tool Selection, Method and Tool Selection, Tool Path Generation and NC Output generation. A scenario is the specification of the sequence in which the main process planning functions (sub-scenarios) have to be executed to perform the automatic generation of process planning results.
The concept behind scenarios in PART is the fact that different companies have different process planning strategies depending on their product mix, the machines in their shop and their standard way of working. A “cutting tool” oriented company might have a large number of identical tools in its shop. Their standard way of process planning may be to perform the method and tool selection first and subsequently assign the job to the machine, which already holds the largest number of required cutting tools. A “machine tool” oriented company might have several rather different tools in its shop (differing in power, number and types of cutting tools, which can be used on the machines, etc.). The typical way of process planning in that company may be to perform the setup and the machine tool selection first and then select the machining methods and cutting tools to be used on the previously selected machine tool. The behavior of PART can, thus, be adapted to the normal way of working within a given company.

After a product has been created, a process-planning scenario has been selected and the system settings have been set, the process plan can be generated. PART stores its internal data in an ORACLE database. In the IMW, this data is extracted from the ORACLE database using SQL queries and stored in STEP\(^1\) format. This STEP format is used to exchange information among the other tools in the IMW.

---

\(^1\) The STEP format is detailed in a later section in this chapter.
2.2.2 Unigraphics

Unigraphics (UG) is a powerful tool for CAD/CAM applications. The functionalities offered by the application can be categorized into two types – design and manufacturing. Unigraphics is divided into the four broad applications outlined below.

Drafting. This application provides for basic drafting capabilities and facilitates automatic generation of drawings while the designer works on the model.

Modeling. Solid modeling and other modeling options like feature modeling and free form modeling are possible with this application.

Assembly. Parts can be assembled in this integrated UG application. Changes made to a part are reflected in all assemblies containing the part. Components can be assembled using either a top-down or a bottom-up approach.

Analysis. Finite element analysis, kinematical analysis and design are some of the features offered by this application.

Modern solid model based computer aided design allows engineers and designers to develop virtual parts in 3D space. The solid model allows the design to be shared with CAM, FEM, CAPP and other applications. Modeling is one of the major applications of Unigraphics, providing the capability to perform conceptual design with swept features, form features, primitives and boolean operators.

Using solid models, Unigraphics supports the creations of tool-paths for turned parts and surface milling. Unigraphics provides manufacturing options for turning, milling, threading, tapping, and drilling. The manufacturing module in Unigraphics is organized into the Operation manager, the Tool manager and the Cutter Location Source
File (CLSF) manager. Operations are chosen from the operation manager, tools are either created or chosen from the tool manager and the sequence of operations is edited in the CLSF manager.

The Operation Manager provides the option to create or edit the tool path. Once operation details like tool, drive geometry, part geometry, blank geometry, check geometry, control geometry, the tool engage and retract positions and the machine parameters are specified, the tool path can be generated. With the tool path defined, the operations are exported to create a Cutter Location Source (CLS) file. The CLSF manager is used to select, edit and review the entire CLS file. It contains a series of GOTO moves and other tool path descriptive data. The CLSF manager also gives the tool path actions option that helps to manipulate the tool paths in the CLS file. The Tool Manager provides a list of all valid tool types in the Unigraphics.

In the IMW, solid models of the part to be machined are generated using UG. The tool paths for the machining process are created and stored in a CLS file. This information is then made available to other tools in the IMW via a STEP file.

2.2.3 Virtual NC

Deneb Robotic's Virtual NC (VNC) is a prototyping tool that provides a simulation environment capable of emulating an entire machine and the actual NC controller, saving valuable machine time for tool path verification [4]. The user creates a set of geometric models for the machine being simulated, assigning translational and rotational motion to the machine and placing limits of travel over, and around the various axes of the machine. An emulator for the NC controller is written in a special language
called MIMIC to manipulate the machine-model to respond to NC instructions. Part blanks and tool-sets can be created using built-in modeling facilities of VNC and the NC programs can be run to virtually machine a part.

VNC may be used to verify the process plans generated by process-planning tools such as PART and UG. The NC programs created in the generation of process plans for a component can be loaded on to the virtual machine and machining process simulated.

VNC is divided into three primary systems -- the VNC Menu System, MIMIC and CLI. The VNC menu system provides for the creation of three-dimensional visual representation of devices, which include work pieces, tooling, machines, pallet changers etc. These devices can be assembled into operational machines and work cells. A specific NC controller’s functionality can be imitated through instructions and assignments in the MIMIC configuration file. MIMIC has features to monitor the machining cycle. The Command Line Interpreter (CLI) is a communication, command and control system for accessing and operating VNC. The CLI provides for two modes of communication with the VNC. The stream mode of the CLI allows a text file to be used as input as well as direct output to an output file. The socket mode allows for any external program to invoke VNC and communicate with it, using CLI commands and return codes.

To simulate the machining process in VNC, it is required that the geometric model of the machine being simulated together with the tool assembly setups for the machining process be supplied by the user. The IMW maintains a list of manually created machines copied into the current VNC library as per the data in the UDMM. The
tool assemblies are created on the fly from the data in the UDMM using the CLI interface. The part blank and NC programs generated using PART or UG to convert the blank model into the finished component are now read into the VNC from the STEP file to perform the simulation.

2.2.4 COST ADVANTAGE

"Cost Advantage is a knowledge-based software system that provides expert-level design guidance and can analyze manufacturing alternatives, producibility, and predictive cost analysis" [2]. Cost Advantage (CA) consists of the following four components:

Cost Modeler – Existing cost models are modified or new cost models are built, using the CA modeler.

Process Model – The knowledge of manufacturing specialists is encapsulated in the process models.

Cost Advantage – The cost models created are used to analyze designs and predict the cost before manufacturing.

Notes – All the cost information created in Cost Advantage is stored in cost notes.

A cost or process model is a set of equations and design rules used to calculate the cost and producibility of manufacturing a part. For example, a stamping model might store the variables, equations, and limitations for the available stamping machines. It might also use a materials database and a description of design features that can be stamped. A cost-model can be built using the cost-modeler. A cost model has four cost drivers – process, material, feature and parts. The analysis of design is driven by the choice of a manufacturing process, the material to be used, contributing design features,
and added parts. A hierarchical tree of specializations is defined in the cost-model, which is used by cost-advantage to generate cost-notes for specific components.

The description of features and their characteristics are read from a text file called a design file. The cost-notes are the results of analyzing a design with a particular cost-model.

Cost notes are created in IMW using the cost model for the IMW. Data is read in from PART or UG via the IMDE architecture and is converted to CA features. Setups are entered into the cost note to represent setups. While a sizeable amount of data is transferred “automatically”, there are always certain gaps in the information, which can only be filled in by the user of the tool. However, the IMDE architecture facilitates the speedy migration of data from the tools.

2.3 The Translation Environment

Data translation in the IMW environment has been made possible by the availability of a neutral format for the representation of data and by the implementation of a translating tool to provide for the exchange of information between the neutral format and the private data of a tool. The neutral format specified by the UDMM is implemented in the standards of STEP.

2.3.1 The STEP Standard

STEP, Standard for the Exchange of Product Model Data, is a comprehensive ISO standard (ISO 10303) that provides a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged. The
exchange is among different computer systems and environments associated with the complete product lifecycle including design, manufacture, utilization, maintenance, and disposal.

As any product data representation should contain information about the product's entire life cycle, STEP has been constructed as a multi-part ISO standard with each part being published separately. These parts cover areas such as testing procedures, file formats, programming interfaces and industry-specific information. STEP uses a formal schema specification language, EXPRESS, to specify the product information to be represented. The use of a formal language enables precision and consistency of representation and facilitates development and implementation. The most important aspect of STEP is extensibility. STEP uses application protocols (AP) to specify the product information for one or more applications. These application protocols form the bulk of the standard and are the basis for STEP product data exchange. In addition, the EXPRESS language can document constraints as well as data structures. These formal constraints are an explicit correctness standard for digital product data. Some of the relevant parts of the standard are listed below:

**Part 11.** Product Data Exchange using STEP (PDES) – The EXPRESS language reference manual [6].

**Part 21.** Product Data Representation and Exchange – Clear text Encoding of the Exchange structure [7]. The data stored as per an EXPRESS schema needs to be communicated across tools using the Part 21 scheme for encoding data as clear
text. Part 21 uses a system of unique object Id's assigned to each object, which may be included in instances of other objects that reference this object.

The overall objective of STEP is to provide a mechanism that is capable of describing product data throughout the life cycle of a product, independent of any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving data. The ultimate goal is an integrated product information database that is accessible and useful to all resources necessary to support a product over its lifecycle.

2.3.2 TNR – A Tool to Facilitate Data Translation

TNR, an acronym for “Templates and Rules”, is a language developed at Ohio University’s Center for Advanced Software Systems Integration (CASSI). TNR bears a strong resemblance to “C” which is a great aid in learning to use the language. It combines compiling and execution into one step.

The primary objective in developing TNR was to provide a mechanism to extract data from complex resources and to put data into other data resources [9]. TNR supports two fundamental procedural types, rules and templates, to facilitate the two major kinds of jobs (input and output of data). Templates and rules implement the “write” and the “read” context in TNR. In the template context, file-objects are treated as output devices while in a rule context file-objects are treated as input devices. Templates and rules provide for logical separation of tasks of data retrieval and that of data insertion into tools in the IMW. A minimum TNR program consists of just one procedure called “main”. The “main” procedure can either be of either type – a template or a rule. As in “C”,
procedure calling in TNR can be done by two means – call-by-value and call-by-reference. A schematic representation of the TNR data translation environment is provided in Figure 5.

<table>
<thead>
<tr>
<th>Source</th>
<th>Formats</th>
<th>Interfaces</th>
<th>Conversion Recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>SQL, EQL, EXPRESS</td>
<td>FILE</td>
<td>Template</td>
</tr>
<tr>
<td>Proprietary</td>
<td>DXF, IGES, etc.</td>
<td>SOCKET</td>
<td>Rule</td>
</tr>
<tr>
<td>Structures</td>
<td>Inter-process</td>
<td>TNR</td>
<td></td>
</tr>
<tr>
<td>Networked</td>
<td>Communication</td>
<td>Translation Environment</td>
<td></td>
</tr>
<tr>
<td>Processes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. The TNR Translation Environment [9]

TNR supports two types of variables – scalars and vectors. Scalars are "typeless" data types in the sense that the same scalar may exist as a real, integer, or a string in its "lifetime”. Vectors are defined as sequences of scalars that grow or shrink on demand. Vectors in TNR can be considered to be analogous to arrays in “C”.

To facilitate data translation, TNR provides several pre-written methods for communication with different information resources such as relational databases, STEP files, ASCII text, and network resources. These pre-programmed methods can be used in TNR programs to interact with data resources to find and to manipulate data in the resources. TNR, thus, provides the translation environment for the IMW.
3 User Interface Issues

"A user interface (UI) of a computer program is the part that handles the output to the display and input from the person using the program" [22]. The user interface includes all aspects of communication between the users and software. According to Powell [16], a user interface should:

- Lead the user, step by step, through standard sequences of operations.
- Warn the user of potential problems and errors.
- Enhance productivity, for example, by streamlining options or arranging them in logical groups.

Though it cannot turn a mundane task into exciting one, a user interface can make the job more pleasant and require less effort.

Siegfried [22] has identified three factors that are pertinent in the design of a Human-Computer Interface (HCI). According to him, "the realistic capabilities and limitations of each of the user, the computer-based application, and the computer, especially with regard to its user interface, become factors that the user must recognize". The three stated factors are described below:

**User Factors.** The factual data about an individual user or user group can be stratified as background factors (e.g. experience, skills), physiological factors (e.g. physical and cognitive factors like vision, dexterity, memory), and application interest and use factors (e.g. choice, goals, frequency). These factors are subject to being
confirmed, counter-acted, or contrasted by either user opinions or relevant performance data, obtained for the target user in question.

**Application Factors.** The design factors that reflect the nature and complexity of an application can be derived from the analysis of the application into its constituent tasks and task patterns. The application factors can be categorized into those impacting on the user's computer such as the required data storage and processing support and those impacting on the user such as required human memory and mental processing support.

**Interface Factors.** The interface factors can be enumerated as software factors (in terms of programming languages, operating systems and environments), hardware factors (CPU speed, storage capacity, architecture), Interface factors (in terms of both software e.g. interaction language and hardware e.g. interactive tools), and application suitability factors to represent the type of application for which the computer is well suited.

Weinschenk et.al [25] have stated that the whole process of interface development could be divided into three phases – analysis, design, and construction. In the analysis phase of the interface development, an assessment is made to get a complete “picture” of requirements for the interface. The task includes the development of user profiles and usability specifications for the proposed interface. The design phase builds on the analysis phase, utilizing the information obtained from analysis and applying it to shape the interface. This phase involves the design of interface objects that the user will manipulate and the specification of actions that will be performed on those objects. The
construction phase is the actual construction of the interface. The steps in the construction phase include creating a prototype of the interface, testing the prototype thoroughly for its usability and carrying out revisions based on the results obtained from the testing process.

This chapter constitutes the first phase of the IMW interface development. The various aspects of design of an interface to IMW are analyzed and discussed. The following sections in this chapter evaluate the various tools available in the development of an interface and identify the issues underlining the requirements of the IMW interface.

3.1 Tools for Interface Development

User interface software is often large, complex, and difficult to implement, debug, and modify. The development of an interface demands that the programmer deal with the problems of elaborate graphics, multiple ways of giving a command, multiple asynchronous input devices, and rapid “semantic feedback” with appropriate response to user actions. Because user interface design is so difficult, special software systems and tools have been made available to help design and implement the user interface software. These user interface tools are variously named as User Interface Management Systems (UIMS), Toolkits, Interface Builders, Interface Development Tools, and Application Frameworks.

According to Myers [26], the use of user interface software tools is expected to result in the following:

1. Increase the quality of interfaces. Interface tools allow for rapid prototyping and implementation of different interface designs. They also make it easier to incorporate
changes necessitated through testing. The interface development tools also provide for a variety of specialists such as graphic artists, cognitive psychologists, and human factor specialists to be involved in designing the interface, rather than having the interface created solely by programmers.

2. Improve the user interface code. Interface specifications can be represented, validated, and evaluated more easily through interface builders. It becomes easier and more economical to create and maintain the user interface code, as most of the code is supplied by the user interface tools. This also leads to an increased reliability of the user interface. The level of programming expertise of the interface designers and implementers can be lower, because the tools hide much of the complexities of the underlying system.

3.1.1 Scripting Languages as Interface Development Tools

A notable development, in the field of computer programming, has been the evolution of scripting languages over the last few years. Computer programming languages are, now, effectively categorized between scripting languages and system programming languages. The scripting languages differ fundamentally from system programming languages as they are designed for different tasks than the system programming languages [12]. System programming languages are designed for data structures and algorithms from scratch, starting from the most primitive computer elements such as words of memory. In contrast, scripting languages are intended primarily for connecting a set of pre-existing components together. Scripting languages and system programming languages are typically used together in component
frameworks, where components are created with system programming languages and glued together with scripting languages.

The system programming languages are usually strongly typed. The term "type" refers to the degree to which the meaning of information is specified in advance of its use. This necessitates that each piece of information being used be previously declared and defined. The language prevents the information from being used in ways other than that are appropriate for the definition. For example, a variable previously declared as a string in a system programming language can hold only a string in any instance of that variable. The data and code are totally segregated making it difficult to generate new code on the fly.

Scripting languages, on the other hand, offer a "higher level" of programming than the system programming languages with many details handled automatically by the language itself, requiring a lesser amount of code to get the same job done. Scripting languages follow a type less approach to achieve this "higher level" of programming in developing applications. In a weakly typed or type-less language there are no restrictions on how information can be used. For example, the same variable can hold a string in one instance and an integer in the next. The type-less nature of the scripting languages simplifies the task of connecting components together. The components look and behave in a uniform fashion so that they are interchangeable.

The strongly typed nature of system programming languages discourages reuse [12]. Typing encourages programmers to create a variety of incompatible interfaces. Each interface requires objects of specific types. In order to use other types of objects
with an existing interface, conversion code must be written to translate between the type of the object and the type expected by the interface. The type-less nature of the scripting languages results in less code and more flexible programs.

Another key difference between scripting languages and system programming languages is that scripting languages are usually interpreted whereas system programming languages are usually compiled. Scripting languages, thus, provide rapid turnaround during development by eliminating compile times. Scripting languages also make applications more flexible by allowing users to program the applications at runtime. Scripting languages sacrifice speed to improve development speed. To sum it all, scripting languages are ideally suited for gluing applications and can be appropriately referred to as "languages of system integration" [12].

Discussed briefly in the following sub-sections are two leading scripting languages.

3.1.1.1 PERL

Primarily used for scripting, “Practical Extraction Report Language” (PERL) was written to manipulate text in files, extract data from files and perform many networking tasks [24]. PERL is an interpreted language and supports run-time evaluation. The language is intended to be practical (ease of use, efficient, complete) rather than beautiful (tiny, elegant, minimal). Optimized for scanning, PERL uses pattern-matching techniques to scan large amounts of data very quickly. PERL does not put any arbitrary restrictions on the size of the data; data structures such as strings and arrays can grow as large as they like. Scripts written in PERL are portable. As is true with most scripting
languages, PERL allows for rapid design, development, and deployment of applications and also for extension of their functionality as and when such a need arises. Unlike most scripting languages however, PERL is not a strictly interpreted language. It does not combine compiling and execution in a step. The source code is, first, compiled by the PERL compiler into an intermediate format and in the process the program is checked for various syntactic and semantic errors. On successful compilation of the program, the intermediate code is passed to the interpreter for execution. However, an outstanding feature of PERL is that it provides a data tracing mechanism to trace data from insecure sources and thus guard against accidental security errors. PERL also provides for specialized compartments in which PERL code from dubious sources can be safely executed. This feature of PERL has endeared itself to system administrators.

3.1.1.2 TCL

TCL stands for “Tool Command Language” [13]. TCL is actually a scripting language and also an interpreter for that language that is designed to be easy to embed in application programs. TCL provides a “parser for simple textual command language, a collection of built-in utility commands, and a C interface that tools use to augment the built-in commands with tool-specific commands” [13]. As a scripting language, TCL provides enough programmability (variables, control flow, and procedures) to help build complex scripts that assemble programs into a tool. TCL is really intended to be embedded in application programs. By adding a TCL interpreter, an application can be structured as a set of primitive operations composed by a script suited to the needs of the user.
TCL is particularly useful in a windowing environment as it provides two advantages. First, TCL can be used as a general-purpose mechanism for programming interfaces the interfaces of applications. Second, TCL provides a uniform framework for communication between tools. In a windowing environment, TCL can be used to configure the “application interface actions”, and also to configure the “application interface appearance”. The primary use of TCL as far as windowing applications are concerned is for interface actions. Each event associated with the application is bound to a TCL command and executed by passing it through the TCL interpreter. The other use for TCL in the windowing environment is to configure an application interface. The various interface components are configured and reconfigured using TCL commands to achieve the desired layout and the appearance for the application being developed. A notable extension to TCL for use in a windowing environment is TK [15], a toolkit for developing graphical user interfaces. TK follows a script-based approach to interface programming, providing a rapid development time for applications. TCL with the TK [11] toolkit is ideally suited for developing graphical user interfaces (GUIs).

3.2 IMW Interface Issues

Presently in the IMW, there is a unidirectional flow of information from PART to CA, from PART to VNC, and from UG to VNC. This flow of information is effected by translators that facilitate the transfer of data from within PART and UG to a local store and then from the local store into applications VNC and CA for data generated by PART, and into VNC for the information made available by UG. The translators have been written in TNR, the translation tool developed at CASSI.
The unidirectional translators for PART, CA, VNC and UG call the `main()` template in the file `main.tnr` [19] and pass to it four parameters in response to specific "services" requested by the user. The execution of the TNR program, in addition to answering the requests, may also result in requests for confirmation, requests for additional information from the user and status reports to the user. A GUI system should be able to communicate requests of users to appropriate TNR translators and handle requests arising from the execution of TNR programs asynchronously.

### 3.2.1 User Request Issues

The four parameters supplied to the `main()` template can be listed as below:

**Command:** A mandatory parameter, this is a delimited request whose components describe the nature of the request, the application that is requested to transact the request and the UDMM entity. The nature of a request can be any of the following types:

- **List/List Model:** Such as request gives a listing of keys available to the user.
- **Set Target:** A request of this nature permits the user to select one of the available libraries in either CA or VNC.
- **Clear/Clear Target:** This is used to clear all or portions of a local repository.
- **Export:** The export request is called by PART and by UG. An export request is issued when it is desired to move data from either PART or UG into the common store.
**Import:** Implemented for VNC and CA, this request is issued to convert information from the local store into the format of the particular application (either VNC or CA).

**Key:** Another delimited parameter, it contains a set of user-specified keys. It allows for a request to be simultaneously executed on a number of entities.

**Step File:** This optional parameter names the local store to be used for a particular request.

**Target Model:** Another optional parameter, the target model names the target-application model for the application.

The execution of a TNR command from a GUI necessitates the mapping of the four parameters into user selectable options in the interface. The GUI should provide the user with a choice of available UDMM entities and a listing of keys for the selected entity. The GUI should also keep track of the current local store (the third parameter of the *main()* template) and, if applicable, the target-model (the fourth parameter of the *main()* template).

The entities in the IMW UDMM are discussed in a later section in this chapter.

### 3.2.2 TNR Output Issues

The results of a command execution by TNR are given to the standard output. All messages to the standard output are prefixed by message “labels”, that indicate the nature of the output being displayed. It is left to the GUI to “intercept” the messages to the standard output, parse the message labels and perform appropriate actions based on the
nature of information. Information could be passed to the user through dialog boxes and the user response could be communicated to TNR by means of a pipe.

3.2.3 Interfacing Issues

As has been stated previously, the four tools that have been integrated into the IMW platform are CA, PART, UG, and VNC. The functionality provided by each tool is vastly different from that provided by other tools in the platform. Solid models of the part being manufactured are generated using UG. The information generated by UG is exported to the local store with the help of the UG translator. Similarly, the process plans generated in PART are also exported to the local store. The information in the current local store is retrieved either by CA or by VNC so as to generate effective cost models or to simulate the machining process.

It is essential that the GUI provide the user with a capability to generate information in the localized domain of a tool and then export this information to the local store. This is true, especially, in the case of PART and UG. It is necessary that the interface facilitates the import of information made available in the current local store into both - CA and VNC.

The development of the IMW is a continuous process. As more and more tools are integrated into the IMW, the GUI should provide enough modularity to interface the new tools with ease into the GUI. Provisions should also be made so that any future changes in the IMW could be reflected in its interface.
3.2.4 The IMW UDMM Entities

The IMW UDMM has been defined using EXPRESS, the schema definition language of STEP. The UDMM is the representation of all different types of data that could potentially exist in the IMW system. The entities in the UDMM have been categorized into the following three types [19]:

1. Independent Entities: These entities are a representation of the objects that exist in the real world. The independent entities are unique and form the “visible interface” to facilitate data flow between the different applications.

2. Dependent Entities: Multiple instances of these entities may exist in the same local-store containing identical entities.

3. Utility Entities: These entities are used all over the UDMM to hold certain specific data. These are really dependent entities which have been contained within multiple independent entities.

The export or import of data in the IMW is through the independent entities. While exporting data from a particular application, an independent entity is chosen and exported. All the sub-entities contained within the independent entity are also exported automatically. In addition, when an independent entity is to be retrieved into the current local-store, it is first verified to see if the requested entity has already been retrieved by an earlier request. If an instance matching the requested entity is found, it is returned.

The independent entities presently defined in the IMW UDMM schema is as follows:

**PartElement.** This represents a component of a part-assembly.

**Feature.** A feature is the representation of a geometric (possibly machined) feature.
**Operation.** An operation can be a machining or an assembly operation that is performed on a part or a part-assembly to produce a feature or sub-assembly.

**Setup.** A setup is a grouping of all operations that will performed in a single setting.

**WorkstationClass.** A workstation class is an instance of a workstation.

**MachineProgram.** This represents a NC program used on a certain workstation.

**OperatorClass.** The operator class is used to represent operator skill levels.

**TooledWorkstation.** The tooledworkstation is a representation of a workstation which has been completely tooled and made ready for some set of operations.
4 The IMW Interface

The various aspects in the development of effective user interfaces were discussed in the previous chapter. The issues specific to a GUI implementation to IMW were identified and enumerated and tools available for the development of the IMW GUI were presented. They form the basis for the actual implementation of the IMW GUI, described in some detail in various sections in this chapter. This chapter focuses on the design and operation of a graphical user interface for the Intelligent Machining Workstation (IMW).

4.1 The IMW GUI Implementation Language

The choice of a viable implementation language for any software project can often be considered a religious issue at best. In selecting the implementation language for the IMW interface, several criteria, such as hardware platform, cost, ease of availability were considered. The TNR layer of the IMW has been implemented on the UNIX platform. Hence, the choice of the implementation language for the GUI was restricted to the UNIX and the windowing system prevalent on UNIX, namely, the X window system.

The availability of several implementation languages under the UNIX platform made the choice of the GUI development tool somewhat difficult. The most obvious contender would be C in conjunction with the Xlib library. However, it was deemed that this library was too low-level for rapid application development. The advantages of a
scripting language over a systems programming language rendered unsuitable the choice of another popular tool, Java. Finally, the scripting language TCL with the TK toolkit was evaluated and chosen as the desired GUI implementation language.

One of the primary advantages of TCL/TK is that the source code is freely available. Consequently, there is no need to deal with the economic burden or the administrative overhead of paying for the package initially and paying again for subsequent upgrades and bug fixes. The existence of a clean, well-documented functional interface to the internal mechanisms of TCL makes it relatively easy to extend TCL to include features which are either too slow or not directly supported in TCL.

Programming a GUI can be a very arduous and demanding chore. TCL/TK helps make the task easier by raising the level of abstraction for the programmer, thereby making the implementation of user interfaces easier and quicker. TCL is relatively easy to learn and provides most of the features one would expect from a general purpose programming language. Since TCL is an interpreted scripting language, there is no need for the developer to compile the code. This makes rapid prototyping more feasible with TCL/TK.

TCL/TK was originally implemented for the X Window System; as a result, it runs seamlessly under a wide variety of UNIX platforms. However, the language has been ported to other popular operating systems, thereby enabling an application written using TCL/TK to be relatively portable across a variety of different architectures and operating systems. The potential user-base of such an application is, therefore, quite large.
There are a number of extensions to the TCL scripting language. One such extension is Visual Tcl. Much of the GUI development work was done in Visual Tcl for reasons outlined in the following sub-section.

4.1.1 Visual Tcl

The development of a user interface is made easy by the use of a user interface builder. This fact has been stated in a section in chapter 3. Application development environments or user interface builders for developing TCL/TK applications are currently available, to state a few – SpecTcl, Visual Tcl and SCO Tcl.

Visual Tcl is a popular and a widely used TCL/TK interface builder. Visual Tcl provides a high-quality application development environment for developing applications in TCL/TK. The Visual Tcl basically consists of a canvas and a toolbar providing a set of widgets and functionalities necessary to develop simple to complex applications. The fact that Visual Tcl is written in TCL and generates pure TCL codes makes porting among the different platforms trivial. Some of the notable features of Visual Tcl include extensible widget geometry manager support and creation of compound widgets and widget libraries.

Visual Tcl has several main windows. All the main windows can be opened and closed through a main menu. The application development support provided by Visual Tcl can be better visualized by the brief description of the functionality provided by the different windows in the Visual Tcl interface. The various windows in the Visual Tcl development environment are as follows:
Attribute Editor. Various attributes of an active window are either set or edited in the attribute editor. These include the setting or changing of font types and sizes, color schemes, active widget height and width, geometry parameters for the active widget and various other properties specific to a particular widget type.

Visual Tcl Console. This is the command console from where the other windows can be opened or closed. The command console also provides for the creation of compound widgets (combination of two or more widgets) and the testing of the application as it is being developed.

Function Editor. Existing functions can be edited and new functions created from the function editor.

Widget Toolbar. The widget toolbar is a palette of widget icons. The palette contains all of the widget types supported by TCL/TK. Appropriate widgets are chosen from the toolbar in the development of an interface.

Toplevel Windows. This windows lists all the top level windows that have created for the given application.

Variables Window. A list of all the global variables that have defined in the application are displayed in the variables window.

Widget Browser. The widget browser is a widget tree diagram of the application.

4.2 Overview of the IMW GUI

This section provides a high level description of the GUI as seen by the end user; it constitutes an abbreviated user's manual, which describes how the user interacts with the GUI to move product data among the different tools in the IMW. Some lower level
details are also presented to describe the implementation of some of the top-level interface elements.

The IMW GUI employs a single window – the "main" window of the interface. The main window (Figure 6, below) provides the functionality necessary to help the user move data in the IMW.

![Figure 6. The IMW GUI Main Window](image)

**4.2.1 The Main Window**

The main window is at the “heart” of the entire application. Using features provided by this window, the user could launch applications, translate data and edit configurations. In order to make the interface easy to use, the main window adopts a
presentation format similar to ones provided by numerous other GUI applications – it employs a pull-down menu bar and a work area arranged as shown above in Figure 6. The work area of the main window resembles the main interface of the WS_FTP package by Ipswitch, Inc. By using a GUI layout based on an already prevalent tool used in the transfer of files between two systems, it is assumed that the users would find the IMW interface relatively easy to use in the transfer of data between two applications.

The following subsections briefly describe the user interface elements comprising the main window display. In particular, an overview will be provided regarding their purpose, usage and implementation.

4.2.1.1 Pull-Down Menu

The pull-down menu user interface component is ubiquitous in software applications today. It is comprised of a row of menu buttons, called a menu bar, along the top of the display each of which is associated with a pull-down menu. The pull-down menu (or submenu) is displayed when the user clicks the leftmost mouse button on the menu button. The user can then drag the mouse pointer to the desired option in the pull-down menu and release the mouse button to activate the feature.

Pull-down menus offer users both familiarity and ease of use. As users become acquainted with the features offered by pull-down menus, accelerator keys may be used instead of the mouse to activate submenu options. The accelerator key for a given option is indicated to the right of the option label in the pull-down menu. In addition, the pull-down menu may be activated using the keyboard by pressing the ‘Alt’ key and the underlined letter of the menu button in the menu bar. The left and right arrow keys will
traverse adjacent submenus while the up and down arrows can be used to select options within the submenu. The current implementation of the IMW interface offers three pull-down menus: File, Utilities and Help. By convention, the Help menu button is displayed to the right of other menu buttons. Figure 7 gives the options for the File and the Utilities pull down menu.

![Diagram of File and Utilities Menus](image)

**Figure 7. File and Utilities Pull Down Menu Options**

The File pull-down menu consists of a single option – Exit. The Exit option terminates the application.

The Utilities pull-down menu provides for three options, all of which enable the user to obtain and modify details regarding the internal configuration of the IMW. The Display Monitor displays the status of the TNR commands as they are being executed. The Edit Configuration option provides for the viewing and for the modification of the variables defined in the “.imw_rc” file of IMW. The Unix Shell option, as the name suggests, launches a UNIX shell so that the user can perform necessary modifications to the TNR layer of the IMW without terminating the interface.
The Help pull-down menu presently contains only one option, “About…”, which displays a dialog box containing the name and version of the application. In the future, this menu should offer hypertext help to the user.

The options offered by the pull-down menu are sparse at present; however new options can be made available to the user relatively easily through the use of Visual Tcl. A discussion of the implementation details as well as how to add new options is presented in the following sections.

4.2.1.2 The Work Area

The most dominant part of the IMW GUI main window is the “work area” which is located under the pull-down menu. The work area is the region where the user interacts with the various applications in the IMW. The work area currently provides support for the selection of one of the four tools presently in the IMW, selection of necessary parameters for data transfer and launching of applications.

The work area is characterized into three distinct regions – the tool selection area, the parameters specifications area and the command area. The following is a brief description of the three sub-areas of the GUI work area.

Tool Selection. This essentially consists of a combo box providing a list of the four applications presently supported by the IMW environment. The user is required to a select tool from the available options in order to proceed with other tasks on the interface.

Parameter Specification. As stated in the previous chapter, the execution of any command in the IMW requires the supply of four parameters to the main()
template of the TNR layer of the IMW. The parameter specification area provides for the specification of the four parameters (Command, Key, Step File, Target Model) through user selectable options on the interface. The parameter specification area is characterized into two similar but distinct regions – the target model side and step file side. These two regions of the GUI work area are comparable to the “local system” and the “remote system” of the WS_FTP interface which facilitates the transfer of a specified file on the local system to the specified destination on the remote system and vice-versa. The target model and the step file sides allow for the specification of the target model or the library of the chosen tool and the step file and the necessary product data to be exchanged between the tool library and the step file.

Command Specification. This area consists of a set of four push buttons that individually provide the user with the ability to launch applications, translate product data and perform queries on a step file.

4.3 The GUI Implementation

This section provides insights into the internal workings of some of the aspects of the GUI described in the previous section. In general, details regarding procedures, internal data representations and strategies adopted by the implementation will be described.
4.3.1 Overview of the Implementation

As mentioned previously, the entire GUI was written in TCL/TK with the help of the Visual Tcl interface development environment. The high-level of abstraction offered by this language helped overcome many of the tedious, low-level implementation hurdles commonly encountered when developing a GUI. Also because there is no need to compile TCL scripts, development is reduced to relatively simple iterations of edit-execute cycles. Conventional compiled languages (most system programming languages), require much more time consuming edit-compile-link-execute cycles.

4.3.2 The GUI Modules

The GUI implementation of the IMW has been organized into several modules (TCL files). These modules implement the varied aspects of the GUI. The following are the IMW GUI modules:

Imw.tcl

This module is the main file of the GUI implementation. All TCL procedures used in the implementation of the different features of the GUI, have been defined in this module. When the IMW GUI is invoked, the contents of the imw.tcl file are sourced. After all the function definitions have been sourced by the TCL interpreter, the main window of the interface is launched by the execution of the main command.

DialogMain.tcl

This file contains the procedures responsible for constructing the main GUI interface. The layout of the main window was developed using Visual Tcl. Visual Tcl then generated the TCL scripts contained in this module.
**DialogFile.tcl**

The `DialogFile.tcl` module implements the “Tk”.file selection dialog box. The “Tk” standard file selection dialog box is similar to the file selection dialog box on the Windows operating system. The user can navigate the directories by double clicking on the folder icons or by selecting the “Directory” option menu. The user can select the files by clicking on the file icons or by entering a filename in the “Filename:” entry widget. The `DialogFile.tcl` module is actually a modification of the `tkfbox.tcl` module provided in the standard Tk library. The “save” file option in the `tkfbox.tcl` module has been modified to accommodate for the “create” option necessitated in the IMW GUI implementation.

**DialogDir.tcl**

The `DialogDir.tcl` module implements the directory selection dialog selection box in the IMW GUI. This module is a modification of the `DialogFile.tcl` module and the standard “Tk” `tkfbox.tcl` module. The module provides for the selection of directories (IMW libraries) as opposed to selection of files provided by the `DialogFile.tcl` module. The user can select the appropriate directory by clicking on the directory icons or by entering a directory name in the “Directory:” entry widget.

**DialogMessageBox.tcl**

This module implements the message box for the `tk_messageBox` TCL command. The message box is displayed with the application supplied message, an icon and a list of response buttons.
**DialogTnrConfirm.tcl**

This module provides for confirmations requested of the user at run time by the execution of certain TNR commands. A dialog box is displayed with a TNR supplied message, a question icon and a couple of “yes” and “no” buttons. The module expects a colon delimited string as input and returns a string with a value of either “yes” or “no”, depending on the button selected. The colon delimited input string is broken up to set values for title and display text of the dialog box.

**DialogTnrSelect.tcl**

The DialogTnrSelect.tcl provides the GUI interface for the `select()` template of the IMW TNR layer. This module pops up a display consisting of a scrollable list box and couple of push buttons. The `tnr_select` procedure of this module reads a colon delimited string from the stdin and splits the string into a list of elements for the listbox. The selected elements from the listbox are output to the stdout as a colon delimited string.

**DialogTnrMAsk.tcl**

The DialogTnrMAsk.tcl module implements a dynamic interface to the `ask()` template. The implemented top-level widget consists of a series of label-entry widget combinations; the number of the label-entry widget combinations are dynamically determined by the number of elements in the colon delimited string, given to the module as input. Each element of the colon delimited string can be further delimited by the “bang” symbol (!) to set the values for the texts of each label-entry widget combination. The output, on the selection of the “Ok” button, is also a colon delimited string but
consisting of contents of all the entry widgets combined together into a string and delimited by colons.

**DialogTnrStatus.tcl**

This module implements a run time display for the IMW. Message strings reporting the execution status of a TNR command are passed to stdout through the `status()`. The `DialogTnrStatus.tcl` then reads from the stdin and displays the input strings in a scrollable listbox.

**DialogTnrWarn.tcl**

This module implements a dialog box displaying a warning message generated during the execution of a TNR command. The `DialogTnrWarn.tcl` module provides the GUI interface to the `warn()` template.

**DialogTnrError.tcl**

This module is essentially similar to the `DialogTnrWarn.tcl` module. In comparison to the `DialogTnrWarn.tcl` module, this module implements the GUI interface for the error messages that may be generated during the execution of a TNR command.

### 4.3.3 The TCL Procedures

The following are the various major routines that provide the functionality to the IMW GUI.

**Source**

This function takes the name of a file as its only argument. This command first checks for the existence of the file in the IMW TCL/TK directory. A failure to locate the file in the specified path results in an error message to the user and an exit from the function. If
the specified file exists in the specified path, the function assigns a value to the global variable "toplevel" and then calls the "source" command of the TCL/TK interpreter. The source command takes the contents of the specified and passes it to the TCL interpreter as the text script.

Main

The "main" function is utilized as an initialization function. This function is invoked after the function declarations in the "imw.tcl" file have been "sourced" by the TCL interpreter. The "messagebox.tcl" file which overrides the default messagebox provided in the TCL library is sourced by the invocation of the "main" function. This function then launches the main menu of the IMW interface.

Tnr_exec

The "tnr_exec" method, as the name suggests, executes a TNR command. The procedure accepts five arguments. The four parameters to an IMW TNR command are passed in as arguments to the tnr_exec function. The fifth argument to the "tnr_exec" method can either be the name of a widget (only an entry or a listbox) or a null string. The output from the execution of a TNR command is directed to a file. The contents of the file are read and appropriate actions taken, according to string identifier tags at the beginning of a line.

selected_tool

When an appropriate tool is selected from the combo box of available tools in the work area of the main window, the selected_tool function is invoked. The function initializes certain parameters on the target model side of the main window and then calls the
initialize_widgets function with the flag for the chosen tool as parameter to the called function.

initialize_widgets

The only parameter to the initialize_widgets function is the name of the tool specified in the tool combo box. The function then enables or disables widgets on the main window of the IMW as per the requirements of the selected tool.

step_file

This procedure provides for three possible actions on a step file. The step_file function expects a single argument on its invocation. The argument is a flag for the possible actions on a step file – open, create and clear. When the flag is set to open, the procedure invokes the file selection dialog box. The user can navigate through the directory structure of the dialog box and select an appropriate file. If create is specified as the flag, the file creation dialog box is invoked. Clearing an existing step file requires the execution of the appropriate TNR command.

target_model

The implementation of this procedure is essentially similar to the step_file procedure described previously. This procedure differs from the step_file procedure in that it provides for the three possible actions (open, create, clear) on a target model (an IMW library).

tnr_udmm_exec

The tnr_udmm_exec procedure provides functionality to the radio buttons on the step file side as well as on the target model side of the main window. The step file side and the
target model sides of the interface contain a similar set of radio buttons. When a radio
button is invoked, the function needs to know about the side on which the radio button is
located. This is given to the function as its parameter. The function performs appropriate
actions depending upon the specification or non-specification of a step file or target
model.

**Tnr_translate**

This common function provides for transfer of data from step file to a target model and
vice-versa. This function is invoked on the invocation of the translate buttons on the
main window. The tnr_exec function is invoked by the tnr_translate function passing to
it parameters specified by selection of different options on the GUI by the user.

### 4.4 Data Translation using the GUI

This chapter has discussed the implementation of several aspects of the IMW GUI.
The GUI facilitates the flow of data between the various tools in the IMW. Through the
implementation of the various modules and procedures, the GUI helps the user to interact
graphically with the IMW. The movement of data in the IMW has been simplified to the
specification of a few parameters on a single window. The use of the IMW GUI in the
transfer of data within the IMW is the topic of the following chapter
5 A Demonstration Session

This chapter provides a demonstration session with the IMW GUI. An example part named “steped_hole” has been chosen for the purpose of the demonstration. The main reason for the choice of the selected part is its simplicity. In this demonstration, the process plan for the “steped_hole” generated using PART, is exported from the local file format of PART to a step file and then imported into VNC and CA to simulate the machining process and to arrive at a relatively accurate cost estimate for the product.

When the IMW GUI is invoked, much of the work area of the main window is in a disabled state. This can be seen in Figure 8.
The user is required to choose an appropriate tool from the “Tools” combo box. The “Tools” combo box lists a choice of four tools – CA, PART, Unigraphics and VNC. The widget elements on the work area of the main window are enabled or disabled according to the selected tool.

5.1 Exporting Data from PART

The two-dimensional drawing of the sample part, as viewed in PART, is displayed in Figure 9.

Figure 9. The Sample Part
To export the process plans generated by PART to a step file, the user first chooses the “Part” option in the “Tools” combo box. The Main Window for “Part” is as shown in Figure 10. It can be seen that the “Translate →” push button has been disabled along with a few radio buttons. The user now specifies the target model whose process plans he/she wishes to export to the step file. This is achieved by selecting an appropriate radio button on the target model side (left side on the Main Window work area). The “steped_hole” has been classified as a part element entity in the schema of UDMM\(^2\) entities. Hence, the “PartElement” radio button on the target model side is selected. On the selection of a radio button, a TNR command is executed to connect to PART’s ORACLE database and obtain a list of all “Part Elements”, which are then displayed in the list box on the target model side. The user also specifies the step file to which the data is to be exported. For the purpose of this demonstration, a step file named “sample.stp” has been used.

\(^2\) The UDMM entities have been discussed in some detail in Chapter 3.
Figure 10. The Main Window for PART

Figure 11 and Figure 12 represent stages in data translation. Figure 11 displays the selected Part Element and the specified step file. Figure 12 shows that the data translation was successful. The step file “sample” now contains a single entity – the “stepped_hole”. This data in the step file can now be imported into CA for cost modeling or into VNC for simulating the machining process.
Figure 11. The Main Window at the Time of Data Translation

Figure 12. The Main Window After Data Has Been Successfully Translated
5.2 Importing Data into CA

To import data into CA, the user selects “Cost Advantage” from the “Tools” combo box. As seen in Figure 13, the step file side of the Main Window work area remains unchanged. The step file side may or may not be reinitialized depending on the selected UDMM entity. The current implementation of the IMW, supports only one UDMM entity (Part Element) in the case of CA. However, the target model side is always reinitialized as the target models are different for different tools. The step file is independent of any particular tool.

![Figure 13. The Main Window for CA](image)

The user either selects an existing target model or creates a new target model. A new target model was created (Figure 14) for this demonstration.
The user now selects the "→ Translate" push button to generate reasonably accurate cost estimates from the information stored in the "sample" step file. Figure 15 shows one of the various dialog boxes displayed at run time in this GUI implementation. Here the user is asked to confirm if he/she wishes to include operations in cost note. The option chosen was "yes".
Figure 15 The Confirm Dialog Box

After the data translation, the “sample” CA library now contains information about “steped_hole” (Figure 16).

Figure 16. The Main Window After CA Data Translation
The information imported into the “sample” CA library can be viewed by clicking on the “Launch” button after selecting “steped_hole” in the target model list box. Figure 17 displays the cost estimates for “steped_hole” as generated by CA.

<table>
<thead>
<tr>
<th>Part Description</th>
<th>Blank Length</th>
<th>Blank Width</th>
<th>Blank Thickness</th>
<th>Estimated Blank Volume</th>
<th>Density</th>
<th>End Material Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLE_ROUND_STRAIGHT 1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>4.4</td>
<td>0.581</td>
<td>0.040</td>
</tr>
<tr>
<td>HOLE_ROUND_STRAIGHT 4</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>4.4</td>
<td>0.581</td>
<td>0.040</td>
</tr>
<tr>
<td>Total</td>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>4.4</td>
<td>0.581</td>
<td>0.040</td>
</tr>
</tbody>
</table>

**Figure 17. Cost Estimates For “steped_hole” as Seen In the CA Window**

### 5.3 Importing Data into VNC

Figure 18 shows the available options for VNC. Data translation in VNC is similar to the data translation in CA; the only variables being the UDMM entities supported by the two tools. An implementation of the “Select” dialog box is displayed in Figure 19. Figure 20 presents a specialized “Ask” dialog box implemented for VNC.
Figure 18. The Main Window for VNC

Figure 19 The Select Dialog Box
5.4 Exporting Data from Unigraphics

Similar to PART, process plans can be generated using UG and the data exported to a step file. Unlike PART, UG stores its data in a local file format. To export data to a step file, a Part Element entity in a UG target model and the step file have to be specified. Figure 21 shows that a UG Part Element “groove” was successfully exported to the “sample” step file. The step file now contains two elements – stepped_hole and groove. Figure 21 also shows the available Main Window options for UG.

---

3 In its current implementation, IMW supports only Part Element entities for UG
Figure 21. The GUI Main Window for UG

The data exported to the step file from UG could, now, be imported into either CA for cost modeling or into VNC for simulation of the machining process.
6 Conclusions And Recommendations

This thesis has discussed the key elements of a prototype concurrent engineering environment, enumerating in some detail the tools integrated into the environment and the integration architecture for the environment. In particular, this thesis has devoted itself to the design and implementation of an interface to the stated concurrent engineering tool (IMW) so as to integrate and manage product data between heterogeneous CAD/CAM tools in the environment. By employing a high-level user interface toolkit, a front-end GUI was implemented to facilitate data transfer within the IMW. The creation of the GUI has dramatically increased the accessibility of the underlying TNR layer of the IMW by enabling the user to issue request for services graphically instead of textually.

The clean separation of the GUI from the TNR layer provides for future extensibility and reuse of the two software modules. In addition, a new GUI can be added relatively seamlessly to the IMW environment provided that it conforms to a mutual protocol for sharing of information with the TNR module of the IMW. The clear distinction between the GUI and the TNR layer also makes it easier to modify the implementation of one of the modules without adversely affecting the other.

However, despite the potential benefits and applications of the GUI, there are still several improvements that can be made to enhance the functionality of the GUI. There are some internal issues which are to be resolved with respect to the existing GUI code
base. As the GUI implementation progressed, some unavoidable but repetitive code was written. Also, the required proliferation of global variables and arrays in the source code compromised the level of encapsulation between TCL modules. In the future, a possible rewrite of the GUI using a TCL/TK extension language with better namespace control and more effective code sharing support may be possible. Many TCL/TK extension languages provide an improved set of “ready-to-use” widgets that are additions to those provided by the TCL/TK core. Some of the TCL/TK extension languages have also improved on the functionality of the existing widget set of the TCL/TK core; an example being the addition of drag and drop feature to the list boxes in BLT\textsuperscript{4}. However, a possible disadvantage with using the extension packages is that they may not always keep pace or be compatible with the latest release of TCL/TK core from the Scriptics\textsuperscript{5} Corporation.

The implementation of the IMW is, at present, solely on an UNIX platform. Given the overwhelming popularity and usage of the Windows operating environment, it may be desirable, in future, to port the IMW application to PCs. This enhancement would serve to increase the commercial and industrial viability of the IMW.

\textsuperscript{4} BLT is an extension to the Tk toolkit, adding new widgets, geometry managers, and miscellaneous commands. It does not require any patching of the Tcl or Tk source files. BLT is freely available and can be obtained from \url{http://www.tcltk.com/blt/index.html}

\textsuperscript{5} \url{http://www.scriptics.com}
Bibliography


