

## **INFORMATION TO USERS**

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

**UMI<sup>®</sup>**



**University of Alberta**

**DSS USER INTERFACE DESIGN METHOD WITH APPLICATION TO SHOP FLOOR  
SCHEDULING**

by

**Xuqing Wu**



A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

**Department of Mechanical Engineering**

**Edmonton, Alberta  
Fall 1999**



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

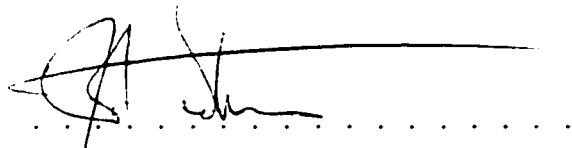
0-612-47118-7

**Canada**

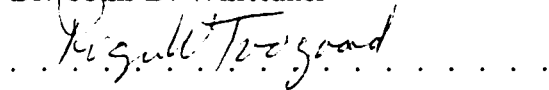
University of Alberta

Faculty of Graduate Studies and Research

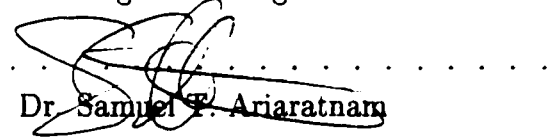
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **DSS User Interface Design Method with Application to Shop Floor Scheduling** submitted by Xuqing Wu in partial fulfillment of the requirements for the degree of **Master of Science**



Dr. John D. Whittaker



Dr. Roger W. Toogood



Dr. Samuel P. Ariaratnam

Date: 7. SEP. 99

## **Abstract**

The daily activity of establishing a job assignment schedule for shop floor in a manufacturing setting can be quite a complex management task. The Decision Support System (DSS), with its ability of helping decision-makers utilize data and models in a semi-structured or unstructured environment is an ideal candidate that can be easily implemented to solve such a scheduling problem. However, many of the previous research efforts have been spent on developing more complicated system structure or embedding more powerful algorithms into the system. Researches on developing systematic methods for designing the user interface of DSS for scheduling problems on shop floor level are few. In this thesis, a new procedure for this type of user interface design is developed. Aided with a powerful user interface, the scheduler may use the DSS in an exploratory manner to browse selectively through stored data or to analyze the implications of experiences related to the scheduling problem on the shop floor level.

## **Acknowledgement**

The author wishes to express his sincere thanks to his academic supervisor, Dr. John D. Whittaker, for his guidance and encouragement during the period of this research.

Special thanks are due to my parents for their unflinching support and understanding during the years of study.

## Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1 Introduction.....	1
1.1.1 Decision Support System and Scheduling Problem on Shop Floor Level..	1
1.1.2 User-Computer Interaction and User-Computer Interfaces Design.....	6
1.2 Thesis Definition.....	8
1.2.1 Research Motivation and Objectives.....	8
1.2.2 Research Methodology.....	11
1.2.3 Thesis Contribution.....	12
1.2.4 Organization of Thesis.....	14
<b>2. Brief Overview of the Applications of Interactive Scheduling Systems.....</b>	<b>15</b>
<b>3. A DSS User Interface Design Method to Provide Consistency and Adaptability</b>	
.....	<b>23</b>
3.1 Defining User Friendliness.....	23
3.1.1 Human Factors Approach.....	23
3.1.2 Task Dependency.....	27
3.1.3 Consistency and Adaptability of User Interface.....	30
3.2 DSS User Interface Model.....	34
3.2.1 Characteristics of DSS-User Interaction.....	34
3.2.2 The Framework for DSS User Interface Design.....	38
3.3 Proposed Implementation Heuristic.....	44
3.3.1 Related Research.....	44
3.3.2 Cognitive Engineering and its Application in User Interface Design.....	48



3.3.3 Task Analysis Method.....	52
3.3.4 User-Centered Implementation Heuristic for User Interface Design.....	54
3.3.5 Conclusion.....	57
<b>4. Case Study: A Prototype User Interface for Schedule Modification on Shop Floor</b> .....	<b>59</b>
4.1 Introduction.....	59
4.2 Framework of the User Interface.....	60
4.3 Implementation of the Framework.....	62
4.4 Software Implementation.....	70
4.5 Conclusion.....	78
<b>5. Conclusion.....</b>	<b>80</b>
5.1 Summary of the Research.....	80
5.2 Recommendation for the Future Research.....	81
<b>Bibliography.....</b>	<b>83</b>
<b>Appendix: Cognitive Findings and its Application Rules.....</b>	<b>90</b>

## **List of Tables**

Table 4-1: Data Structure of Scheduling Result Table.....	60
Table 4-2: Data Structure of Modification Information.....	61
Table 4-3: Data Structure of the Scheduling Knowledge.....	71

## List of Figures

Fig 1-1 The Basic Framework of the Decision Support System for Scheduling.....	5
Fig 2-1 CUISE System Overview.....	19
Fig 3-1 Ingredients of User-Friendliness.....	28
Fig 3-2 Overview of Decision-Making.....	29
Fig 3-3 A Modified Generic DSS.....	35
Fig 3-4 A Model of User/Support System Interaction.....	37
Fig 3-5 Prototype User Interface Development.....	40
Fig 3-6 A Framework for DSS User Interface.....	41
Fig 3-7 Proposed DSS User Interface Model.....	43
Fig 3-8 Generic Description of User Interface Model.....	43
Fig 3-9 Traditional Cognitive Model of HCI.....	49
Fig 3-10 A New Cognitive Model for HCI.....	50
Fig 3-11 Logical Flow of the Heuristics.....	57
Fig 4-1 Framework of the User Interface.....	62
Fig 4-2 TDH_1.....	63
Fig 4-3 TDH_2.....	65
Fig 4-4 TDH_3.....	67
Fig 4-5 TDH_4.....	69
Fig 4-6 Beginning of Schedule Master.....	71
Fig 4-7 Input Box of Schedule Master.....	72
Fig 4-8 File Open Screen.....	73
Fig 4-9 Main Window of Schedule Master (1).....	73
Fig 4-10 Main Window of Schedule Master (2).....	74
Fig 4-11 Main Window of Schedule Master (3).....	75
Fig 4-12 Main Window of Schedule Master (4).....	76
Fig 4-13 Text-Based Schedule Editor.....	77
Fig 4-14 Color Configuration Page.....	77

## List of Abbreviations

A/C:	<i>Adaptive/Consistent</i>	UCIs:	<i>User-Computer Interfaces</i>
AI:	<i>Artificial Intelligence</i>	UI:	<i>User Interface</i>
API:	<i>Application Programming Interface</i>		
CH:	<i>Conceptual Hypothesis</i>		
DAO:	<i>Data Access Object</i>		
DSS:	<i>Decision Support System</i>		
ERP:	<i>Enterprise Resource Planning</i>		
HCI:	<i>Human-Computer Interaction</i>		
JOH:	<i>Job Oriented Heuristic</i>		
KRG:	<i>Knowledge Representation Grammar</i>		
KS:	<i>Knowledge System</i>		
LS:	<i>Language System</i>		
MIS:	<i>Management Information System</i>		
MRP:	<i>Material Requirement Planning</i>		
OH:	<i>Operational Hypothesis</i>		
PPS:	<i>Problem Processing System</i>		
TAG:	<i>Task Action Grammar</i>		
TDH:	<i>Task Descriptive Hierarchy</i>		
UAN:	<i>User Action Notation</i>		
UCIH:	<i>User-Centered Implementation Heuristic</i>		

# **1. Introduction**

## **1.1 Introduction**

Usability of interactive computer systems is at the very core of the computer, communications, and information revolution. The challenge of this work is to create usable interactive systems that provide effective communication with their users, so that the power of computers can be available to the user for meaningful and enjoyable use. Creating an easy-to-use or user friendly interface is even more important for scheduling systems since participation of human power is inevitable under modern complex manufacturing environments.

This thesis discussed several issues related to establishing a flexible user interface for a Decision Support System (DSS) for the scheduling problem. The situation attention is placed on the shop floor level where feasible, quick response and easy-to-use scheduling tools are needed. A prototype user interface of DSS that provides modification function for a shop floor level scheduling problems is established based on a proposed heuristic for the user interface design.

### **1.1.1 Decision Support System and Scheduling Problem on Shop Floor Level**

Scheduling is one of the most important issues in the planning and operation of manufacturing/fabrication systems. It arises in situations where limited resources are to be assigned over time to perform a set of activities. Many productivity problems are

associated with scheduling problems such as not having the right item when needed, not having the right equipment when needed, using excess inventory to hide problems, and inflexibility and lack of responsiveness (French, 1982). Resolution of these difficulties can translate into considerable direct and indirect savings. Applications for scheduling can be found in production planning, time tabling, or real-time system control.

Introductions to scheduling can be found in Conway et al. (1967), Baker (1974), French (1982), and Pinedo (1995).

The scheduling problem is in essence very simple to state:

*A set of  $n$  jobs have to pass through  $m$  processes in such a way that some objective function(s) is (are) optimized.*

The simplicity of this statement, however, belies the extraordinary complexity of the problem in the practical situation. Three difficulties immediately emerge (Baker, 1974):

- The combinational nature of the problem.
- The difficulty of specifying the objective function.
- The need to accommodate change.

A complex manufacturing organization may have thousands of jobs and processors waiting to be scheduled. Scheduling problems on shop floor level can be viewed as a general job shop model of this kind of problem. In a general job shop, each job set is usually unique in work content and process sequence requirement. The number of alternative schedules can be incredibly large. Such a scheduling procedure usually

requires a scheduling mechanism with a response time sufficiently short to provide reschedule information as rapidly as required by the dynamics of the situation. Existing constructive algorithmic techniques for factory scheduling are capable of incorporating only a small fraction of scheduling knowledge and result in schedules that bear little resemblance to actual schedules (Reisman et al., 1997). For these reasons, more and more efforts are put into finding approximation algorithms for job shop problems. Heuristics provide the only viable approach to job shop scheduling problems of realistic size (Hastings, 1990). Heuristic algorithms can obtain solutions to large problems with limited computational effort, and their computational requirements are predictable for problems of a given size. The drawback of the heuristic approach is, of course, that it does not guarantee optimality. However, efficiency and flexibility are much more important than other issues in shop floor management. Additional information of scheduling heuristics can be found in White (1986) and Yeh (1997).

It is obvious that analyzing a scheduling problem and developing a procedure for dealing with the problem on a regular basis are only part of the story (Pinedo, 1995). Since most of the general scheduling problems in the real world are NP-hard problems and can not be solved by constructive algorithms, the procedure has to be embedded in a system that enables the scheduler to actually apply it. The Decision Support System (DSS), with its ability of helping decision-makers utilize data and models in a semi-structured or unstructured environment is an ideal candidate that can be easily implemented to solve such a scheduling problem (Schniederjans and Carpenter, 1996). The emergence of the notion of DSS can be traced back to the early 1970s. Morton

(1971) defined such systems as “interactive computer based system, which help decision makers utilize data and models to solve unstructured problem”.

According to Holsapple and Whinston (1976), the purpose of a Decision Support System can be stated as:

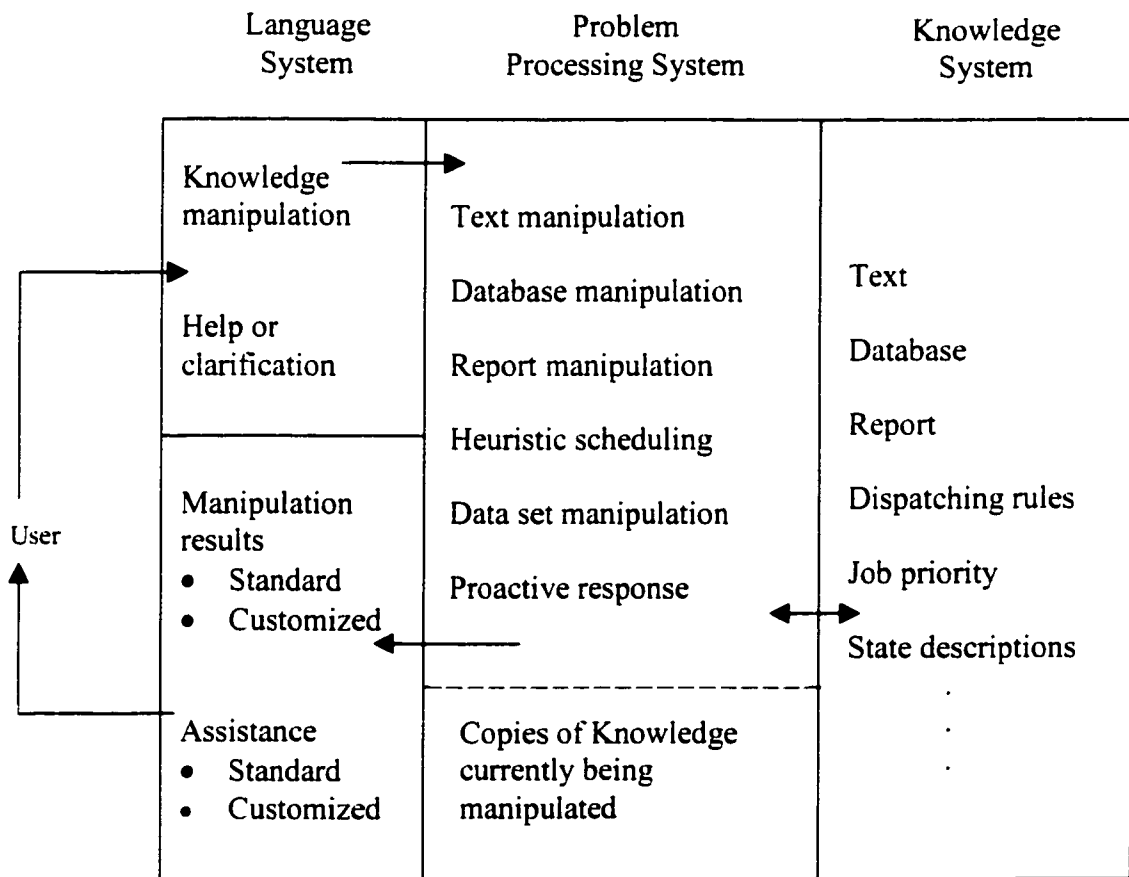
- Supplementing the decision-maker.
- Allowing better intelligence design or choice.
- Facilitating problem solving.
- Providing aid for unstructured decisions
- Managing knowledge.

Alter (1980) and Bonczek et al. (1981) identified five characteristics as the important DSS trait:

- A DSS includes a body of knowledge that describes how to accomplish various tasks and indicates what conclusions are valid in various circumstances.
- A DSS has the ability to acquire and maintain knowledge.
- A DSS can present knowledge in various customized ways.
- A DSS is able to select any subset of stored knowledge for either presentation or deriving new knowledge in the course of problem recognition and/or problem solving.
- A DSS can interact directly with a decision-maker in such a way that the user has flexible choices.



In the shop floor level planning, the objective of DSS for scheduling is not only to obtain a short term schedule of the operations, but also to make the system interactively response to any dynamic changes made by the scheduler. A typical architecture of knowledge-based DSS for handling such a problem is shown in Fig1-1 (Dos Santos and Holsapple, 1989).



**Fig 1-1: The Basic Framework of the Decision Support System for Scheduling**

Knowledge-based DSS could be defined in terms of four essential aspects, a language system (LS), a presentation system (PS), a knowledge system (KS) and a problem-processing system (PPS). The LS consists of all messages the DSS can accept

and the PS consists of all messages the DSS can emit. LS and PS are the two basic components of the user interface. In any case, a scheduler may make a request to accept knowledge or to clarify previous requests or response by selecting a desired element of LS. After the system successfully finishes processing the request, the response is fed back through the PS. Knowledge system consists of all knowledge the DSS has stored and retained. Knowledge system is the data source for further reasoning process. The KS, LS and PS are essential elements of a DSS. Each is used by the fourth element: the problem processing system (PPS). This is the active part of a DSS. PPS identifies three indispensable DSS abilities: knowledge acquisition, knowledge presentation and knowledge selection/derivation. Detailed instruction to Decision Support System and its application can be seen in Bennett (1983), and Holsapple and Whinston (1996).

### **1.1.2 User-Computer Interaction and User-Computer Interfaces Design**

User-Computer Interfaces (UCIs) are often essential parts of an information or Decision Support System. Andriole and Adelman (1995) thought “In fact, UCIs are often critical components of software-intensive systems of all kinds”. Since the early 1980s, technological developments have led to the appearance of a variety of interface peripherals (color, pointing devices, optical readers, sounds, etc.) which bring with them new opportunities for solving the ergonomic problems encountered by the user. In this context, taking the user into account in designing an interactive computer tool is no longer limited to the development of interaction tools associated with exclusively technical characteristics of computer hardware. It also includes the skills and preferences

of the users and different classes of users. New approaches to the design of user-friendliness of interfaces emerge with the development of the ergonomics and cognitive science. From a knowledge-based DSS perspective, an interface is a shorthand way of referring to the language system, the presentation system, the portion of the problem processing system software that accepts the former and produces the later, plus any linguistic or presentation knowledge in the knowledge system (Holsapple and Whinston, 1996). To a DSS user, the user interface is the system (Bennett, 1983). It is a user's point of contact with the DSS. Someone says that a user interface is the most important aspect of a DSS (Sprague and Carlson, 1982). Empirical studies have shown that the nature of the user interface can influence the impact of a DSS on decision making (Webster, 1990).

Thus, it is vital for the developer to understand why the user interface is so important, what kind of user interface should be developed and how to create a suitable user interface. To answer these questions, researchers first have to find a goal for user interface design. It is often remarked that a good user interface is a user-friendly interface or is easy-to-use. But what do friendly and easy really mean? This question and its answer are often ignored (Lapointe, 1991). However, the definition does begin to be sensible when the designer tries to identify who the user is and in what tasks that user is engaged when interacting via the interface (Holsapple and Whinston, 1996). This issue will be discussed in detail in the following chapters.

Based on the definition of user-friendliness and easy-to-use user interface, scholars try to map the psychological term into system analysis domain. Several user interface

models are proposed. Some of them will be shown in the following chapters. Another challenging issue related to user interface design is how to implement the conceptual user interface model and embed it into the system. Cognitive engineering and behavioral science, human studies, empirical studies, semantic analysis, task analysis and system analysis supply different approaches for developing implementation procedures to demonstrate the feasibility and technical validity of the model. However, none of them is suitable for creating a consistent and adapted user interface of DSS for shop floor level scheduling problems. In this thesis, a procedure for this kind of user interface design is developed with the demonstration of its applications.

## **1.2 Thesis Definition**

### **1.2.1 Research Motivation and Objectives**

Scheduling decision is of particular importance to both researchers and practicing managers. Research in this area obviously has not been carried out to its completeness since it is apparent that so many areas still remain untouched. The practicing managers of the shop floor are increasingly faced with difficult situations in which jobs have to be delivered on time; otherwise, costs will be incurred. Making an efficient schedule becomes a necessity in today's extremely competitive business world. It is the basis of leading to the concept of zero inventory (ZI) which insists on carrying no inventory at all. Furthermore, manufacturers who want to stay ahead of the competition by even more responsive to their customer's need, especially when customers' lead time are shorter than manufacturer's cycle time, have expected to have a sophisticated planning

capabilities for a long time. To be able to meet those kinds of demands, they need a planning system that optimizes the production process and allows manufacturers to get more capacities out of by being more efficient. These scheduling and control requirements impose a scheduling mechanism with a response time sufficiently short to provide reschedule information as rapidly as required by dynamics of the situation in the shop floor. The characteristics of DSS make it an ideal computer system to solve such scheduling problems on shop floor level. Many of the previous research efforts in this area have been spent on developing more complicated system structure or embedding more powerful algorithms and more effective heuristics into the system. Some viable examples of such systems can be seen from the papers published by Viviers (1983), Collinot et al. (1988), Chung et al. (1993), Wong and Monaco (1995), Schniederjans and Carpenter (1996), Kim and Lee (1997), Nkasu and Leung (1997), Gopalakrishnan (1998). The development of better system structure, algorithms and heuristics is essential if scheduling theory is to continue to improve scheduling practice. However, there are other research directions requiring more attention, which may have as great impact on scheduling practice (Graves, 1981). There is a great need, not only for better scheduling system structure and algorithms, but also for more realistic systems with the scheduling setting and for increased understanding of the dynamics inherent in the scheduling environment. Since much of the power, flexibility and usability of the DSS are derived from the capabilities of the user interface (Sprague and Watson, 1986), more and more attentions have been placed on developing an efficient method for designing an effective user interface to close the gap between the decision-maker's requests and the functions of the internal components of the DSS. However, the design of the user interface of DSS for

scheduling problems on shop floor will be quite complex since the schedulers have to manipulate the database, make feasible schedule, show the scheduling results and cope with the unexpected situation in one system. Although the method for designing the user interface of scheduling system is very important, to date, little systematic work has been applied to this area.

The primary objective of this thesis is to find a way to construct a flexible user interface of DSS for scheduling problems to close the gap between user interface design theory and its application. At the same time, the user interface can help to eliminate the differences between user's requests and commands of the internal component of the system by separating the user interface model from the scheduling algorithms used in the background. This effort leads to a 'precisioning procedure'<sup>\*</sup> that is based on cognitive method and task analysis to implement the model suggested by Sankar et al. (1995). A prototype user interface for schedule modification purpose is then established with the ability to:

- Be adapted to the requests of different users.
- Be consistent to internal components.
- Be independent of the algorithms used in the background.
- Minimize possible interruptions by encapsulating the user interface model.
- Expose changeable properties to the user

---

<sup>\*</sup> Precisioning procedure is a term adopted by Sankar et al. (1995) to describe a detailed and easy-to-follow process.

### **1.2.2 Research Methodology**

Sankar et al. (1995) suggested a DSS user interface model that provides consistency and adaptability. The proposed model allows an end-user to converse with the DSS using an adaptable interface and, in parallel, allows effective communication with the internal PPS and KS using consistent internal commands. This is achieved by separating each user interface component (LS and PS) into adaptive and consistent sections. The adaptive section adapts to the unique requirement of a user and the consistent section provides consistent commands to the internals of the DSS. The adaptive terminology module and the consistent terminology module are connected by an Adaptive/Consistent (A/C) interface module. The appeal of this model lies in the complete separation of the unique interface of a user from the PPS and KS.

However, the procedure that can be used to implement the proposed model is based on semantic analysis, which requires users to interact with the system, identify synonymous terms under each heading, and replace them with consistent terms. This procedure makes the module difficult to be applied to the scheduling problems on a shop floor because it is impossible to split such complex scheduler's requests into syntax headings. Cognitive-based design method, which is derived from cognitive engineering research, suggests that user interface design is extremely sensitive to both the task characteristics and user characteristics. Instead of using linguistic analysis to solve adaptive problems in user interface design, cognitive engineering concentrates on human-oriented design in contrast to technology-driven design. Powals et al. (1996)

demonstrated how cognitively engineered user interface could enhance user-computer performance in their case study. This is accomplished by designing an interface using empirical findings from the cognitive literature. Research of Stary and Peschl (1998) showed that human-computer interaction was not driven by the execution of actions to achieve goals. They thought human-computer interaction was multi-level interaction process between two representation systems. The understanding of the task is as important as the representation of the user knowledge. Task descriptive hierarchy (TDH) tree, which analyses the data from the observation of relevant tasks and redescribes them all within a single consistent representational form, is an idea tool for task analysis purpose (Diaper, 1989). This paper describes a precisioning procedure that is based on cognitive method and task analysis to implement the user interface model proposed by Sankar et al. (1995) into DSS for shop floor scheduling problems.

### **1.2.3 Thesis Contribution**

This thesis is a continuation of work that was done previously by Gopalakrishnan (1998). The research leads to a new approach, user-centered implementation heuristic, for designing an adapted and consistent user interface to reduce the cognitive effort of the end user during the interactive scheduling process. This new user interface design method can be applied to not only scheduling system but also other interactive computer systems. Based on the review of the concept of user friendliness, the author finds that user friendliness, when translated from philosophy domain into system analysis domain, can be realized by a user interface that provides both consistency and adaptability. This



finding results in a new approach to design an interface adaptable to the various styles and needs of different users yet consistent to internal DSS components. Compared to other existing user interface design methods, the new heuristic concentrates on both user characteristic analysis and system usability analysis. Since the user interface is the channel where the user communicates with the system, it is very important to involve the user's cognitive preference into the design procedure. The new heuristic simplifies the procedure for user cognitive preference analysis by including the findings derived from the research of cognitive engineering and the rule-based cognitive findings are very easy to be implemented compared to other semantic/syntactic analysis techniques (The deficiency of the semantic/syntactic analysis method is discussed in chapter 3.3.1).

Based on the proposed user interface design heuristic, a prototyped user interface for general job shop scheduling problems on shop floor level is developed. The job shop scheduling problem is important in production planning. It represents a common manufacturing environment that includes both static and dynamic scheduling problems. However, it is too complex to be optimally solved. It is not until the application of NP-Completeness theory to the sequencing problem reveals that the real reason for a lack of success in theory is due to the intractable properties of the general job shop problems. It seems that the participation of the user during the scheduling procedure is inevitable and there exists the need for developing powerful and easy-to-use scheduling systems in an interactive way. This research attempts to close the gap between the user interface design theory and its application in scheduling systems. As the result, the application of the DSS for scheduling problem may reduce inventories, and cycle times, improve product quality

and customer satisfaction – all while reducing overall costs. It is a technology that leverages the planner’s knowledge. It effectively extends the decision-maker’s capacity for representing and processing knowledge in the course of manufacturing decisions. Aided with a powerful user interface, the decision-maker may use DSS in an exploratory way to browse selectively through stored data or to analyze the implications of experiences related to the scheduling problems.

#### **1.2.4 Thesis Organization**

Chapter 2 briefly reviews the application of interactive scheduling system. In Chapter 3, the precisioning procedure for developing the user interface is presented. Chapter 4 demonstrates the prototype user interface for scheduling purpose. Chapter 5 indicates the lessons learned and lists proposals for future research.

## **2. Brief Overview of the Applications of Interactive Scheduling Systems**

Over the past two decades, with the advent of personal computers in the factory, numerous DSSs for scheduling problems have been developed, and many more are under development. This section provides a brief review of those systems and highlights the design of interactive user interfaces in the DSS for scheduling purpose.

The research in this area can be traced back to 1960s. At that time, it was hypothesized that a symbiotic relationship between humans and electronic computers would be more powerful at certain types of problem-solving activities than either would be alone (Edwards et al., 1974). Although interactive systems have become widely used in several design areas after then, little had been done in the area of scheduling. Ferguson and Jones (1969) developed an interactive job shop scheduling system in the mid sixties. It was a relatively small shop consisting of six machines, and all interactions took place on a tele-type or other similar hard-copy terminal. By selecting the appropriate values of a number of operating parameters, the scheduler had the option of making all decisions or of having the computer make most of them. Performance could be measured by sales price of all jobs shipped. This system used text-based terminals as its display device. The process of the decision making was driven by text-based prompts one by one. Sometimes it provided several options that schedulers could select according to their experiences. The users could not change the text-based prompts for their own purposes and neither could they jump to the procedure that the schedulers prefer to deploy. This system was more like a data processing system than a DSS because it just computes with records of

what is known as a result of various processes waiting to be scheduled. However, its text-based prompts gave the users more choices from which they could select according to the specific situation. The interaction that exists in this system is unidirectional since the user can not send request to the system. This kind of one way interaction impedes the system to receive new knowledge from its environment.

In 1980s, research on DSS increased (Sprague and Watson, 1986). Based on the classic view of decision-making activity, the DSS for scheduling consists of four basic modules. The fundamental components of the DSS include (Pinedo, 1995):

- Database
- Schedule generation and regeneration modules
- User interface modules
- Application Programming Interface (APIs) with the ability to communicate with other information systems, such as Material Requirements Planning (MRP) system

The APIs module is in charge of the communications with other data sources. Without the APIs module, it is hard to input data into the database and impossible to receive up-to-date information concerning availability of machines, progress of jobs or other emergent situations in the shop floor. On the other hand, most commands and requests of the end user are transferred to the system through the functions supplied by APIs. So APIs module is a bi-channel engine that combines the external and internal components of the DSS together.

Viviers (1983) described an interactive DSS based on the traditional structure for a labor-constrained workshop at the University of Stellenbosch. In the mechanical workshop, typical problems such as unpredictable delivery time and numerous rush jobs make it difficult to do the job scheduling on a computer. In the meantime, many repetitive calculations, which have to be done, are ideally suited to programming. It was therefore decided to combine the abilities of man and computer by designing an interactive job shop scheduling system. When input into the DSS, a job is first scheduled and then a due-date based on the schedule lead-time of the job is calculated. This special workshop scheduling program (SMDS) package consists of six PASCAL programs with a common database. One of them enables schedulers to view the text-based report or graphical Gantt Chart on the screen. The SMDS is a text-driven interactive DSS. It is obvious that the designer of this system noticed the practical request of the scheduler and tried to interact with the scheduler to solve dynamic problems. In the same time, a graphical Gantt Chart provides a visual display to the end user. The problem of the user interface of this system lies in that all of the schedule selections are predefined and fixed because the APIs module can just supply limited functions to the users. No changes are allowed from its original configuration. The interactive ability of the system is rigid and limited.

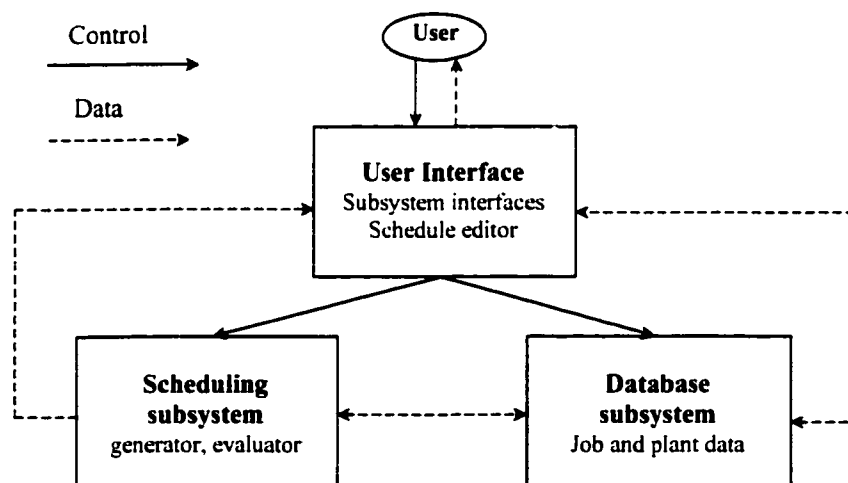
In addition to the classic DSS structure, a knowledge-based DSS is developed to cope with the dynamic scheduling problems (Holsapple and Whinston, 1996). Knowledge-based DSS could be defined in terms of four essential aspects, a language system (LS), a presentation system (PS), a knowledge system (KS) and a problem-

processing system (PPS). The LS consists of all messages the DSS can accept and the PS consists of all messages the DSS can emit. LS and PS are the two basic components of the user interface.

A demonstration version of Empress-1, a knowledge-based system that was developed for the purpose of helping Kennedy Space Center (KSC) prepare payloads, was produced in 1985 (Mulvehill, 1988). The significant improvement of this system is its design of the user interface. Since data about a payload are typically incomplete or totally unavailable at the time the planner would like to have them, planners often had to hypothesize likely processing requirement. The user interface of Empress-1 can help the planners modify the schedules by a user-specified manner. Developers of Empress-1 noticed that the definition of an interface was not limited to the system's display. The interface includes the representation of the data contained in the system, the way it is presented to the user and the mechanisms by which the user can manipulate it. Empress-1 supports a menu-driven graphical interface. KSC planner monitor payload processing activities through the master multiframe schedule Gantt Chart (MMF). The Gantt Chart permits experts to manually create schedules by pasting bars and labels on a board with a calendar as a background. All the modifications made by experts are subject to the constraints that represent task relations, predecessor-successor and temporal end points. Alerts are given when conflicts are identified after the user's modifications. So MMF can be used to generate plausible schedules. The Empress-1 interface is composed of five independent windows, standard, histogram, utilization and two system developer windows, inspect and debug. The user windows are to facilitate scheduling and resource

allocation. Each window is composed of a working area, in which the prime activity is displayed, a hierarchy of menus (menu containing submenus) and a dialog-based window where messages are displayed to the user and limited user interaction is performed. The user interface of Empress-1 makes it similar to the user's traditional tools. But users still have difficulties when using the menus and responding to prompts for values, because a user without training may not understand the command syntax of the LS/PS and often the command languages are difficult to learn and use.

The Columbia University Interactive Scheduling Editor (CUISE) 2.0 system is a scheduling system developed in the early 1990s (Pinedo, 1995). The goal was to design and develop an interactive flexible scheduling system that can be easily adapted to many different manufacturing environments. The system was designed to be easy to understand at the program level and retain enough flexibility to facilitate the integration of new features and algorithms. CUISE 2.0 is built around two major subsystems: a database subsystem and a scheduling subsystem. The two are unified by a common user interface (see Fig 2-1).



**Fig 2-1 CUISE System Overview**

The user interface of the schedule editor uses both text and graphics. These interfaces allow viewing, filtering, and editing of schedules as well as access to performance measures. Filtering is a means whereby the user may opt for emphasizing certain jobs based on a characteristic. For example, all jobs that are finished on time may be dimmed, and jobs that are late are emphasized in a brighter color. A text-based dispatch list interface is used to display how jobs are sequenced on each machine at each stage. Schedule editing is performed by selecting the job name on a machine and moving it with the cursor to a new position. This type of editing is more suited to event-driven scheduling because it is more cumbersome to specify an exact starting time. The graphic editor takes the form of a Gantt chart. Editing is accomplished by using a mouse to click and drag jobs to new positions. Depending on the options chosen, the move may be either event or time oriented. In CUISE 2.0 system, user interface is treated as an independent model with the ability to communicate with internal components. The user is not only a “listener” any more. The user’s requests will trigger the scheduling process and those requests can affect the performance of every scheduling step that is run in the schedule generator. However, the user’s knowledge is not included in the system. So the system can not be adapted to the various preferences of different users.

Park et al. (1998) presented an object class extraction methodology of production planning and control system as a part of enterprise resource planning (ERP) environment. They defined the object classes within the system and the detail attributes and operations about the object classes. They also established the relationships among these classes, hierarchical structure of the classes and the event trace diagram. Depending on the



characteristics of the function, the system is divided into object class group associated with production plan function, object class group associated with production plan information, and object class group associated with user interface on a large scale. The function of the user interface object includes processing progress indicator object and generating event object by user input data. The user interface classes consist of OrderManagement, LongTermPlanning, MidTermPlanning, ShortTermScheduling, RealTimeScheduling, PurchasingControl, InventoryControl, PcgInputOutputControl, and MfgInputOutputControl. Each of the components has its own internal interface functions with the exposed properties to the end users. In this system, since the user interface is encapsulated, the desired services and data associated with an object are only made available to the client through a predefined interface without revealing how those services are implemented. This means the user interface model can be transplanted to another system with less effort as long as the other system accepts the same scheduling mechanism.

The aforementioned cases show that much of the research on user interface for scheduling problems focussed on data, scheduling procedures, rule sets, text, forms and spreadsheets associated with the problem/decision area. Some other interactive DSSs for scheduling problems are reported by Collinot et al. (1988), Chung et al. (1992), Wang and Monaco (1995), Schniederjans and Carpenter (1996), Lipske (1996) and Kim et al. (1997). The review also shows that most of the systems are developed without explicitly demonstrating the user interface model. Some systems, such as Empress-1 (Mulvehill, 1988) and CUISE 2.0 (Pinedo, 1995), ignored the importance of the user characteristics

and their effects imposed to the system although they have independent user interface model. Lustman et al. (1985) pointed out that the lack of the research on user interface resulted in the system difficult to learn and use. The research conducted by Sankar et al. (1995) shows that the cost of a DSS user interface can run as high as 60 to 70 percent of the total cost of building a DSS. It is suggested that the deficient user interface has impeded the large scale of implementation of DSS for scheduling on shop floor level.

### **3. A DSS User Interface Design Method to Provide Consistency and Adaptability**

#### **3.1 Defining User Friendliness**

Until the late 60's and early 70's, the main effort of software application designers, was aimed at developing applications with higher functional characteristics. The main applications of computers were in scientific research and high volume commercial data processing. Usually the people using the computer applications were highly computer literate and properly trained (Cudd and Oskouie, 1996). Emergence of personal computers changed this situation quite rapidly. On one hand the availability of more computing power helped to reach the functional objectives much easier, and on the other hand the computer user community became more heterogeneous (Paffaf, 1985). These facts diverted the application designers focus from functional excellence to interaction flexibility and ultimately "user friendliness". But what does user friendliness really mean? This question and its answer are often ignored. This material is not intended to be an in-depth presentation; neither is it the thesis's goal to deal with psychological underpinnings of the guidelines. The overview of this concept is included because understanding of user friendliness is really helpful for defining the objective for user interface design.

##### **3.1.1 Human Factors Approach**

Santos and Bariff (1988) said "Like beauty, user friendliness is in the eye of the beholder". What one person regards as an excellent interface may be seen as too

primitive, inflexible, or simplistic by someone else. On the other hand, the latter's preferred interface may be viewed as overly complex, too hard to learn, or even mind-boggling by the first person. Such differences are partially due to variations in communication styles preferred by people. But they may also reflect different levels of experience in computer usage. What a novice regards as straightforward can seem too limiting to an experienced user. What an experienced user regards as straightforward can seem arduous or baffling to a novice.

Besides the psychological considerations, Norman (1983) presents a series of principles underlying the approach to the user-computer interface, from the perspective of the study of the human factors associated with operating computer tools, presented as an individual field of scientific practice. Norman (1983) took into consideration the characteristics (strengths and weaknesses) of the technology itself in the study of phenomena he describes as parallel or independent (work environment, etc.). This involves establishing at the design stage (and not after the event) a close relationship with the various local constraints (technical, organizational, budget and time-related, etc.) surrounding siting of the tool and, above all, introducing appropriate design principles. The idea is to build an interface whose parameters can be easily modified, independently of the tool, thus offering the possibility of creating personalized interfaces. The goal is to allow greater flexibility, both to the designer and to the interface itself. In this perspective, Stevens (1983) demonstrated that interface design must adopt an approach encompassing broad, flexible criteria for defining its objectives, in order to cater to diversified needs, expectations and satisfaction criteria that are variable over time.

Bennett (1986) follows the same lines as Norman, stressing the importance of encouraging builders to separate the design of the presentation and dialogue with the user from the design of the content of specific applications. In Bennett's view, keeping the interface function separate from the rest of the tool must be encouraged, at both the design and the evaluation stages. It is up to the ergonomist to acquire modeling and prototyping tools for developing and validating the specifications of user-computer dialogue interfaces alone. Bennett thus proposes to isolate the interface from the internal and purely technical aspects to facilitate their adjustment and flexibility.

Several authors challenged this approach of separating the user interface from the internal components of the system. Storrs (1989) makes the distinction between interaction and interface. He thought an interaction is an exchange of information by which two agents each modify the state of the other, while an interface is the information channels which support and influence the content and structure of the exchange. In this perspective, the user interface is not something which can be treated separately from the rest of the system and there is no clearly defensible place to draw the line between front-end and back-end of the system.

Defining the parameters of an applied psychology of the user, Moran (1983) adopts a similar position. Moran said "Psychologically, the user interface is any part of the computer system that the user comes in contact with". Moran presents the psychology of the computer used as a subfield of computer science, which is defined as an extension of

human factors engineering. Moran prefers to adopt a more comprehensive approach from the viewpoint of the psychology of the user as individual (rather than social agent). Moran focuses on the cognitive aspects of the user, which include learning, performing, and reasoning (but not motivational, emotional or personality aspects of the user). According to Moran, it is thus impossible to differentiate explicitly the interface elements from the overall internal structures of the computer tool, since the latter are also likely to influence the user's perception or representation. This user's conceptual model is an integral part of the user interface. For Moran, the issue of creating the parameters of the user-task-machine dialogue is to integrate three ways of approaching the interface question: that of the technician, for whom real progress involves increasing the machine's technical capabilities; that of the designers, who, based on their own intuition, must predict the approach that will satisfy the most users; that of the psychologist, whose role is to ensure reliability, on the basis of objective studies and a satisfactory computer-user interaction results. Young (1981) reaches similar conclusions in his analysis of representations resulting from the use of different pocket calculator models by users with no in-depth technical knowledge of these devices.

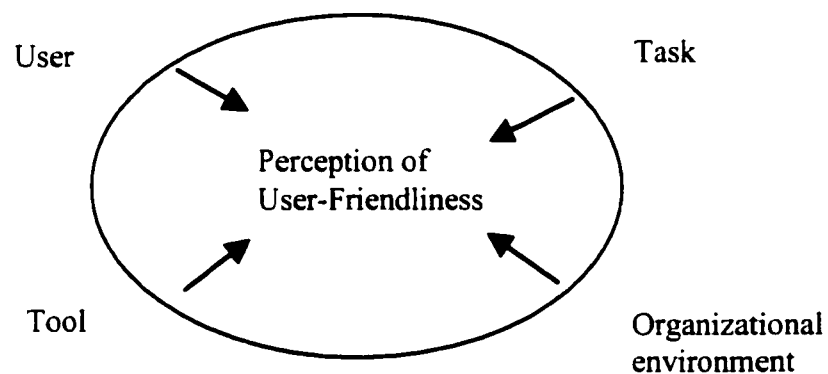
Whether completely separating the user interface model from the internal system or not, the human factor approach to user friendliness is essentially to generate a better overall understanding of the psychology of the user. From this point of view, the concept of user-friendliness implicitly referred to a computer tool-user interface geared to users' needs, in terms of ease and comfort of use, ease of learning, suitability of functions and attractive presentation (Lapointe, 1991).

### **3.1.2 Task Dependency**

User friendliness is also task dependent (Holsapple and Whinston, 1996). An interface style preferred by a user for a particular problem may be seen as cumbersome when that same user faces a different kind of problem. For a certain degree of problem complexity, users would like to maximize the simplicity in stating that problem. Stevens (1983) is in full agreement with this: the user interface is more than the keyboard, screen, menus, messages, command syntax and semantics, etc. According to the author, it is also the mental model which the user has of the system, and it is the structure of the tasks to be performed. The user interface depends primarily on the user's psychological processes and the context in which the computer interacts. Thus, to arrive at a definition of what an optimum interaction would be, one must avoid the user-friendliness of an interface in terms of its intrinsic hardware or software characteristics. What is important is the impact of these characteristics on the effectiveness of the individual-computer system, which may be measured by different performance criteria. In this perspective, the user interface can not be seen simply as discrete components standing side by side in performance of a task. In fact, the user, the interface, the task to be performed and the conditions for performing this task (i.e., the organizational context, constraints associated with organization of work and the specific objective of the task, involving constraints on the style of interaction) are the components of a single set.

So the concept of user-friendliness can not be defined merely by the extent to which a tool is easy to use to perform a series of tasks. By translating all the aspects influencing

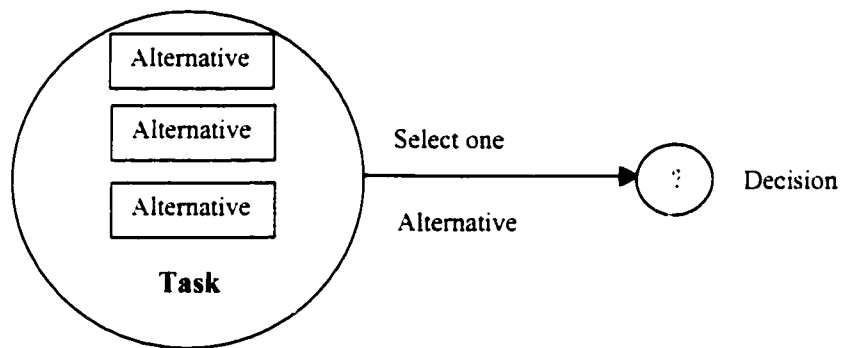
a easy-to-use user interface, Lapointe (1991) points out that user-friendliness also refers to the organizational conditions which contribute to sharpening user's motivation, so that they will be more or less receptive to technological change. What allows one to judge the relative user-friendliness of a user interface is how it meshes with the interaction linking the user-computer and task-organizational dynamic environment. In other words, the concept of user-friendliness to a user interface is based on whether this interface fits in with its link with the user, the execution process, the organization of the task, the formal or informal recognition of the user's efforts, and the culture and practices of the organization. The specific needs of a group of users in terms of interface user-friendliness are thus defined in a complex relationship between their backgrounds in terms of technological know-how, the requirement and characteristics of their tasks, and their relationship (perception, expectations and aspirations) with the organization to which they belong. Fig 3-1 shows the relationship of user, task, tool, organizational environment and the perception of user friendliness.



**Fig 3-1 Ingredients of User-Friendliness**



In the DSS environment, human information processing and decision making are extremely sensitive to task characteristics (Payne et al., 1987). Decision-making is a common thread running through all managerial functions. For example, scheduling is a process in which schedulers produce feasible decisions in alternative selections. In general, a decision is the choice of a strategy of action (Fishburn, 1964). Fig 3-2 illustrates the process of decision-making activities.



**Fig 3-2 Overview of Decision-Making**

Fig 3-2 shows that the decision-making activity results in the identification of some alternative courses of action and will select one of them as the decision based on the content of the task. The number of alternatives identified and considered in decision making can be very large. The DSS is used to help a decision maker cope with the work involved in searching ideal solutions in those alternatives according to the characteristics of the tasks. Which of the alternatives is selected depends on the study of the alternatives. It is an effort to understand the user's purpose and inference mechanism embedded in the DSS. It is obvious that the quality of the communication between the user and the

computer system is decided by the function of the user interface. A user-friendly interface can be very beneficial in supporting the decision making process.

Based on the research of decision making process and task depended properties of user interface, Lapointe (1991) suggested that the question of user-friendliness partly refers to the issue of user's representations, which are themselves constructed from relationships associated with the organization's operational dynamics. The user's representation and its cognitive model will be discussed in the following chapters.

### **3.1.3 Consistency and Adaptability of User Interface**

The concept of consistency is developed from cognitive ergonomics and applied psychology (Barnard et al., 1981). The purpose of using these notions is to contribute to a software design consistent with user's cognitive reference, the representation of the operation and the role of the tool in performing tasks. Consistency designates the interface's internal relationships, which refer to the type of knowledge or representation to which the users refers to carry out their tasks, to determine the best way of executing an operation, etc. In concrete terms, consistency may translate into using similar presentation, logic and language to operate commands in the different software.

The initial proponents of consistency emphasized internal consistency, i.e., consistency within the system (Kellog, 1987). Current directions emphasize both internal and external consistency. The objective is to create standardized user interface software

both within and across product lines, thereby creating a corporate user interface (Nielsen, 1989). The emphasis is on allowing end-users to use the same interaction concepts and techniques throughout many different applications and systems. Such consistency of interaction concepts and techniques is helpful in providing the internal DSS with commands that the PPS and KS can recognize and process. However, to provide this consistency users are required to learn the specific language, icons or other conventions of the interface (Farber, 1989). Farber thought methods of providing interface consistency could be categorized into two levels: (1) consistency between and within systems and (2) consistency within an application. Consistency between and within systems requires a user to be trained in a particular interface that is used throughout the corporation. For example, AT&T uses consistent commands to transmit and manage information between and within their operational support systems. Different computer-based information systems within AT&T use the same user interface. The advantage of a common user interface is that users do not have to learn a new interface when starting a new system and training users only once reduces costs. Consistency within an application means that the same interface is provided for each user of a specific application. All use the same commands and the results are presented in the same format. Each individual user's experience, cognitive style, and other characteristics are not explicitly considered in designing the interface. Examples are the screens provided by popular microcomputer software such as Microsoft Windows and Lotus 1-2-3 (Duce, 1991).

Although consistency is important, users may find it difficult to learn and use (Sankar et al., 1995). A consistent interface may not allow users to complete a task or

fulfil requirements as they would like. In a consistent system, the command language of the interface is constant for all users and each application of the system. A user must be trained, formally or informally, to understand the command syntax of the LS/PS. The command languages are often difficult to learn and use, and may not fulfil the needs of an individual user. A consistent interface also may impede performance of experienced and skilled users. For example, the user is forced to learn the new command and operation style of Windows as a pre-condition for using WordPerfect. The introduction of another version of Windows might force the user to learn another version of the consistent commands/styles. Developers of DSS try to choose the most common terms to represent commands or actions, but these terms may not improve user friendliness or provide the flexibility needed in the DSS. Duce (1991) describes the ineffectiveness of choosing the most common term as the "vocabulary problem." Users employ a variety of words to describe an action or an object. No single access word can be expected to cover more than a small proportion of user's attempts. Duce (1991) concludes, "Simply stated, the data tell us that there is no good access term for most objects." Since even the best possible name is not very useful, it follows that there can exist no rules, guidelines or procedures for choosing a good name for the unfamiliar user. On the other hand, keeping consistency is important for the user interface since it ensures the liability of the communication between the user and the system. However, a consistent interface may not provide the flexibility needed to satisfy the different users' cognitive styles, the users' experience levels, and the different decision approaches supported by the DSS. Thus, from the DSS end user's perspective, a consistent interface can not be an ideal user interface.

The deficiency of consistent user interface is that it is not adapted to individual user's request. If one interactive application presents itself with different user interfaces to different end users or groups of end users, this ability is called adaptability (Schlungbaum, 1997). A user interface with adaptability supports several different dialogue modes or communication styles, allows the user to switch smoothly and naturally between dialogue modes or communication styles at any time, and makes it easy for the user to learn the different interaction methods. Adaptive user interface is needed when one interactive system has to perform different tasks or is operated by different users with individual characteristics or preferences. According to Schlungbaum (1997), there are three kinds of adaptive user interfaces:

- Adapted user interface: the user interface is adapted to the end user at design time.
- Adaptable user interface: the end user changes the functionality and/or some characteristics of the user interface manually.
- Adaptive user interface: the user interface changes its characteristics (presentation layout as well as dynamic behavior and availability of application services) dynamically at run time according to the end user's behavior.

In an adaptable interface, the inputs and outputs of a user's queries can be displayed on screen, printed in graphical or textual form. The choices presented in the PS and LS of DSS reflect the cognitive preferences of the user and the complexity of the task being performed (Holsapple and Whinston, 1996). User-controlled interfaces give the users an opportunity to control the method of communication with the system (control of the LS), and the presentation of the results (control of the PS). A self-adaptive interface

automatically adjusts to the users' preferences and tasks. However, based on current technology, it is almost impossible to build a system which can self-adapt to a change in problem domain or new technology (Stary and Peschl, 1998).

Based on the surveys of the different approaches to the concept of user-friendly user interface, Lapointe (1991) concluded that: "To a large extent, a user-friendly interface is expected to be conducive to speed and ease of learning, with respect to the context of use, the power of the software, and differentiation between the acquired knowledge and the knowledge actually used. User-friendliness also concerns the retention of learning (depending on frequency of use, ease of identification of interface codes, etc.) and the transferability of this learning. A partial definition would therefore consist in saying that a user-friendly interface requires minimal effort from the user in terms of training and adjustment, that it requires basic knowledge, shared by the greatest number."

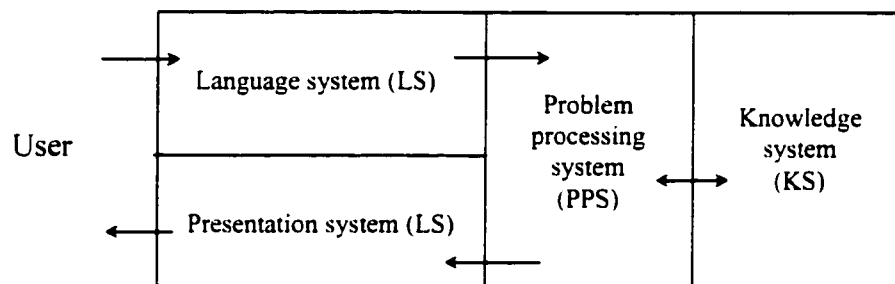
The purpose of this paper is to find a way to build a consistent, adapted and adaptable DSS user interface for short term scheduling purpose with the ability to be independent of the scheduling algorithms used in the background.

## **3.2 DSS User Interface Model**

### **3.2.1 Characteristics of DSS-User Interaction**

Storrs (1989) makes the distinction between interaction and interface. According to Storrs, an interaction is an exchange of information by which two agents each modifies

the state of the other (i.e., the composite of the states of an agent's beliefs, goals, plans, intentions, attitudes, values, dispositions and physical state), while an interface refers strictly to the information channels which support and influence (through their characteristics and the constraints they present) the content and structure of the exchange. In order to establish a user interface model of DSS, Santos and Holsapple (1989) tried to clarify the mechanism of the interaction happened in the decision making process based on the generic framework for DSS shown in Fig 3-3.



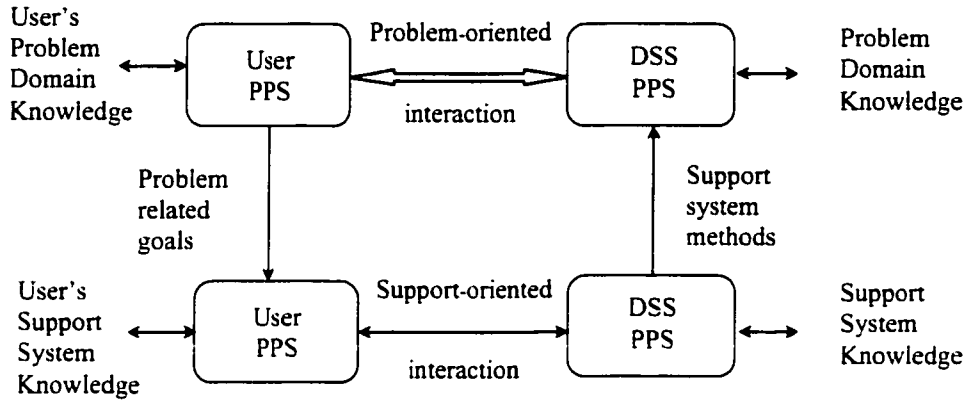
**Fig 3-3 A Modified Generic DSS**

Traditionally, the user is viewed as part of the systems environment (i.e., a black box), providing inputs to and receiving outputs from the system, and vice versa (Bennett, 1983). Although it is generally recognized that users interact with a DSS, little attention has been paid to the interaction (Sprague and Waston, 1986). One consequence of a black box view of the user is that DSS interface designs are not easily customized to individual users. For example, DSS interfaces generally do not distinguish between novice and experienced users of the DSS, although different interfaces are necessary for users with different levels of experience. Furthermore, in unstructured problem solving situations, the user is an adaptive system, learning from DSS interaction about both the decision

problem and the use of the DSS. Interestingly, user learning about the decision problem can result in changes to user interaction with the DSS (Santos and Holsapple, 1989). For example, after observing last month's sales data, a user may wish to build and use a new simulation model. As a consequence, use of a DSS may cause users to move from DSS interactions with which they are experienced to those with which they are novices.

So, in order to provide a friendly interface to a DSS that provides extensive capabilities, Santos and Holsapple (1989) suggested that interface design must consider both the user and the support system. It is useful to consider the decision system as a user-support system aimed at reaching a decision in response to a problem. The user is viewed as an active entity in this system. Within the decision system, problem solving is an interactive process between a user and a support system. The interaction is in the form of messages that are sent from one party to the other. Messages from a user to the support system are referred to as the input interface, the LS. Messages from the support system to the user are referred to as the output interface, the PS. In this interactive environment, both the user and the support system possess problem domain knowledge, which is relevant to the current problem (Santos and Bariff, 1988). In addition, meaningful communication between the user and the support system requires that the user possess support system knowledge, which the user must bring to the DSS, and problem processing capabilities that allow the user to use that knowledge to send appropriate messages. Interaction within the DSS is shown in Fig 3-4.





**Fig 3-4 A Model of User/Support System**

Fig 3-4 shows that interaction between a user and the support system can be viewed as a communication system with interaction occurring at two levels. And each requires the user to possess specific knowledge. At the upper level, the user's problem processing system (PPS) uses problem domain knowledge to identify the problem related goals that the support system may satisfy. At the lower level, the user's PPS uses support system knowledge to obtain information relevant to the problem-related goals generated at the upper level. In terms of user DSS interaction, there are two types of interaction: (1) problem-oriented interaction, and (2) support-oriented interaction. Problem-oriented interaction is directly related to the user's decision problem. Support-oriented interaction is initiated for each set of problem related goals, which often are the result of problem-oriented interaction. However, support-oriented interaction is not directly related to the user's decision support system. Instead, it includes the interaction necessary to satisfy problem-related goals. As an example for this interaction, suppose that a shop floor manager is faced with the problem of changing a scheduled assembling plan. At the upper level, the manager's PPS may determine the target of the last modification and

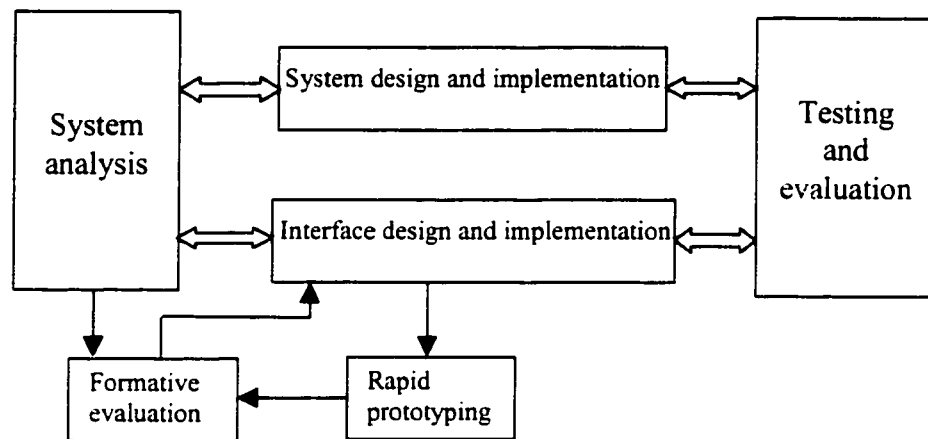
actual schedule information that is desired. In order to obtain this information, the schedule goals must be translated by the manager's PPS into an appropriate request, using support system knowledge and/or obtain support system help to do so. For sending an appropriate message, user-DSS interaction between the two knowledge support systems may be necessary, even though the manager has the necessary support system knowledge. Thus, for a given problem-related goal generated at the upper level (e.g., determining last change's effect and actual planning goals), there generally is user-DSS interaction at the lower level. In the above example, the interaction to create the schedule belongs to the problem-related interaction, while the interaction to retrieve the historic scheduling data belongs to support-oriented interaction.

### **3.2.2 The Framework of DSS User Interface**

The effort on studying interaction styles and user interfacing features of software applications goes back to the mid-60's. Newman's light handle concept in new interaction objects (Newman 1967) and more significantly his work on devising a language to define interaction (Newman 1968) are the outstanding examples. They introduced possibility of new alternatives in Human-Computer interaction design and development. Research and development of user interfaces as an independent component in software systems followed more seriously by researchers during the seventies, particularly in design frameworks and separation issues. Examples of such efforts can be found in Edmonds (1978), Newman (1978). With the development of cognitive science, researches on mental process and human cognitive action show that human factors have

significant effects on user-computer interaction. Norman (1983) presents some principles for designing user-computer interface based on the study of human factors. He encouraged the builder to separate the design of the interface from the internal component of the system. The purpose is to isolate the interface from the internal/purely technical aspects to supply much more flexibility. However, research conducted by Storr (1989) suggested that interaction and interface are two different concepts. He concluded that no clearly defensible line can be drawn between the front-end and back-end of the system. Several authors who hold the same point of view also challenge the approach of separating the user interface model from the system. According to Liang (1987), although design of an effective interface remains an art rather than a science because so many issues are involved, some general guidelines do exist. He divided interface design guidelines into three categories, computer technology, task requirement, and user characteristics. Hix and Hartson (1993) included the user interface design into the life circle of system analysis. The parallel paths (shown in Fig 3-5) for development of interface and non-interface components of an interactive system will enforce both the similarities and consistence. Consistency is one of the most significant factors affecting usability, because users expect certain aspects of an interface to behave in certain ways. When that does not happen, it can be very confusing. However, consistency by one criterion can conflict with consistency by another. The guideline of consistency can even conflict or trade off with other guidelines. Sankar et al. (1995) thought a consistent interface was not an ideal user-friendly interface. Besides consistence, a user-friendly interface should give the users an opportunity to control the method of communication

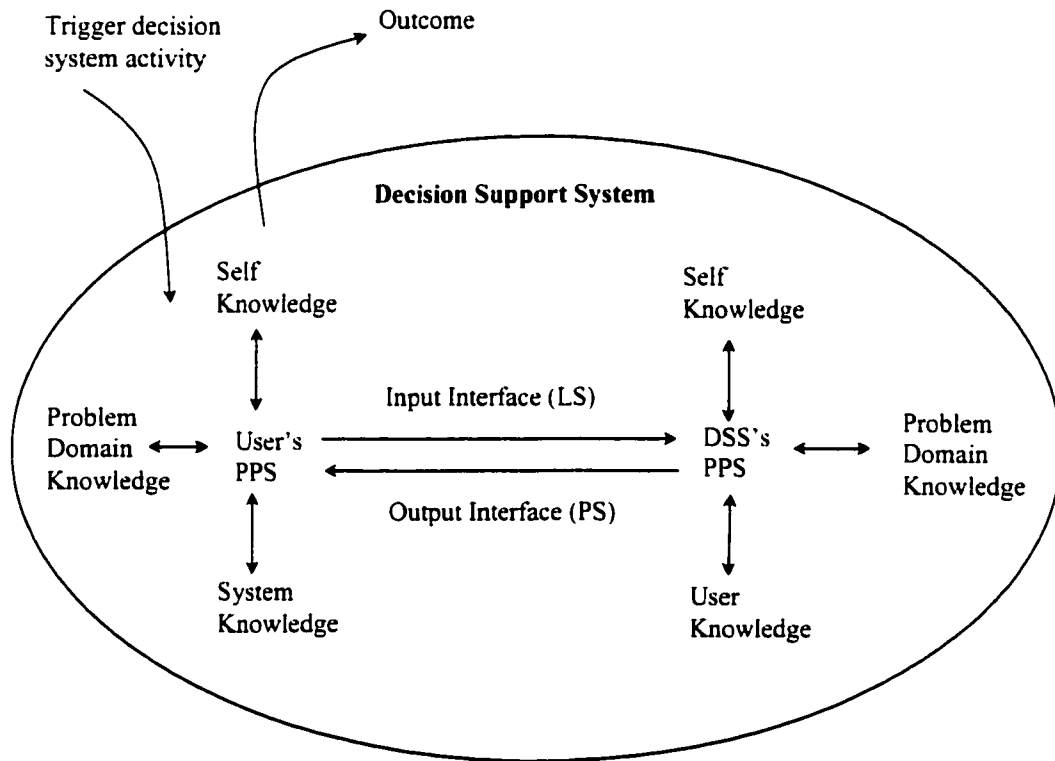
and adapt to user's preferences and tasks. Consequently, an easy-to-use interface will minimize the cost and effort of the interaction between the users and the system.



**Fig 3-5 Prototype User Interface Development**

In this perspective, Tyler and Treu (1989) propose an adaptive interface design based on the “modularity” of the interface system, which resides in a separated module that may be easily modified independently of the rest of the programming. The user is thus offered the opportunity to organize and modify the input content and output presentation. The interface can provide some error messages and operating instructions in a personalized way. The philosophy underlying this interactive interface design is to encourage non-expert users to learn and give them the opportunity to evolve in their use. Here, user-friendliness is seen in terms of adaptability and need to respond to a multitude of needs. However, because of the technological limits, an automatically self-adapted system is almost impossible to be realized today (Stary and Peshl, 1998). Liang (1987) suggested adaptive design could be separated into three approaches: (1) user-involved adaptive design, (2) user-controlled adaptive design, and (3) self-adaptive design. Schlungbaum

(1997) gave another set of definitions for the user interface with adaptability. They are adapted user interface, adaptable user interface and adaptive user interface. Detailed explanation can be seen in Chapter 3.1.3. It seems that the involvement of user for an adapted and adaptable user interface is inevitable. This is also true when a scheduling system tries to adapt to the user's preference while the problem domain and technology are changed. In order to incorporate the users and their knowledge into the DSS, Santos and Holsapple (1989) conducted the research on the interaction between the user and the system. Based on their two level interaction model (Fig 3-4), they suggested a framework for DSS interface showed in Fig 3-6.



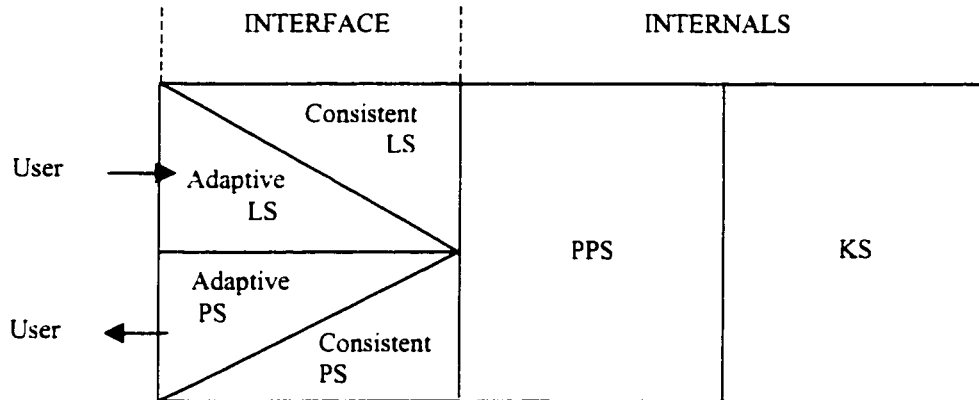
**Fig 3-6 A Framework for DSS User Interface**

This framework suggests that DSS interfaces can be improved if the KS in a DSS contains the same categories of knowledge that the user possesses. In addition to problem

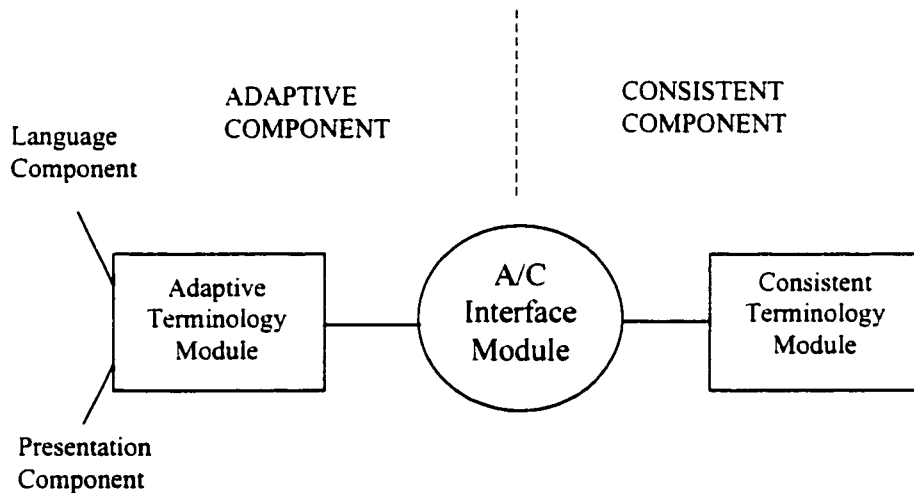
domain knowledge, the DSS should include two other types of knowledge: (1) user knowledge, and (2) self-knowledge. Further, the PPS must be capable of using all three types of knowledge in order to be able to adapt itself. By maintaining these two additional types of knowledge, the DSS interface can be adapted to meet the needs of both experienced and novice users. In addition, it has a potential for adjusting to accommodate user preferences.

The framework shown in Fig 3-6 separates user's knowledge system and DDS's knowledge system into three parts and connects the user with the system by PS and LS. However, Santos and Holsapple (1989) did not explicitly specify the structure of knowledge representation underlying this framework. Without a proper knowledge management system, it is not very easy to implement this framework. Sankar et al. (1995) noticed that user interfaces are extremely sensitive to both the user characteristics and task characteristics and user-involved operation is necessary for a DSS system to find a satisfactory solution. However, prior models of DSS user interface are oriented largely toward programmers and system designers rather than end-users. And problems arise in the situation when designers try to provide both adaptability and consistency because of the lack of the communication between the adaptive module of the interface and the consistent module of the internal components. Based on the framework suggested by Santos and Holsapple (1989), Sankar et al. (1995) propose a model that allows an end-user to converse with the DSS using an adaptable interface and, in parallel, allows effective communication with the internal PPS and KS using consistent internal

commands. Fig 3-7 shows the architecture of the proposed model and Fig 3-8 is the generic description of the proposed model.



**Fig 3-7 Proposed DSS User Interface Model (Sankar et al., 1995)**



**Fig 3-8 Generic Description of User Interface Model**

This model divides each user interface component (LS and PS) into adaptive and consistent sections. The adaptive section adapts to the unique requirements of a user and the consistent section provides consistent commands to the internal of the DSS. The

model has two modules: an adaptive terminology module and a consistent terminology module. These are connected by an Adaptive/Consistent (A/C) interface module (Fig 3-8). A user's dialogue with the system and the resulting presentation by the DSS is adapted to the unique characteristics of the user using the adaptive terminology module. The PPS and KS of the DSS are provided consistent terminology by the consistent terminology module. The advantage of this model lies in the complete separation of the unique interface of a user from the PPS and KS. The expected implementation of the model will be discussed in the following chapters.

### **3.3 Proposed Implementation Heuristic**

#### **3.3.1 Related Research**

Since the model is clarified, it is important to develop implementation procedures to demonstrate the feasibility and technical validity of this model. Diaper (1989) introduces a method called knowledge representation grammar (KRG) that allows a task analysis hierarchy to be written in a sentence form. KRG provides a means of describing the route of each task step. The KRG sentences are intended to inform the analyst what are the common classes of activity associated with a task. Thus, in designing a computer system and its interface, it is reasonable to expect that frequently occurring activities should specify the primary functionality of a system and that the users are supported by the human-computer interface so that they can access such functionality easily but also safely. If the functionality of the system is well specified and implemented then the common activities should be simple and easy to be embedded into the user interface. The



crucial aspect of KRG sentence analysis is the understanding of the grammar of KRG sentences. It is obvious that only well trained designers can use KRG method for user interface design. Besides, KRG method is derived from the process of task analysis. Study in ergonomics and cognitive science shows that properties of user interface are affected by both the task characteristics and user characteristics. KRG method does not include the human factor into its analysis process. So the interface designed by KRG method can not adapted to the end user according to the user's preferences.

Harrison and Thimbleby (1990) believed that much of the early work in human computer interaction had been encumbered by a lack of appropriate abstractness or applicability to the design process. Since human computer interaction is a multidisciplinary activity, it is difficult to express the contributions of various approaches. A formal method, defined by Harrison and Thimbleby (1990), produces a precise framework in which the role and scope of these models may be clearly understood for interactive system interface design. The goal of this method is to formalize the notion of task, the actions that the user must carry out in order to complete this task, and observe the nature of consistency to regularize action sequences. The core of this method is so-called task action grammar (TAG), which provides a structure and analysis for a notion of competence. The authors thought it would be easy to predict and build the characteristics of the interface as long as the interface has been formally described. Based on the TAG framework, the increased interest in consistency can be easily satisfied. For example, the task "draw a square" can be defined by the semantic components as follows:

Effect	=	add
Type of object	=	rectangle
Constraint	=	yes

When actions for the task are added, the task “draw a square” can be redescribed as:

Task	[Effect	=	add
	Type of object	=	rectangle,
	Constraint	=	yes
	] := select-tool [Type of object = rectangle]		
	+ press SHIFT + place mouse ...		

In formal method, every predicted action will be abstracted as a set of rules using previous procedure. However, Stary and Peschl (1998) concluded that the internal knowledge of the human would be lost not only in the moment of externalization but also in semantic analysis process. So, the interface designed by formal procedure may not satisfy the user’s request because some of the knowledge of the user is lost in the TAG formalizing process.

Hix and Hartson (1993) notice most of the existing user interface design methods are task-oriented. Task-oriented design methods impede the direct feedback and communication between the user and the system. In order to overcome the deficiencies that exist in those methods, they suggested a new method, user action notation (UAN), to describe physical (and other) behavior of the user and interface by user and task-oriented notation. However, the human decision-making process is connected not only to the physical behavior but to the human mental process. For example, scheduling is a mental logical process other than a physical action. The attention on describing physical behavior of UAN method limits its use for designing user interface of scheduling system.

The brief overview of the existing methods for user interface design shows that none of them is suitable for designing a flexible user interface for scheduling system, which is consistent to the internal components of the system and adapted to the preference of the end users. Most of the existing methods do not include human factor in their design procedures. Semantic and syntactic analysis is the basis for most of the user interface design methods. Stary and Peschl (1998) pointed out that semantic shifts in the process of transferring knowledge from humans to computer systems and vice versa occurred due to a variety of reasons:

1. Natural language is the major instrument to externalize human (internal) knowledge. However, any kind of language is capable of externalizing only a small fraction of the semantics that humans have in mind when they try to externalize a particular chunk of knowledge by making use of language. Hence, the knowledge is not only lost in the moment of externalization, but also some kind of semantic distortion occurs. Due to the individual differences in experiences of humans, different receivers of the language utterance will interpret the syntactic environmental patterns differently. Although both parties are referring to the same environmental pattern/syntactic structure, they have different meanings when they are using it or referring to it.
2. Whenever someone/thing is externalizing behavior through (language) symbols, and someone/thing tries to interpret these meaningless artifacts, the semantics for different users or designers might differ considerably. Although they are confronted with the same symbol, icon, graphical representation etc., these artifacts might trigger different internal representations and semantics in the

participating cognitive systems. As a consequence, the occurrence of semantic shifts can not be avoided in principle.

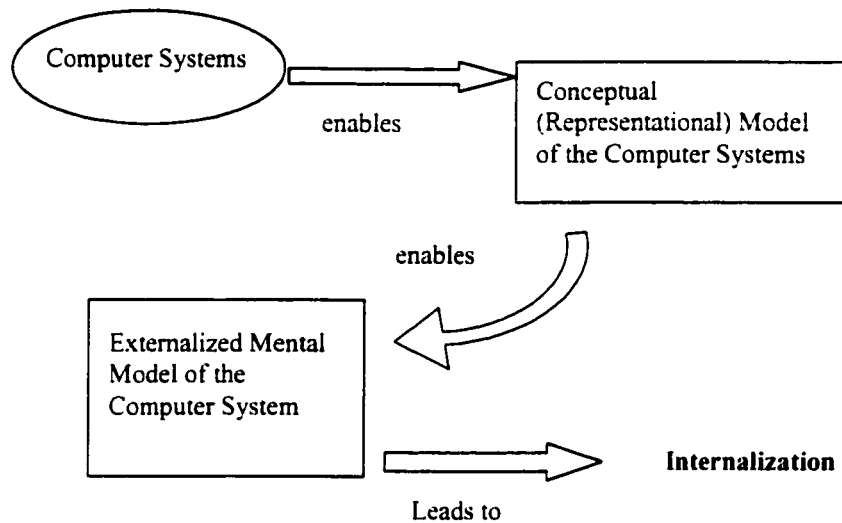
3. Despite all attempts to introduce semantic features into symbol systems, natural language is deprived of its final semantic features and dimension in the process of formalization. Hence, the distortion is taken even one step further in the process of formalizing natural language into purely syntactic and formal structures. Losing the semantic dimension implies that the process of interpretation may lead to unintended semantic shifts.
4. There is no steady semantic flow. Semantics is always system-dependent and communication is based on mutually adapting the individual use of symbols.

Consequently, the idea of formalizing the interaction between the user and the system by semantic/syntactic analysis can not create a user-friendliness user interface.

### **3.3.2 Cognitive Engineering and its Application in User Interface Design**

Cognitive engineering, which researches the human behavior in complex worlds (Woods and Roth, 1988), reveals that human-computer interaction is a multi-level, multi-channel, and multi-modal interaction process between the two representation systems. The complexity of the interaction makes the effort that tries to avoid distortion during the process of semantic analysis impossible. In traditional cognitive model of human-computer interaction (showed in Fig 3-9), human-computer interaction is understood to be driven by the execution of actions to achieve goals (Stary and Peschl, 1998). In this model, the goal of the task is the center. All of the modules are established to achieve the

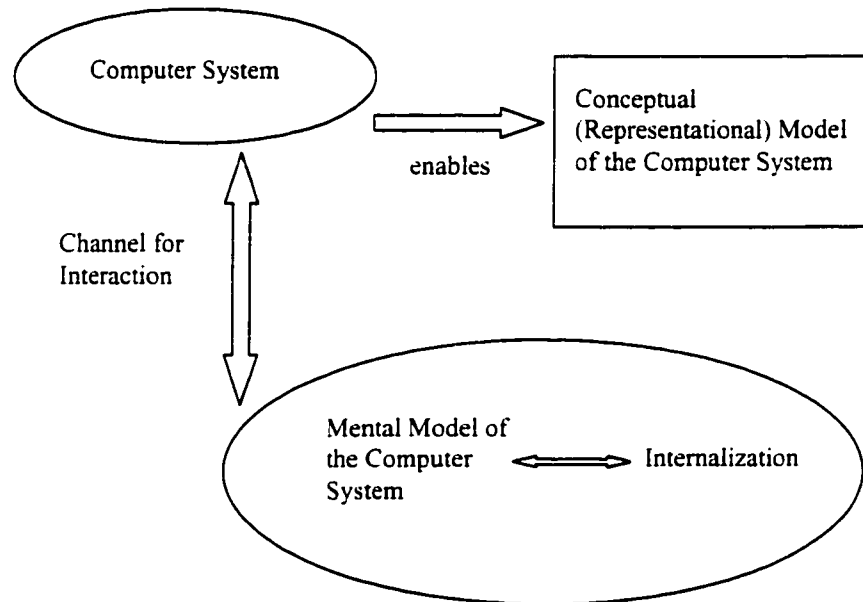
goals defined by the designers. The externalized mental model of computer system (Fig 3-9) is created by matching the user's goals to the designer's goals. In this model, task other than user becomes the center of user interface developing process.



**Fig 3-9 Traditional Cognitive Model of HCI**

In questioning the understanding of human-computer interaction as an exchange of data or information based on the goal-driven execution of actions, Stary and Peschl (1998) proposed a cognitive model which matches the behavior parameters of the two representation systems, the user and the computer system, along task accomplishment instead of matching developer and user goals explicitly. The new model is shown in Fig 3-10. In this perspective, human-computer interaction becomes the mutual adaptation of two systems involved, the computer system and the cognitive system of the user. In modeling the user this way, cognitive modeling will not have to address an externalized and internal mental model of end user any more. The transition between explicit and tacit

mental representation will be resolved by unifying both representations and behavior patterns.



**Fig 3-10 A New Cognitive Model for HCI**

As a consequence, developing a human-computer interface always requires the finding and construction of a representational and computational structure that meets the constraints given by the cognitive dynamics of the potential users. Cognitive engineering provides feasible techniques to solve the addressed problem (Andriole and Adelman, 1995). Cognitive engineering represents the attempt to base system design on cognitive research findings regarding how people think, to aid but not replace the human problem solver, to design systems that utilize human strengths and compensate for human cognitive limitations, and to more systematically integrate display formats and analytical methods with intuitive thought. Cognitive-based design method is a user-centered

interface design process. Human actions in the decision-making process are thoroughly described by the cognitive scientist before each rule can be established. After a series of experiments, Powals et al. (1995) confirmed that system performance and adaptability would be improved with the implementation of the cognitively engineered interface. They suggested a set of executable and easy-to-use implementation rules based on four tested hypotheses and the findings from cognitive process research. The four hypotheses are:

1. Human performance will be more accurate with the cognitively engineered interface.
2. Human performance will be faster with the cognitively engineered interface.
3. Workload associated with the cognitively engineered interface will be less.
4. Users will prefer the cognitively engineered interface.

The rules domain include:

- Situational awareness
- Schematic incorporation of new information
- Context-dependent reconstructive retrieval
- Top-down and bottom-up processing
- Limited processing ability
- Inflexible representation in problem solving
- Cognitive load
- Interference
- Problem solving
- Simplicity

Each rule's domain consists of several conceptual hypotheses and operational hypotheses. These specific hypotheses were derived by the translating of the cognitive findings into characteristics of the interface. Sometimes one cognitive finding suggested more than one conceptual hypothesis, which then indicated more than one operational hypothesis. When implementing these rules into user interface design process, the conceptual hypotheses should be translated into the operational rules. Implementation example of the rule set will be shown later. A detailed description of the rule set can be seen from the Appendix of the thesis.

### **3.3.3 Task Analysis Method**

Cognitive findings supply a method for user-centered user interface design and enable the user interface to adapt to the user's preferences. On the other hand, in order to keep the interaction of user interface and the internal component of the system consistent, task analysis is necessary. Task Descriptive Hierarchy (TDH) heuristic, presented by Diaper (1989), is the heart of Task Analysis for Knowledge Descriptions. TDH includes two types of entity lists: specific objects and specific actions. The specific object list contains all the objects that are relevant to the performance of the tasks or set of tasks. The specific actions are the behaviors performed by a person that are directed towards specific objects. Generally, there are few different specific actions that people carry out in scheduling tasks since most of the decision-making activities belong to mental logic process. So specific actions list can be omitted from the TDH for scheduling problems analysis. Diaper (1989) suggested a three-step heuristic for establishing the TDH.



- The first step is to construct a very high-level description that will describe all the specific objects. The highest-level description is so general that, by itself, it is immediately recognized to be too high for any useful analysis.
- The second step is to generate a hierarchy which has the property that every low-level specific object can be described by a single unique route through the TDH. The important thing that needs to be followed is that each node in the hierarchy should have only a small number of immediately lower-level nodes linked to it. This means that the analyst should work towards building a TDH which is deep (i.e. has many levels) rather than shallow. If the heuristic is not followed, a shallow, flattened hierarchy will lack analytical power because there will be little opportunity for redescription of the task observation data at a useful level. The cost of increasing the depth of the TDH is an expansion of breadth at the lower levels because of the need to repeat sections of the hierarchy under different higher level nodes.
- The last step is starting building relatively shallow hierarchy and then expanding it in depth. An advantage to this approach is that it allows the analyst to check if the criterion is met by a single unique route through the hierarchy.

TDH must be a true hierarchy, rather than a network, such that all lower levels can be redescribed as higher ones without cross-linkages or loops. There are three types of relationships which could be used, mutually exclusively, at any level. First is the logical XOR relationship which specifies that a specific object can be redescribed as either one type of higher-level entity or another. Second is AND relationship that allows several properties to be possessed at a level. The last is OR relationship where there may be more

than one property possessed at a level but only one is necessary. TDH is extremely helpful to any kinds of task analysis. The exhausted task lists prevent conflicts among internal commands.

### **3.3.4 User-Centered Implementation Heuristic (UCIH) for User Interface Design**

The review of the literature shows that the proposed DSS interface model (Fig 3-7 and Fig 3-8) can not be implemented by the existing procedures, which should possess both adaptability and consistency characteristics and can help designers out of the complex linguistic analysis procedure. In order to compensate the deficiency of those methods, a new user interface implementation heuristic should satisfy two hypotheses:

1. User interface design is extremely sensitive to both the task characteristics and user characteristics.
2. Adaptability and consistency are two of the important properties to guarantee the user-friendliness property.

For creating a consistent user interface, task analysis and its method should be incorporated in the process of user interface design. Task analysis is also an important procedure for system development. Therefore, user interface design can not be separated from the system development process (Moran, 1983). At least, the process of user interface design should be parallel with the development of the whole system. In this perspective, some methods of system analysis are also suitable for user interface design.

For creating an adaptive user interface, designers have to find a proper way for representing the user's knowledge. This problem can be solved when resorting to the cognitive engineering and its findings. The cognitive rules set suggested by Powals et al. (1995) can be easily transferred from conceptual hypotheses into operational rules when applied to construct the knowledge representation of the users.

Besides, building a user interface is an iterative process other than a "one way" process. Designers usually establish a quite simple prototype user interface first based on high-level task analysis and basic user's requests. Then the user interface will be evaluated, revised with the development of user and task analysis. In the development cycle, more and more functions will be added or modified until both the user's and the system's requests are satisfied.

The following is the heuristics proposed by the thesis for designing friendly user interface based on the model showed in Fig 3-7 and Fig 3-8:

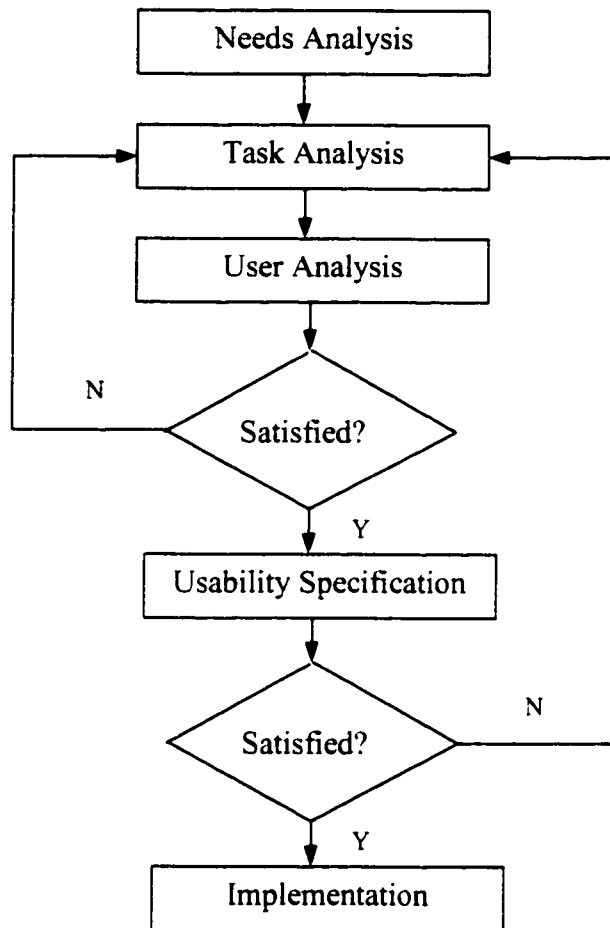
The seven-step heuristic can be described as follows:

1. Needs analysis: Determine the basic goals and features desired for the application system. The result is an external view of what a user will be able to do with the system.
2. Task analysis: Provide a complete description of the tasks, sub-tasks, and methods involved in using the new system. Identify necessary resources for performing the

tasks. Build a TDH (there are cases where only specific objects will be necessary when there are few relevant actions).

3. User analysis: Define the user characteristics. Clarify the user's needs and explore the skills and knowledge that users bring into the system. Translate the conceptual cognitive-based hypotheses into operational rules. Go to step 5 if all of the rules are satisfied.
4. Revise TDH according to the cognitive –based analysis in Step 4.
5. Usability specification: Define the usability attribute of the user interface. This procedure gives a criterion to measure whether the development process is converging towards a satisfied end point. Go to step 7 if the end point is reached.
6. Go to step 2. Revise TDH if not all of the user's requests have been included in the TDH. Otherwise, go to step 7.
7. Implementation.

Fig 3-11 shows the logical sequence of the heuristics.



**Fig 3-11 Logical Flow of the Heuristics**

### **3.3.5 Conclusion**

In this chapter, a precisioning user interface design heuristic is proposed. The purpose of the heuristic is to establish a consistent user interface with the ability to adapt to different user's requests. This heuristic, which combines the findings of cognitive engineering and task analysis methods together, can be easily embedded into the system

analysis and development process. An example of implementation of this method will be showed in the next chapter.

## **4. Case Study: A Prototype User Interface for Schedule Modification Operation**

### **4.1 Introduction**

Shop floor level managers usually need an easy-to-use scheduling facility to solve short-term scheduling problems. However, the complicated environment of the shop floor and some unexpected events (e.g., machine breakdown, operation tardiness, rework, etc) make the scheduling problem quite difficult to be solved. Usually, the scheduling system implemented in the manufacturing company is established according to Client/Server structure in which the client and server are independent components and they communicate with each other through the Application Programming Interface (API). Under this structure, complex and time consuming tasks (such as data mining, production planning, scheduling) can be finished on the remote server which possesses large data saving facilities and high performance of data processing ability. The client or the end-user terminal is usually used for data retrieving or simple data input. However, on the shop floor, unexpected events happen every day. Shop floor managers need a “handy” facility to deal with these changes. This facility can help the manager modify the existing schedule with the least effort and communicate with the remote server or a local DSS scheduling engine to create a feasible schedule in a short time.

In the next section, a prototype user interface that helps shop floor managers make quick scheduling modifications will be described using the proposed user interface design heuristics.

## 4.2 Framework of the Prototype User Interface

Suppose the end-user terminal has a local DSS connected in the background (This hypothesis does not mean that the UCIH can not be used under client/server structure. It is just for lightening the programming burdens). Usually, the scheduling results are saved in a table with several fields that hold basic scheduling information. The data structure of the table is shown in Table 4-1.

### Job and resources information

Field Name	Job ID	Operation ID	Work Center	Machine Center
Data Type	Text	Text	Text	Text

### Scheduling information

Field Name	Start Time	End Time	Start Date	End Date	Segment
Data Type	Date/Time	Date/Time	Date/Time	Date/Time	Text

**Table 4-1 Data Structure of Scheduling Result Table**

Job ID and Operation ID identify each single scheduling item. Work Center and Machine Center assign the resources to each operation. Each of the operation is broken down into segment (like Setup, Run etc.). The segment field is necessary when the previous operation releases the occupied resource and the same resource will be assigned to the next operation. This table shows the scheduling result. From another point of view, this is the scheduling knowledge given by the scheduling system.



On the user's side, the modification purpose of the user also can be translated into a table-format information. The proposed data structure of the table for saving the modification information is showed in Table 4-2

Job and resources information

Field Name	Job ID	Operation ID	Work Center	Machine Center
Data Type	Text	Text	Text	Text

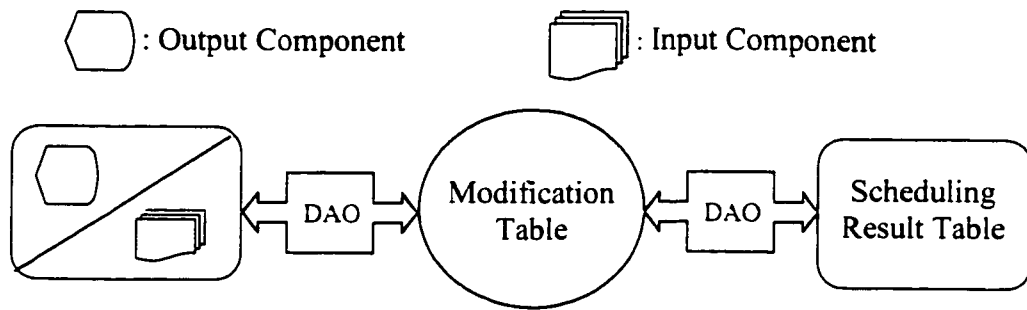
Scheduling information

Field Name	Start Time	End Time	Start Date	End Date	Segment
Data Type	Date/Time	Date/Time	Date/Time	Date/Time	Text

**Table 4-2 Data Structure of Modification Information**

Table 4-2 can be viewed as the modification knowledge given by the user. Comparing the data structure of modification information with the data structure of the scheduling result, there is no structure difference between the two tables. The purpose of constructing the modification data structure identical to the scheduling data structure is to keep the messages consistent with the internal commands of the system when they are transferred from the user to the system or vice versa. The data structure of modification information does not imply that a spreadsheet will be the unique presentation form to the end-users. Choosing what kinds of presentation forms depends on the user's cognitive preference analysis. The modification table can be viewed as the Adaptive/Consistent Interface module that connects the adaptive module linked with the end users with the consistent module linked with the system. In other words, the modification table can translate the user's requests into consistent internal commands, or vice versa, translate the internal

messages into a kind of user-preferred data presentation form. Fig 4-1 shows the framework.



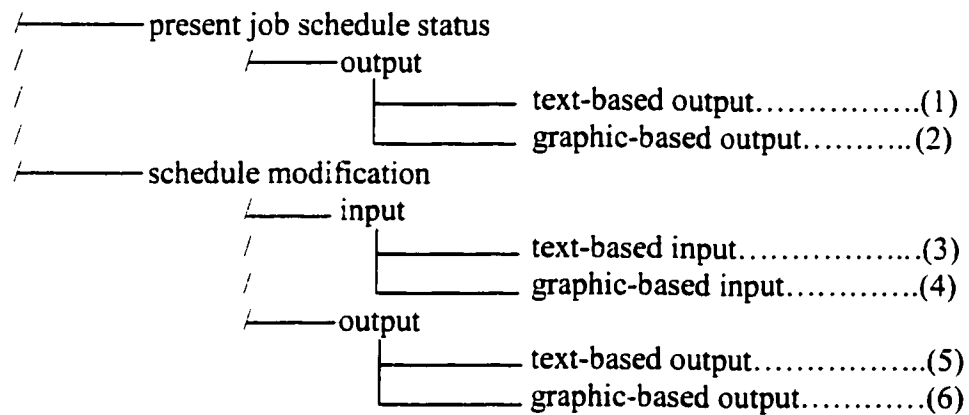
**Fig 4-1 Framework of the User Interface**

Some notes for the framework are:

- The DAO is the abbreviation of Data Access Object, which connects different kinds of database objects through a standard interface.
- The format of the input/output component depends on the preference of the user.
- The structure of the modification table is subject to change since the effects of human factors have not been revealed.
- The scheduling result table is connected with the system database by DAO. This makes the interface model independent from the background scheduling mechanism since the scheduling result table has no schedule algorithm information. In the shop floor, managers will not put much of the attention on manipulating the database system. The background-scheduling algorithm is a “black box” to the managers. Incorporating detailed scheduling information is meaningless to the shop floor managers.

### 4.3 Implementation of the User Interface Framework

According to the proposed user-centered implementation heuristic (UCIH), the first step is needs analysis. This has been done in §4.1. The purpose is to design an easy-to-use facility to help the shop floor manager modify the existing schedule with the least effort. The second step is task analysis. The TDH is built as follows:



**Fig 4-2 TDH\_1**

The conventions adopted in the TDH are as follows: the hierarchy is read from left to right with the highest level on the left; nodes at the same level are spatially located under each other; XOR relationships are characterized by a vertical line '|', OR relationships by a brace '{', and AND by a slash '/'. What has been established here is just the high level TDH. The TDH will go deep in the following steps. In addition, the following points are needed to be considered when constructing TDH:

- The two main functions of the interface are to enable the user to browse the schedule data and modify the existing schedule data.

- **Basic browsing and modification interface is either text-based or graphic-based.**

The third step for UCIH is user analysis. In this case, the user usually is a shop floor manager. The workload for inputting data and modifying the schedule should not be heavy. According to the cognitive findings (detailed contents are listed in the Appendix. “OH” is the abbreviation of Operational Hypothesis, “CH” is the abbreviation of Conceptual Hypothesis) and user characteristics analysis, the following improvements of the user interface should be considered:

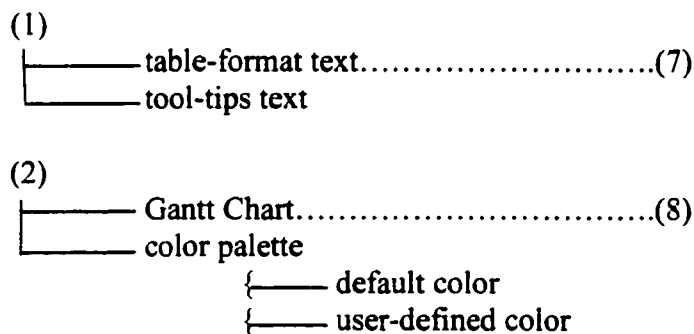
- According to OH 4.1, graphic display can help to limit data-driven tasks. So, the schedule information will be displayed by Gantt Chart. Gantt Chart is selected because it is the most popular graphic tool used in scheduling field. Gantt Chart fits most of the users’ cognitive conventions and it will enhance user-computer performance.
- According to OH 7.1 and CH 2, text-based display is also available to the end user. There are two ways for displaying text-based information: (1) separated forms: the table-format forms display all of the necessary schedule information. (2) tool-tip text: this is a flying window that indicates necessary schedule information when user operates on the graphic displays.
- According to OH 1.2, color coding can reduce uncertainty for graphic display. And according to OH 3.2, consistently meaningful grouping results in better user-computer performance. So, the color of the bars of Gantt Chart can be defined according to the Job ID and Operation ID. Default color will be set if the user does not select the color based on individual preference. Users also can call

default color set back if they are not satisfied with the color they have selected.

Since different users may have different color preference, the knowledge of user's color preference should be included into the system. So, an additional field with name pColor and the data type Long Integer will be added into the table of scheduling result. In order to stay consistent, the same field will also be added into modification table.

The above analysis on user's cognitive preference brings some changes to the TDH.

The extension work for the TDH in Fig 4-2 is shown in Fig 4-3



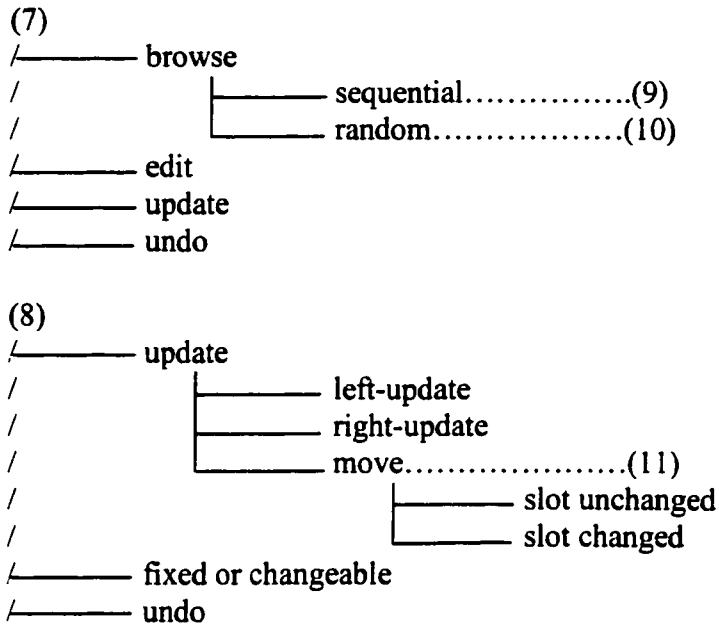
**Fig 4-3 TDH\_2**

The sub-tree of (3) and (5) in Fig 4-2 is as same as the sub-tree of (1). The sub-tree of (4) and (6) is as same as the sub-tree of (2).

The next step of the design, according the UCIH, is usability specification. One of the main objects of this interface is to let the user make simple modifications quickly and efficiently. Underlying this point, the following actions should be considered:

- Text-based browse, edit, and update operations must be available.
- Graphic-based edit can be triggered according to the position of the mouse and the button selection (e.g., when the mouse is put on the left side of the bar and left button of the mouse is pressed, the left-change operation will be triggered. That means the schedule's start time will be changed).
- The users can select the contents for displaying. This can be done by constraining the schedule start date and schedule end date.
- The user can "undo" the operation which is not proper. It means users can give up the changes they have made to the schedule. To "undo" the changes, the system needs to save the operations of the user. This can be done by keeping a table that is used to record the schedule information before any changes. The structure of the table is the same as the structure of the modification table for consistent purpose. A table with the name "Original" is added to the system.
- User can "fix" an operation to indicate that any changes on this operation are forbidden. This is quite useful when the operation is on line and some changes are necessary to be applied to the wait-in-line operations. To realize this function, schedule result table needs a field to save the user's intention. A field with name "FixOrNot" and data type "Boolean" or "Yes/No" is added to the schedule result table. The same structure change also applies to modification table and the original table.

**Based on the usability analysis, more extensions can be applied to the TDH:**



**Fig 4-4 TDH\_3**

The above modifications only apply to the schedule modification hierarchy. The following is some explanations of the new TDH:

- (9) and (10) are two choices for browsing the schedule data in text-based window.  
User can browse the data in sequence or go to any record selected by the user.
- Left-update changes the start date of an operation. Right-update changes the end date of an operation. Move changes both the start date and the end date while keeping the operation's span time.
- "Slot unchanged" indicates that the changes of the operation are kept in the same slot (e.g., the same machine center and work center). "Slot changed" indicates that the changes of the operation occur in different slots (e.g., different machine center or work center).

The new modification of TDH needs to be revised by analyzing the user's characteristics and cognitive conventions according to the new operations derived from the usability analysis.

- According to OH 4.1, any changes made in the graphic display need to be indicated. Users should always keep track with any changes they have made on the schedule. For example, if there is no status indicator for the change, the same operation may be changed twice or more by accident.
- Before undo operation, users need to be aware about the original position of the schedule. This can be done by leaving a track on the original position when any changes happen.

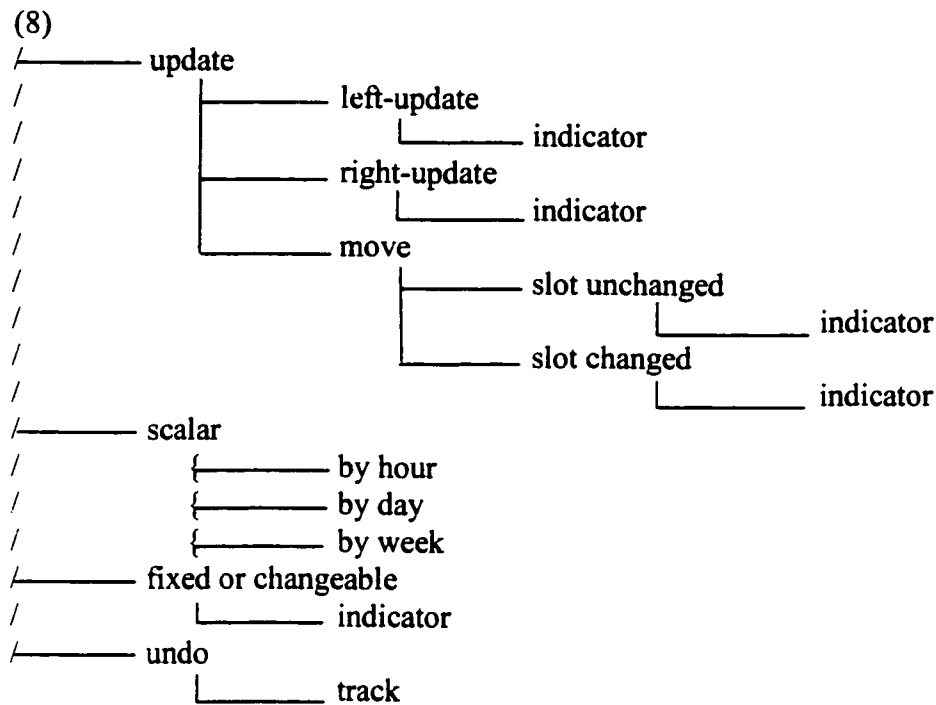
The new TDH for graphic modification is showed in Fig 4-5.

Again, user analysis is followed by the usability analysis.

- Since the user can decide the schedule span time, it is quite useful if the time scalar can be changed according to the length of the time. Otherwise, the user has to scroll the window from time to time.
- In order to keep the track of modification, the user has to know whether or not the operation has been changed. This needs an extra field in the modification table to save the knowledge for the user. The name of the new field is "ChangedOrNot" with the data type "Boolean". The same modifications also apply to the schedule result table and the original table.

The new modification is also showed in Fig 4-5.





**Fig 4-5 TDH\_4**

Until now, all of the functions of the user interface are quite clear to the designer.

The above analysis shows that:

- There are two cycles that exist in the user centered user interface design procedures (see Fig 3-11). The outer cycle is for the usability analysis, while the inner cycle is for the user analysis. The design process should continue until both of the conditions of the two cycles have been satisfied.
- TDH is always subject to change in the user interface design process. Since TDH is the handbook for the software implementation, TDH should be as complete as possible.
- There is no semantic or syntactic analysis included in the design procedure.

Designers can use their own language to describe each node of the TDH. But two

rules should be followed: (1) description should be detailed enough to represent the knowledge. (2) abbreviations do not have negative effects on the designers for them to comprehend the knowledge included in the description.

The above analysis supplies a complete instruction for the software implementation of the graphic facility. In the next section, the effect of the software implementation will be presented.

#### 4.4 Software Implementation

The platform selected for this facility is Windows NT/Windows 98. Visual Basic 5.0 is selected as the development tool. The local database is created by Microsoft Access 97. The name of the schedule software is "Schedule Master". Based on the framework shown in Fig 4-1, three tables are established first. "Schedule Result" table saves the knowledge of the basic system schedule information. "Modification" table saves the knowledge of the modification made by the users. "Original" table saves the schedule information before any changes have made. The structures of the three tables are identified. Table 4-2 shows the detailed structure.

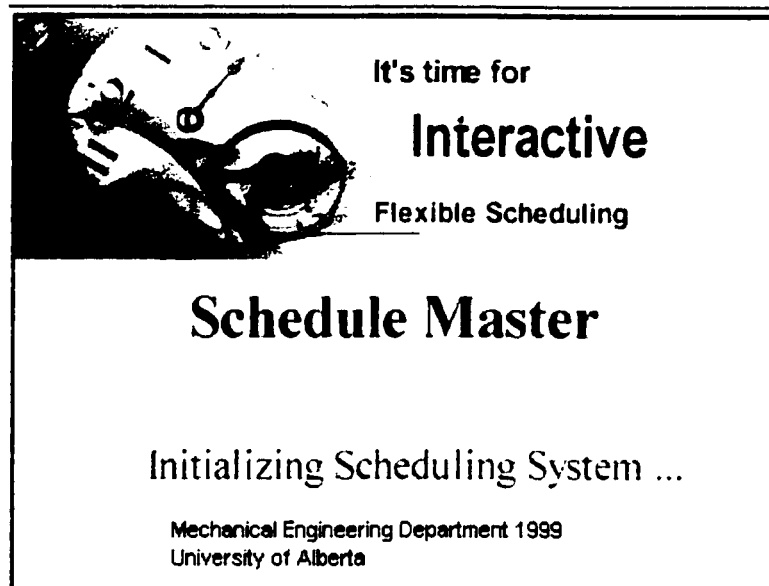
Field Name	Job ID	Operation ID	Work Center	Machine Center
Data Type	Text	Text	Text	Text

Field Name	Start Time	End Time	Start Date	End Date	Segment
Data Type	Date/Time	Date/Time	Date/Time	Date/Time	Text

Field Name	PColor	ChangedOrNot	FixOrNot
Data Type	Long Integer	Yes/No	Yes/Not

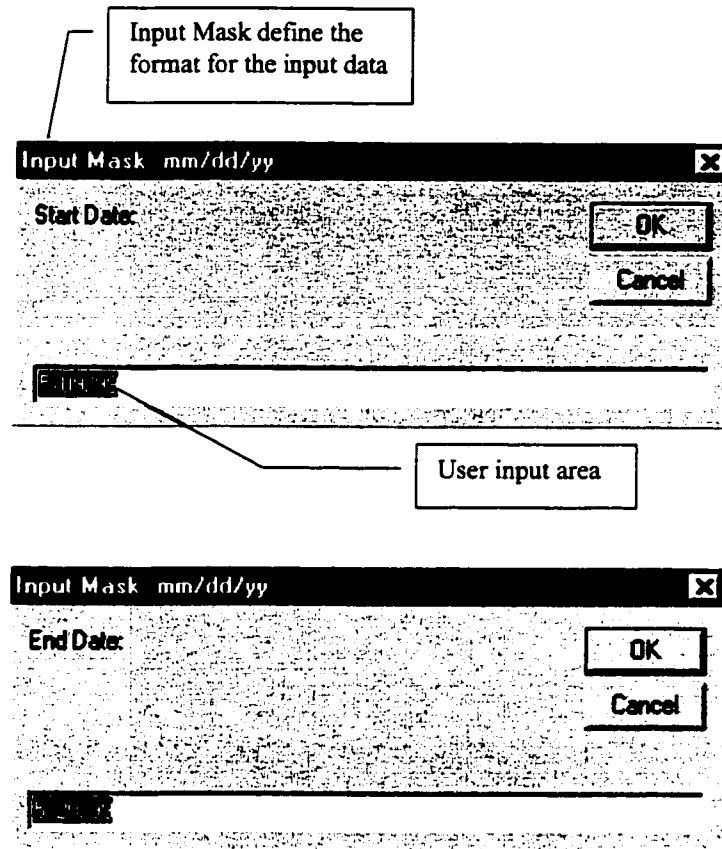
**Table 4-3 Data Structure of the Scheduling Knowledge**

When the program begins, the first screen displays the title of the software. Fig 4-6 shows the first window of Schedule Master.



**Fig 4-6 Beginning of Schedule Master**

The second and third screens are input boxes for the user to enter the schedule span time.



**Fig 4-7 Input Box of Schedule Master**

If "Cancel" button is pressed, the program will be ended by the confirmation of the user.

The next screen is the main window of Scheduler Master (Fig 4-8). The user needs to select "File|Open" from the menu to open a existing schedule database. After the database is selected, the job schedule saved in the open database will be displayed (see Fig 4-9). Besides the Gantt Chart, tool-tips text window which shows the text-based schedule information will be displayed when the mouse is pointed to the bar (see Fig 4-10). The left side of the screen shows the resource information in spreadsheet format. Time ruler locates on the top of the window.

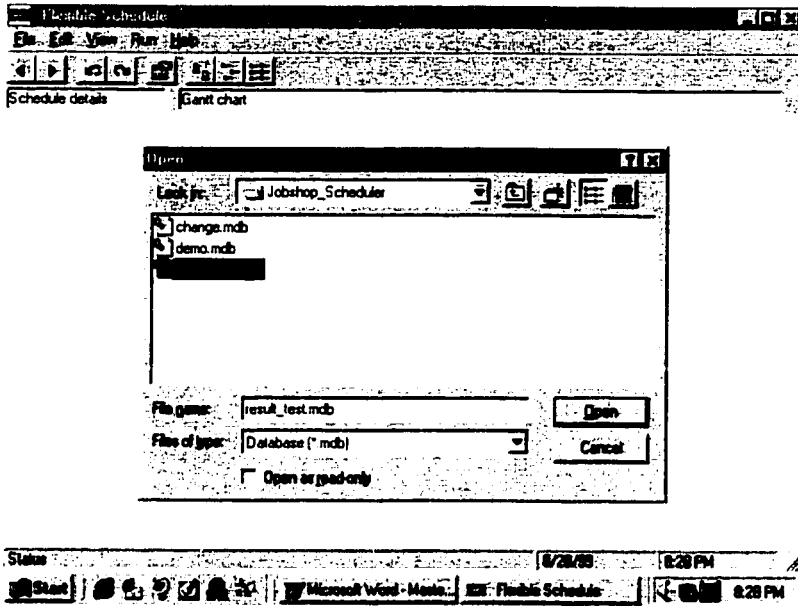


Fig 4-8 File|Open Screen

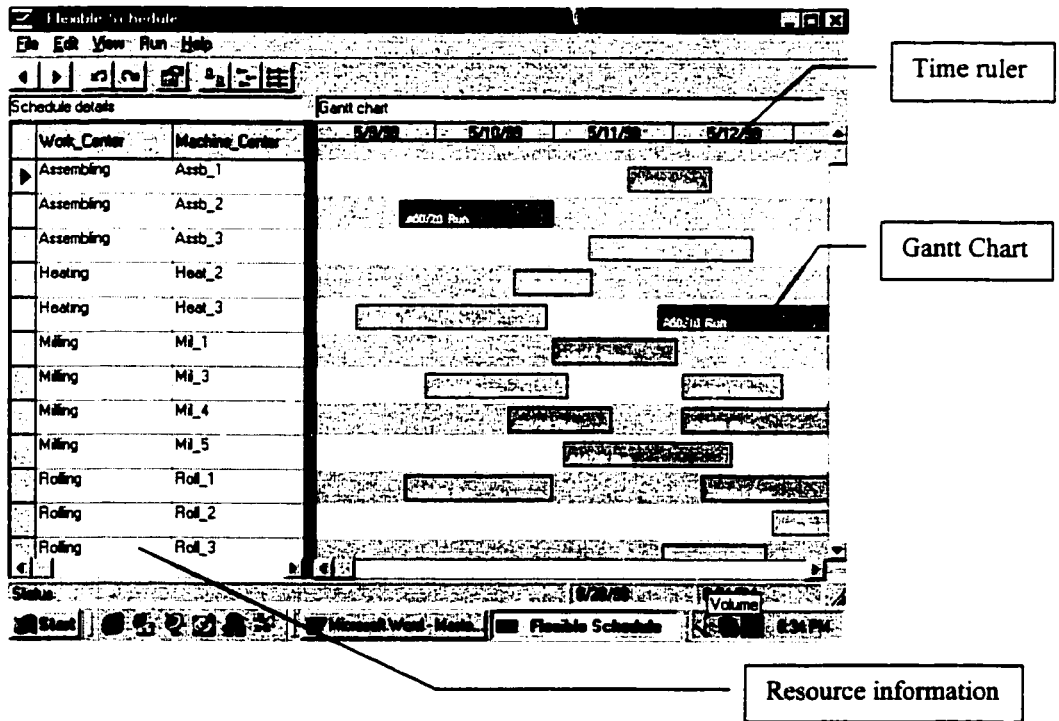
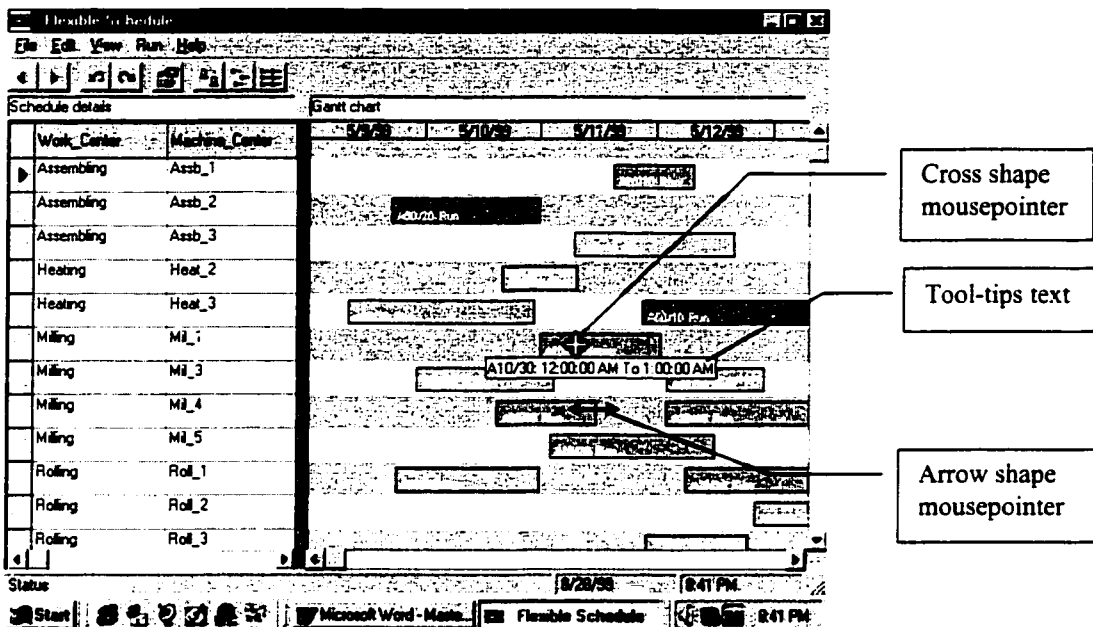
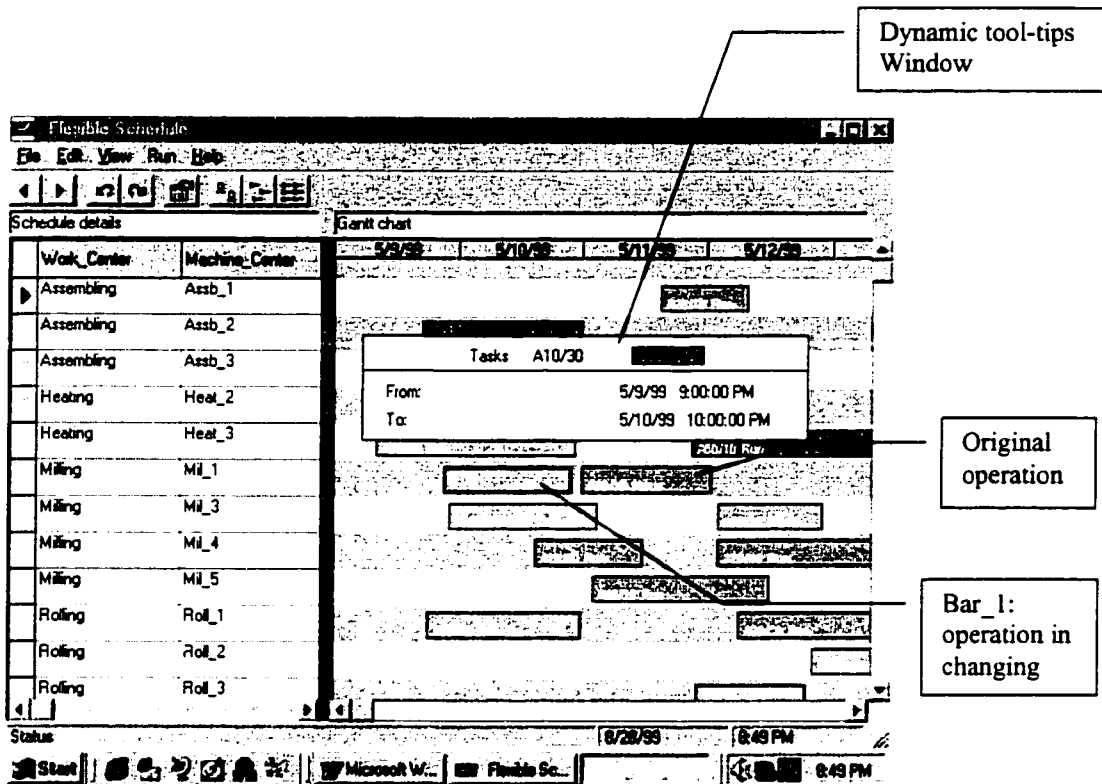


Fig 4-9 Main Window of Schedule Master (1)



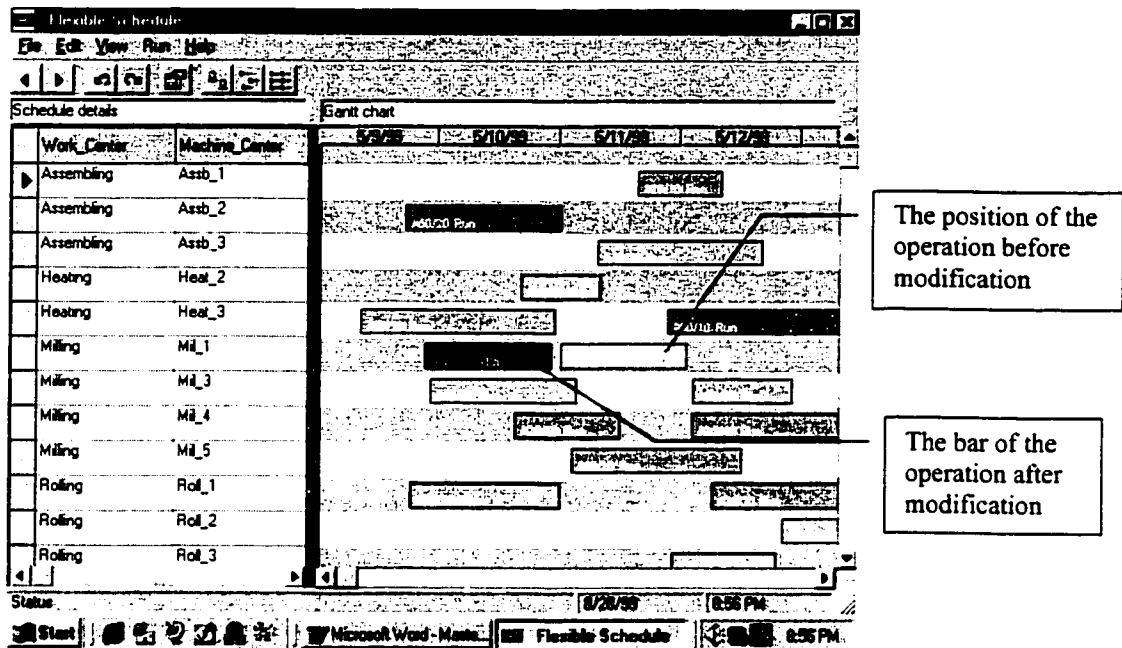
**Fig 4-10 Main Window of Schedule Master (2)**

The crossed shape mousepointer showed in Fig 4-10 indicates that the “move” action can be triggered if the left button of the mouse is pressed when moving the mouse. “move” action can change the start time and end time of the operation while keeping the span time unchanged. The arrow shape mousepointer showed in Fig 4-10 indicates that “left update” or “right update” action can be triggered if the left button of the mouse is pressed when moving the mouse. “left update” and “right update” actions change the start time and end time of the operation. The shape of the mousepointer depends on the relative position of the mouse on the bar. For example, when moving the mouse to the left side of the bar, the arrow mousepointer will appear and “left update” action is ready to be triggered.



**Fig 4-11 Main Window of Schedule Master (3)**

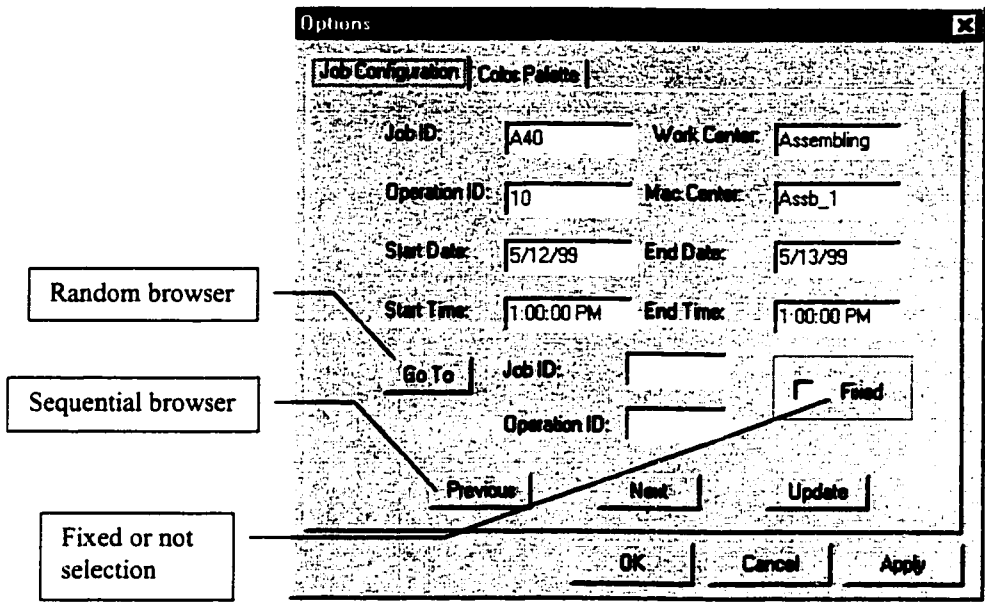
Fig 4-11 shows what the display looks like when “move” action is in process. Dynamic tool-tips window gives the real-time schedule information according to the position of the mouse. Bar\_1 indicates the position of the operation is in changing. Fig 4-12 shows what the display looks like after the modification. Changed operation leaves track in the position where is occupied by the same operation before any modifications are applied. The color of the modified operation is changed according to the operation’s original color. If the original colors of two operations are different, the colors of the two operations will still be different after modification.



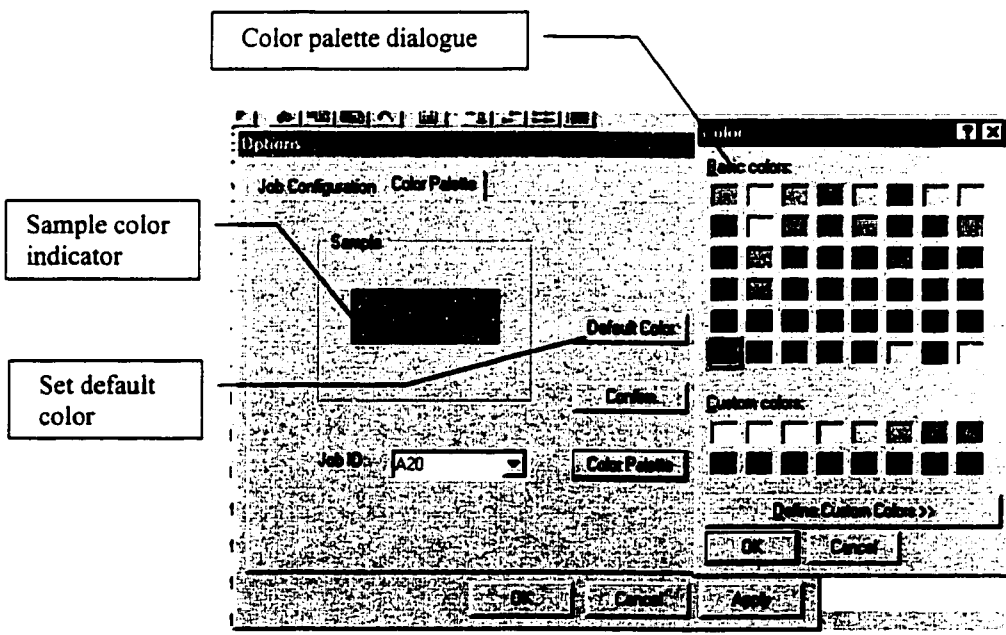
**Fig 4-12 Main Window of Schedule Master (4)**

In addition to the graphic display, users can change the colors of the Gantt Chart according to their own preference. Users can also select a text-based modification method for accurate time control. There are two ways to trigger the text-based editor and color configuration page: (1) clicking the right button when mouse is pointed to the bar, selecting “Configuration” from the popup menu. (2) selecting “View|Configuration” from the main menu. Fig 4-13 and Fig 4-14 show the appearance of text-based editor and color configuration page. The configuration window likes a notebook that has tags (dividers). Pressing the tag can bring the corresponding page. There are two pages in this window. One is text-based scheduler editor (see Fig 4-13), the other is the color configuration page (see Fig 4-14).





**Fig 4-13 Text-Based Schedule Editor**



**Fig 4-14 Color Configuration Page**

There are two ways to browse the record in text-based editor, sequential method and random method. For example, pressing “Next” button if user wants to go to the next record. The user can retrieve a specific record by pressing the “go to” button after assigning the Job ID and Operation ID fields.

Color configuration page is designed for two functions: (1) set default color for the Gantt Chart, (2) choose user’s preferred color for each job. The sample color bar let the user preview the effect before making the decision. Color palette dialogue gives the user flexible choice for color selection. In case of wrong selection, color changes will be ignored if the user does not press the “Confirm” button.

Now, the user can call the scheduling mechanism in the background by selecting “Run|Start” from the main menu when satisfied with the modification. As long as the background scheduling system begins, the records saved in the modification table and original table will be deleted. At this point, users can not “undo” the modification they have made. New feasible schedule will be created with the effort to fit the modifications. However, some modifications are subject to be changed by the scheduling system since the changes made by users may violate the internal constrains of the scheduling system.

#### **4.5 Conclusion**

The aforementioned prototype user interface which is developed by the proposed user-centered implementation heuristic (UCIH) has the following features:

- The prototype user interface is independent from the scheduling algorithms used by the scheduling system in the background. This is realized by separating the adaptive user interface module from the consistent interface module and keeping data structure of the two modules identified.
- Because of the existence of modification table (A/C interface module), both the adaptability and consistency characteristics are guaranteed. User's requests can be easily translated into consistent internal commands since the A/C interface module has the same data structure as the consistent module (schedule result table). And the scheduling result can be presented according to the user's preference.
- User-centered philosophy is implemented by analyzing the user's cognitive preference.
- Users can easily understand the operation prompt supplied by the system since no semantic symbols are used in system analysis process.
- Graphical displays improve the operation performance because data-driven actions are limited in graphic environment.

## **5. Conclusion**

### **5.1 Summary of the Research**

In this thesis, a user-centered implementation heuristic is proposed for user friendly user interface design. The case study gives an implementation sample based on the proposed user interface design heuristic. Benefits provided by this proposed heuristics are:

- Both the task characteristics and user characteristics are thoroughly considered during the user interface design process to guarantee the consistent and adaptive properties of the user interface.
- TDH tree describes complete functions of the interface with easy-to-understand words. No symbols or grammars are needed for constructing the TDH tree. This means no training program is needed for the designers to learn the method.
- Semantic/syntactic analysis procedure is avoided since the knowledge representation of the user or system may be distorted during the linguistic analysis process. This also eliminates the barriers that impede the users from understanding the internal commands of the system since the prompts given by the system are expressed by meaningful word or sentence instead of symbols or abbreviations.
- Circulating structure (see Fig 3-11) prevents incomplete system function analysis. The outer loop, which guarantees usability of the user interface, is the complement of the inner loop, which ensures the adaptability of the user interface, and vice versa.

- User-centered philosophy is implemented by the research on user characteristics and cognitive preferences.

However, implementing this heuristic in a user interface is quite time consuming since there are two loops in the analysis process. In most situations, the time that is spent on the initial design procedure is worthy because little needs to be changed when developing the software.

## **5.2 Recommendation for the Future Research**

The purpose of the proposed heuristic is to create a user-friendly interface. Based on the research in this field, one of the important features for the user-friendly interface is the ability to automatically adapt to the end user according to the user's preference. This object has not been realized because of the limit of today's technology. Researches in artificial intelligence (AI) field supply some methods for this problem, such as neural networking. Future research will be considered including some of the AI methods into the heuristics to improve the adaptive performance of the user interface.

Another problem is how to evaluate the usability, flexibility, consistency and adaptability of a user interface. In other words, it is necessary to find a way to create a set of criteria to judge whether an interface is user-friendly. The difficulty of evaluation lies on two aspects. Firstly, many factors have been involved in the design procedure. Some steps, such as the cognitive-based analysis, include purely subjective parameters. As a result, objective measurement is difficult to set. Secondly, users always have their own

criteria on “beauty”. It is hard to find a uniform and well-accepted “perfect” model.

Further research on this problem may begin by finding a way to separate the objective parameters and the subjective parameters so that corresponding measuring instrument or procedure can be established.

## Bibliography

1. Alter, S. L. 1980, *Decision support systems: Current practice and continuing challenges*, Addison-Wesley, Reading MA.
2. Andriole, S. and Adelman, L., 1995, *Cognitive Systems Engineering for User-Computer Interface Design, Prototyping, and Evaluation*, LEA Publishers, NJ.
3. Baker, K. R., 1974, *Introduction to Sequencing and Scheduling*, Wiley, New York.
4. Barnard, P. J., et al., 1981, *Consistency and Compatibility in the Human-Computer Dialogue*, International Journal of Man-Machine Studies, No. 15, p87-134.
5. Bennett, J. L., 1983, *Building Decision Support Systems*, Addison-Wesley.
6. Blanning, R. W., 1979, *The Functions of a Decision Support System*, Information and Management, 2(3).
7. Bonczek, R. H., C. W. Holsapple, and A. B. Whinston, 1981, *Foundations of decision support systems*, Academic Press, New York.
8. Chung, W. C., Tam, M. C., Saxena, K. B. C. and Yung, K. L., 1993, *Evaluation of DSS use in Hong Kong Manufacturing Industries*, Computers in Industry, Vol. 21, p307-324.
9. Collinot, A., Pape, C. L. and Pinoteau, G., 1988, *SONIA: A Knowledge-based Scheduling System*, Artificial Intelligence in Engineering, Vol. 3, No. 2, p85-94.
10. Conway, R. W., Maxell, W. L., and Miller, L. W., 1967, *Theory of Scheduling*, Addison-Wesley, Reading MA.
11. Cudd, P. A., Oskouie, R., 1996, *Combining HCI Techniques for Better User Interfacing*, IEE-Colloquium-(Digest), No. 126, p11/1-11/8.

12. Diaper, D., 1989, *Task Analysis for Human-Computer Interaction*, Ellis Horwood Limited, Hartnolls, U.K.
13. Duce, D. A., et al., 1991, *User Interface Management and Design*, Springer, NY.
14. Dudek, R. A., Panwalkar, S. S., and Smith, M. L., 1992, *The Lessons of Flowshop Scheduling Research*, Operations Research, Vol. 40, No. 1, pp 7-13.
15. Eason, K. D., 1984, *Toward the Experimental Study of Usability*, Behavior and Information Technology, Vol. 3, No. 2, p133-143.
16. Edmonds, E., 1978, *Adaptable Man-Machine Interfaces for Complex Dialogues*, Proceedings of the European Computing Congress, London.
17. Farber, J. M., *The AT&T User Interface Architecture*, AT&T Technical Journal, 68(5), p9-16.
18. Ferguson, R. L. and Jones, C. H., 1969, *A Computer Aided Decision System*, Management Science, 15, p550-561.
19. French, S., 1982, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood Ltd., New York.
20. Giroux, L. and Larochelle, S., 1987, *Learning Information of Cognitive Systems: Questions and Research*, University of Montreal and Canadian workplace, Automation Research Center, p154.
21. Gopalakrishnan, K. N., 1998, *DSS for Scheduling Steel Fabrication Projects*, Ph.D. Thesis, Department of Mechanical Engineering, University of Alberta, Edmonton, AB.
22. Graves, S. C., 1981, *A Review of Production Scheduling*, Operations Research, 29(4), pp 646-675.



23. Harrison, M. and Thimbleby, H., editor, 1990, *Formal Methods in Human-Computer Interaction*, Cambridge University Press, NY.
24. Hastings, N. A. J., Marshall, P. H., and Willis, R. J., 1982, *Scheduled based M.R.P.: An integrated approach to production scheduling and material requirements planning*, Journal of the operational Research Society, 33(11), pp 1021-1029.
25. Hastings, N. A. J., Yeh, C.-H., 1990, *Job oriented production scheduling*, European Journal of Operational Research, Vol. 47, pp 35-48.
26. Hix, D. and Hartson, H. R., 1993, *Developing User Interface – Ensuring Usability Through Product & Process*, John Wiley & Sons, NY.
27. Holsapple, C. W. and Winston, A. B., 1996, *Decision Support System: A Knowledge-based Approach*, West Publishing, MN.
28. Johnson, S. M., 1954, *Optimal Two- and Three- Stage Production Schedules with Setup Times Included*, Naval Research Logistics Quarterly, Vol. 1, No. 1.
29. Jones, C. V., 1990, *An Introduction to Graph-Based Modeling Systems*, Spring Press, NY.
30. Kellog, W. A., 1987, *Conceptual Consistency in the User Interface: Effects on User Performance*, Proc. INTER-ACT' 87 Conference on Human-Computer Interaction, p1-4.
31. Kim, S. H. and Lee, K. K., 1997, *An Optimization-based Decision Support System for Ship Scheduling*, Computer Industry Engineering, Vol. 33, p689-692.
32. Lapointe, F., 1991, *User-Friendliness of Interfaces: Proposal for an Evaluation Methodology*, Canadian Workplace Automation Research Center.

33. Liang, T. P., 1987, *User Interface Design for Decision Support Systems: A Self-Adaptive Approach*, Information & Management, Vol. 12, p181-193.
34. Lipske, K. R., 1996, *A Greedy-based Decision Support System for Scheduling A Manufacturing Operation*, Production and Inventory Management Journal, First Quarter, p36-39.
35. Lustman, F., Mercier, P. and Gratton, L., 1985, *A Dialogue-based Architecture for Interactive Information Systems*, Data Base, Spring, p18-24.
36. Moran, T. P., 1983, *An Applied Psychology of the User*, Computing Survey, Vol. 13, No. 1, p1-11.
37. Morton, S., 1971, *Management decision systems: Computer-based support for decision making*, Cambridge, Mass.: Division of Research, Harvard University.
38. Mulvehill, A. M., 1988, *A User Interface for a Knowledge-based Planning and Scheduling System*, IEEE Transactions on System, Man, and Cybernetics, Vol. 18, No. 4, p514-602.
39. Newman, W. A., 1967, *A Graphical Technique for Numerical Input*, Computer Journal, Vol. 11, p63-64.
40. Newman, W. A., 1968, *A System for Interactive Graphical Programming*, Spring Joint Computer Conference, AFIPS Press, p47-54.
41. Newman, W. A., 1978, *Personalized User Interface to Computer Systems*, Proceedings of the European Computing Congress, London.
42. Nielsen, J., 1989, *Coordinating User Interfaces for Consistency*, Academic Press Inc., San Diego.

43. Nkasu, M. M. and Leung, K. H., 1997, *A Resources Scheduling Decision Support System for Concurrent Project Management*, International Journal of Production Research, Vol. 35, No. 11, p3107-3132.
44. Norman, D. A., 1983, *Design Principles for Human-Computer Interfaces*, Proc. CHI'83, Human Factors in Computing Systems, Boston, ACM, NY, p1-10.
45. Norman, D. A., 1986, *User Centered System Design*, Cognitive Engineering, Hillsdale, NJ: Lawrence Erlbaum Associates.
46. Panwalkar, S. S., Iskander, W., 1977, *A Survey of Scheduling Rules*, Operations Research, Vol. 25, No. 1, pp 45-61.
47. Park, H. G., et al., 1998, *An Object Oriented Production Planning System Development in ERP Environment*, Computers and Industrial Engineering, Vol. 35, p157-160.
48. Payne, J. W., Bettman, J. R., and Johnson, E. J., 1987, *Adaptive Strategy in Decision Making*, Durham, Duke University.
49. Pew, R. W., et al., 1983, *Research Needs for Human Factors*, National Academy Press, Washington.
50. Pfaff, G. E., 1985, *User Interface Management Systems*, Sring-Verlag.
51. Pinedo, M., 1995, *Scheduling Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs.
52. Powals, J. G., et al., 1995, *Cognitive Redesign of Submarine Displays*, Cognitive Systems Engineering for User-Computer Interface Design, Prototyping, and Evaluation, edited by Andriole, S. and Adelman, L., LEA Publishers, New Jersey.

53. Reisman, A., Kumar, A., and Motwani, J., 1997, *Flowshop Scheduling/Sequencing Research: A Statistical Review of the Literature, 1952-1994*, IEEE Transactions on Engineering Management, Vol. 44, No. 3.
54. Sankar, C. S., Ford, F. N. and Bauer, M., 1995, *A DSS User Interface Model to Provide Consistency and Adaptability*, Decision Support Systems, Vol. 13, p93-104.
55. Santos, B. L. and Bariff, M. L., 1988, *A Study of User Interface Aids for Model-Oriented Decision Support System*, Management Science, 34(4), p461-468.
56. Santos, B. L. and Holsappie, C. W., 1989, *A Framework for Designing Adaptive DSS Interface*, Decision Support System, 5(1), p1-11.
57. Schlungbaum, E., 1997, *Individual User Interfaces and Model-based User Interface Software Tools*, International Conference on Intelligent User Interfaces, Proceedings-IUI, ACM, p229-232.
58. Schniederjans, M. J. and Carpenter, D. A., 1996, *A Heuristic Job Scheduling Decision Support System: A Case Study*, Decision Support Systems, Vol. 18, p159-166.
59. Sprague, R. H. and Carlson, E. C., 1982, *Building Effective Decision Support Systems*, Prentice Hall, NJ.
60. Sprague, R. H. and Watson, H. J., editors, 1986, *Decision Support System: Putting Theory into Practice*, Prentice-Hall, NJ.
61. Stary, C. and Peschl, M. F., 1998, *Representation Still Matters: Cognitive Engineering and User Interface Design*, Behavior and Information Technology, Vol. 17, No. 6, p338-360.
62. Stevens, G. C., 1983, *User Friendly Computer Systems? A Critical Examination of the Concept*, Behavior and Information Technology, Vol. 2, No. 1, p3-16.

63. Storrs, G., 1989, *Toward a Theory of HCI: A Conceptual Model of Human-Computer Interaction*, Behavior and Information Technology, Vol. 8, No. 5, p323-334.
64. Tyler, S., Treu, S., 1989, *An Interface Architecture to Provide Adaptive Task-Specific Context to the User*, International Journal of Man-Machine Studies, No. 30, p303-327.
65. Viviers, F., 1983, *A Decision Support System for Job Shop Scheduling*, European Journal of Operational Research, V14, p95-103.
66. Webster, J., 1990, *The Relationship Between Playfulness of Computer Interactions and Employee Productivity*, Desktop Information Technology, Edited by Kaister and Oppelland, Elsevier Science Publisher.
67. White, C., Hastings, N. A. J., 1983, *Scheduling techniques for medium scale industry*, Australian Society for operations Research Bulletin 3, 1-4.
68. Wong, B. K. and Monaco, J. A., 1995, *Expert System Applications in Business: A Review and Analysis of the Literature (1977-1993)*, Information and Management, Vol. 29, p141-152.
69. Woods, D. D. and Roth, E. M., 1988, *Cognitive Engineering: Human Problem Solving with Tools*, Human Factors, Vol. 30, p415-430.
70. Yeh, C. H., 1997, *Fast finite loading algorithm for job oriented scheduling*, Computers and Operations Research, Vol. 24, No. 2, p193-198.
71. Young, R. M., 1981, *The Machine Inside the Machine: User's Models of Pocket Calculators*, International Journal of Man-Machines Studies, No. 15, p51-85.

## **Appendix: Cognitive Findings and its Application Rules**

Cognitive findings are expressed by a set of conceptual hypotheses (CH).

Operational hypothesis (OH) is derived from the corresponding conceptual hypothesis.

One cognitive finding may suggest more than one conceptual hypothesis, which then indicates more than one operational hypothesis. The following is the conceptual and operational hypotheses derived from the cognitive findings (Powals, 1995).

### **1. Situational Awareness**

**Conceptual Hypothesis (CH) 1:** if situational awareness can be improved by controlling demands on attentional resources, user-computer performance will be enhanced.

**Operational Hypothesis (OH) 1.1:** The automation of unwanted workload will result in better user-computer performance than when unwanted workload is not automated.

**OH 1.2:** The use of alert messages and color coding to reduce uncertainty will result in better user-computer performance than when uncertainty is not reduced by alert messages and color coding.

**OH 1.3:** Using appropriate data grouping to fuse data will result in better user-computing performance than when data are not appropriately grouped.

### **2. Schematic Incorporation of New Information**

**CH 2:** If new information is presented together with meaningful aids to interpretation, user-computer performance will be enhanced.

**OH 2.1:** Presenting new information together with familiar metaphors such as miniature subs and conventional use of “red” and “green” will result in better user-

computer performance than when new information is not presented along with familiar metaphor.

### 3. Context-Dependent Reconstructive Retrieval

CH 3: If display names and labels provide a meaningful context for recall and recognition, user-computer performance will be enhanced.

OH 3.1 Display names and labels that are conceptually related to function will result in better user-computer performance than when display names and labels are not conceptually related to function.

OH 3.2: Data that are grouped in consistently meaningful ways on the screen will result in better user-computer performance than when data are not grouped in consistently meaningful ways.

### 4. Top-Down and Bottom-Up Processing

CH 4: By limiting data-driven tasks, user-computer performance will be enhanced.

OH 4.1: Using status indicators to limit data-driven tasks will result in better user-computer performance than when status indicators are not used to limit data-driven tasks.

CH 5: If informational load is minimized in order to allow human processing to comfortably absorb all displayed data, user-computer performance will be enhanced.

OH 5.1: Displays that include only that information needed by the operator at a given time will result in better user-computer performance than displays that include additional information.

OH 5.2: An interface with judicious redundancy will result in better user-computer performance than an interface with unnecessary redundancy.

## 5. Limited Processing Ability

CH 6: Displays that promote cognitive flexibility will enhance user-computer performance.

OH 7.1: Multiple coding of data will result in better user-computer performance than when data are only in one form.

## 6. Cognitive Load

CH 8: If cognitive load is decreased, learning time, fatigue, stress, and proneness to error will be decreased, thus user-computer performance will be enhanced.

This conceptual hypothesis will be operationalized in the same manner that was used for conceptual hypothesis 4.

## 7. Interference

CH 9: By reducing the cognitive load associated with each task performance and by improving the quality of data available to the user, the likelihood of interference occurring will be reduced and user-computer performance will be enhanced.

This conceptual hypothesis will be operationalized in the same manner that was used for conceptual hypothesis 4.

## 8. Problem Solving

CH 10: If the user can easily comprehend the situation using a minimal of valuable cognitive resources, functional problems will be mitigated, and user-computer performance will be enhanced.

This conceptual hypothesis will be operationalized in the same manner that was used for conceptual hypothesis 1 and the same method that was used in conceptual hypothesis 4.



## 9. Simplicity

CH 11: If a system is simple, it is easier to learn and easier to use, therefore, user-computer performance will be enhanced.

With the exception of the operational hypotheses for conceptual hypothesis 7, all of the other operational hypotheses contribute to making the interface more simple and therefore would operationalize conceptual hypothesis 11 as well.