THESIS REPORT

Master's Degree

A Family of User Interface Consistency Checking Tools

by R. Mahajan
Advisor: B. Shneiderman

M.S. 96 -5

INSTITUTE FOR SYSTEMS RESEARCH

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Abstract

Title of Thesis: A Family of User Interface Consistency Checking Tools

Name of degree candidate: Rohit Mahajan

Degree and year: Master of Science, 1996

Thesis directed by: Ben Shneiderman
Systems Engineering & Institute of Systems Research

Designing a user interface with a consistent visual design and textual properties with current generation GUI development tools is cumbersome. *SHERLOCK*, a family of consistency checking tools, has been designed to evaluate visual design and textual properties of interface, make the GUI evaluation process less arduous, and aid usability testing. *SHERLOCK* includes a dialog box summary table to provide a compact overview of visual properties of hundreds of dialog boxes of the interface. Terminology specific tools, like Interface Concordance, Terminology Baskets and Interface Speller have been developed. Button specific tools including Button Concordance and Button Layout Table have been created to detect variant capitalization, distinct typefaces, distinct colors, variant button sizes and inconsistent button placements. This thesis describes software architecture, data structures and the use of *SHERLOCK*. An experiment with 60 subjects to study the effects of inconsistent interface terminology on user's performance showed 10-25% speedup for consistent interfaces. *SHERLOCK* was tested with four commercial prototypes; the corresponding outputs, analysis and feedback from designers of these applications is presented.

A Family of User Interface Consistency Checking Tools

by

Rohit Mahajan

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1996

Advisory Committee:

Ben Shneiderman, Chairman/Advisor
Catherine Plaisant & Mark Austin

Acknowledgements

I am thankful to Prof. Ben Shneiderman for supporting me throughout my stay at HCIL. Due to his support, I received lot of recognition in the software industry. Further he helped me to improve my communication and presentation skills. I am amazed by his expertise in the field of Human-Computer Interaction (HCI) and other areas beyond this discipline.

Dr. Catherine Plaisant at HCIL has always been there to help me, whether related to design of *SHERLOCK* tools or finding previous research papers. Her extensive knowledge in the discipline of HCI helped me throughout my research.

Prof. Mark Austin who guided me as an instructor in the Systems Design class is so helpful that I could knock his door anytime for help. His excellence in the field of Civil Engineering, Systems Engineering and Computer Science helped me to improve my thesis in the multi-disciplinary Systems Engineering curriculum.

I appreciate the support for this project from GE Information Services and the Maryland Industrial Partnership Program. I am thankful to Dr. Ren Stimart and David White at the GE Information Services for their help and support in every part of the project which includes providing test applications and giving feedback at every step.

Prof. Victor Basili's expertise in Software Engineering helped me in developing and modifying the *SHERLOCK* tools. I am thankful to him for sharing his software metrics expertise with me and Prof. Shneiderman. I would also like to thank Gianluigi Caldiera, who helped me in the design of *SHERLOCK* with his experience and knowledge in software engineering. Prof. Adam Porter

with his experience in software development gave me feedback on improving *SHERLOCK*.

Anna Giannetti, Head of Software Usability at Sogei Inc. in Italy has been so helpful and kind to me in providing test application for *SHERLOCK* and giving me suggestions at every step of the development cycle.

I am thankful to Marshall McClintock, director of Usability at Microsoft Corp. for inviting me to Seattle to present my work on *SHERLOCK* and share ideas and get feedback from the diverse intellect of developers, designers and managers at Microsoft. I would also like to thank Timothy Loungeway at the American Management Systems (AMS) for inviting me to present my work on *SHERLOCK* at the AMS, AMSCAT day. I am thankful to Chris Dryer, President Society of Software Quality for inviting me to present my work on *SHERLOCK* at the "Visual Programming Roundtable".

I am thankful to everyone at the AT&T teaching theater at the University of Maryland, especially Dr. Walt Gilbert and C.S. Chang for providing me with test applications for *SHERLOCK*. I also would like to thank Prof. Maryam Alavi and Mun Yi at the University of Maryland, College of Business and Management for also providing me with test applications.

Fellow members at HCIL, have contributed significantly to this research through their constructive suggestions. Richard Chimera and Ninad Jog contributed significantly in development of *SHERLOCK* by desigining the canonical format, constructing the initial metric set and implementing portions of *SHERLOCK* tools. I am so thankful to Stephan Greene for reading my dissertation in detail and providing thoughtful comments to improve the presentation of my work. Anne Rose, the HCIL, Lab Manager and my colleague Esser Kandogan

helped me whenever I had questions on nitty griddy of UNIX, C++ programming language or even making presentation slides. Everyone at HCIL, both undergraduates and graduate students helped me in the development of *SHERLOCK*. Also, I can never forget Ara Kotchian, who helped me in dealing with any network or computer hardware problems. We at HCIL worked as a family in helping each other on different projects.

My family (father, mother and brother) have always been the main source of my inspiration and encouragement. Given the amount of sacrifice they have made, especially by my brother, they deserve the most credit for my work. My friend Giles Charleston and Adil Rajput always helped me see the brighter side of things and were there whenever I needed them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

It is a rare user interface designer that has not had a vision of an interface that is easy to learn, uses the same interaction concepts and techniques throughout many different applications and programs, and is natural once mastered. This vision of a consistent user interface is not just a dream; many organizations are indeed developing and marketing consistent user interfaces. These consistent interfaces were achieved with some effort; it takes more than just a desire to create a consistent interface. The interface design must promote consistency; appropriate tools and methodology must support that design. In addition, political support for consistent design is critical. Practices to support consistency are also mandatory, for consistent designs require a new way of development. Clearly, consistency does not just happen; a strategy or plan for realizing a consistent interface is required.

Ianne H. Koritzinsky [22]

The current generation software development is inclined towards interactive computer systems design. Design of these interactive systems involves integration of two different and difficult disciplines, Psychology and Computer Science, into "Human-Computer Interaction" known as HCI, thus making GUI (Graphical User Interface) design a complex and challenging discipline. The user interface design processes have sped up remarkably in the last couple of years as pow-

erful development tools and user interface management systems have emerged. The impact of these powerful development tools has been so massive that only a few weeks are needed to create a test application. These interfaces may embody inconsistencies in visual design and textual properties which can't be detected by these powerful current generation development tools. Such inconsistencies can have a subtle and negative impact on the usability of the interface. Not only are more quality control and GUI test procedures required, but also new analytic and metric based tools are needed for the creation of cognitively consistent interfaces having a common "look and feel".

1.1 Contributions

The major contributions of this work are as follows:

- Consistency Checking Tools: Developed *SHERLOCK*, a family of consistency checking tools to evaluate visual design and textual properties of a user interfaces.
- Scientific Validation: Conducted a scientific experiment to study the effects of GUI terminology inconsistencies on user's performance and subjective satisfaction.

1.2 Content

Chapter 2 presents the original motivation for this work and a review of the relevant previous literature. Chapter 3 describes the *SHERLOCK* tools including the type of inconsistencies evaluated by each tool, outputs of each tool and any

specific numerical values linked with the tool metrics. Chapter 4 is an overview of the software system architecture and data structures used for developing *SHERLOCK*. Chapter 5 reports on the scientific experiment conducted to study the effects of GUI terminology inconsistencies on user's performance and subjective satisfaction. Chapter 6 discusses the evaluation outputs of the four applications analyzed by *SHERLOCK*. Chapter 7 describes the feedback given by the designers on *SHERLOCK* evaluations. Chapter 8 summarizes and suggests directions for future work.

Chapter 2

Analysis

Much of the success of Macintosh can be traced to its consistent look and feel. Some of that consistency can be traced to the simple mechanics of a user interface toolkit and a guidelines book. But consistency was an idea embraced by the entire Macintosh community- Apple Computer itself, our developers, our dealers, our users, and the reviewing press. Without everyone's support, consistency would not have been achieved.

Bruce Tognazzini [42]

This chapter explains the motivation for developing *SHERLOCK*, specifies the system requirements, and analyzes the related work.

2.1 Motivation: A Generic User Interface Evaluation Tool

The motivation for this work was our research on developing a metrics based tool to evaluate user interfaces. We worked closely with General Electric Information Services to design and develop task-independent metrics to evaluate visual design and textual properties of user interfaces. We developed a single tool with

multiple metrics to accomplish this task. After analyzing a few interfaces with this tool, we found that a single metric table was insufficient to encompass all the different kinds of visual design inconsistencies and was too cumbersome to interpret inconsistencies. Also, as more metrics were added, the output of the tool expanded beyond the limit of a compact overview, with too many values to be compared and the output too big to fit all the metrics columns on a single page.

2.2 Evolution of *SHERLOCK*

The complexity of interpreting the results of a single evaluation tool led us to modify our evaluation approach by constructing smaller tools, each of which evaluates a few design aspects. Together these tools evaluated multiple design issues, forming a family of consistency checking tools. The advantage of this new approach is that some of these tools perform exception reporting by outputting the possible anomalies in visual design and textual properties of the interface. The reports generated by these mini tools require less interpretation, thereby expediting the quick evaluation process and providing feedback to the designer. The designer must then decide whether the inconsistencies detected are relevant to the particular application.

2.3 System Requirements

SHERLOCK needs to meet the following system requirements.

Requirement 1: Tools of *SHERLOCK* should be generic. That is, not designed for any particular interface development tool.

multiple metrics to accomplish this task. After analyzing a few interfaces with this tool, we found that a single metric table was insufficient to encompass all the different kinds of visual design inconsistencies and was too cumbersome to interpret inconsistencies. Also, as more metrics were added, the output of the tool expanded beyond the limit of a compact overview, with too many values to be compared and the output too big to fit all the metrics columns on a single page.

2.2 Evolution of *SHERLOCK*

The complexity of interpreting the results of a single evaluation tool led us to modify our evaluation approach by constructing smaller tools, each of which evaluates a few design aspects. Together these tools evaluated multiple design issues, forming a family of consistency checking tools. The advantage of this new approach is that some of these tools perform exception reporting by outputting the possible anomalies in visual design and textual properties of the interface. The reports generated by these mini tools require less interpretation, thereby expediting the quick evaluation process and providing feedback to the designer. The designer must then decide whether the inconsistencies detected are relevant to the particular application.

2.3 System Requirements

SHERLOCK needs to meet the following system requirements.

Requirement 1: Tools of *SHERLOCK* should be generic. That is, not designed for any particular interface development tool.

Requirement 2: *SHERLOCK* should be able to analyze aspects of graphical user interfaces (developed on any tool or platform) as long as the interface description or resource files are converted into a canonical format (described in Chapter 3)

Requirement 3: Every GUI development tool would require a different translator to convert the interface description or resource files to a canonical format. The Translator tools are independent of *SHERLOCK* and can be written in any programming language and may be developed on any platform.

Requirement 4: The *SHERLOCK* documentation needs to answer these basic questions:

- What numerical values of the evaluation metrics indicate the possibility of bad screen design ?
- What does the output of each tool imply?
- What are the different kinds of inconsistencies that each tool can detect?

2.4 Literature Review

Research related to user interface design and tools to evaluate visual design and terminology in user interfaces was studied in order to create the best design for *SHERLOCK* and is discussed in this section.

2.4.1 Human Factors in GUI design

Designing a user interface is a complex process involving iterative design, usability testing and evaluation [38]. User interface design is influenced by three

factors [15], namely:

- The constraints that implicitly or explicitly predefine much of the user interface
- The different phases of product development during which the design evolves
- The planning needs for future development efforts of the managers above the project manager.

Interface system design should be user-centered [32], as the user is the first and most important player in the computer system game [50]. The problem with current systems development is that the software methods that are being used widely today were developed before GUI development came into the picture. Therefore designs are often not sufficiently user-centered. Thus, these software development methods create obstacles in user interface design.

Three categories of software system development are contract development, product development, and in-house development with customer development [13]. Each of these development types creates a "wall" between users and developers. According to Grudin [13], contract development relies on specification and imposes "walls" between users and developers, the market and customers in product development buffer developers from end-users and for the in-house development, the internal developers create documentation "walls" between themselves and the users.

GUI programming in recent years has become a major part of software development, and is minimally 29% of software development budgets [35]. Moreover, data analysis has shown that the user interface is 47-60% of the total lines of application code [27]. GUI design encompassing more than one third of the

software development cycle plays a major role in determining the quality of a product. Human factors research and the goal of usability are intended to enhance the quality of the product. Applying proper human factors techniques including early completion of user requirements definitions, usability prototype testing, and usability walkthroughs can significantly reduce the cost of identifying and resolving usability problems, and can save time in software development [24]. Earlier, Karat [23] showed that these human factors techniques can be tailored to fit the financial aspects and time schedules of software projects and thus remain cost-effective. Her analysis showed that inclusion of human factors work, including usability testing in a large software development project can give a 100:1 saving-to-cost ratio.

2.4.2 Consistency and Evaluation Techniques in Graphical User Interfaces

Consistency in User Interfaces follows the second law of thermodynamics. If nothing is done, then entropy will increase in the form of more and more inconsistency in your user interface

Jakob Nielsen [31].

Consistency in design is an important aspect to be considered when creating user interfaces and is supported by all user interface experts. Shneiderman [38] says that the first golden rule of dialog design is strive for consistency. This principle is the most frequently violated one, and yet the easiest one to repair and avoid (violating). Nielsen [31] says that one of the most important aspects of usability is consistency in user interfaces. Consistency should apply both

within an individual application, and across complete computer systems; and product families.

Definition of Consistency: Defining and achieving consistency in this rapidly changing world of Graphical User Interfaces is a difficult task. Experts have struggled to define exactly "what consistency is?" and "how to identify good consistency?". Reisner [34] states that consistency is neither a property of the system nor the user, it is a relation between two languages, that of the system, as a designer intended it, and of the competent user. Wolf [49] says that consistency means that similar user actions lead to similar results. A consistent user interface is an interface that maximizes the number of shared rules across tasks [33]. Consistency deals with meeting expectations of people and ensuring predictability across the system. Therefore, consistency is a relational concept and can't be defined by itself. A user interface is consistent or inconsistent with respect to something, which may be within the individual application or across a product family or for all the applications running on a particular system.

According to Blake [4] a consistent interface is the one with the following attributes within an application and across multiple applications.

- **Predictable:** Users anticipate what the system will do.
- **Dependable:** The system fulfills the user's expectations.
- **Habit-forming:** The system encourages development of behavior patterns
- **Transferable:** Habits developed in one context apply in new situations
- **Natural:** The interface is consistent with user's understanding.

Consistency across the application in those components of the user interface which require human perception and cognitive mechanisms like visual scanning, learning and remembering is very important. Spatial organization which may include the organization of menus, placement of frequently used widgets, symmetry, and alignment of widgets is one of those components. Other components may include fonts and colors used in the interface. Shneiderman [38] says that consistency refers to common actions, sequences, terms, units, layouts, colors, typography and more within an application program; it is naturally extended to include compatibility across application programs and compatibility with paper or non-computer-based system. Shneiderman and Chimera [6] demonstrated that consistency in color, terminology, layout, instructions, and so forth, do influence a user's perception, performance, and subjective satisfaction.

Display design with graphical objects has been studied for human performance by manipulating three visual features of polygon displays, namely background, shading and form [46]. The implications of this study are that shading figures may enhance consistency of performance and that human judgment of similarity is weighted towards distinctive, similar features of the figures. Also, the order in which variables are arranged for presenting data displays should be designed to minimize the occurrence of distinctive local features. Hence, this study showed that spatial arrangement of objects in displays is also important in user interface design. Koritzinsky [22] states that consistency can be maintained when the spatial and visual characteristics of objects persist over time. For example, if the folder icon is the same across all programs, dependability and reliability of the interface is enhanced. Also, the appearance of the user interface is consistent when similar items look alike. Consistency should also be

maintained in the interaction mechanism. The sequence of pointing, selecting or clicking should be the same throughout the application [40].

Kellogg [19] studied the impact of the conceptual dimension of consistency by prototyping a "consistent" version (common look and feel and conceptually consistent) and an "inconsistent" version (only common look and feel) of an interface. The results of the study, which incorporated a variety of measures like learning time, subjective satisfaction and more, showed that the "consistent" interface was better than the "inconsistent" (consistent in visual appearance and behavior only). Although visual appearance and behavior are very important aspects of consistency, they need to be combined with conceptual consistency in designing user interfaces. Tognazzini [42] says "you can change the entire look and feel of an application as long as you honor the user's previously learned interpretations and subconscious behaviors."

GUI Guidelines: The use of guidelines has been identified as important within the framework of a representation of the human-computer interface (HCI) by many experts [17]. Consistency facilitates positive transfer of skills from one system to another leading to ease of use, reduced training time and improved retention of operating procedures [31, 33]. However most guidelines do not include specifications of consistency or properties of a consistent interface that produce positive transfer of skills. Therefore, creating consistent interfaces involves developing proper standards and guidelines implying overhead in the software lifecycle. But as stated earlier, human factors techniques are not a burden to software development and applying these techniques properly can save both time and money in the software lifecycle. Also, careful analysis is required when adhering to common standards or guidelines as they can prevent enhance-

ments even when needed and can lead to a poor design. Frederiksen, Grudin and Laursen [11] demonstrated through an experiment that a consistency guideline, "The direction in which the contents of a window move when a mouse is used with a scrolling arrow at the edge of the window", which is strongly endorsed by the Apple Human Interface Guidelines [1], may lead to poor design for certain tasks. This study showed that consistency guidelines should be applied cautiously to be in harmony with the user's task.

Task Analysis: Kellogg [20] discussed the case of dragging document icons to different destinations in the Macintosh Interface. Dragging a document icon from one folder to another on the desktop moves the document. However dragging a document icon to a disk icon copies the document to the disk, leaving the desktop icon in place. The semantics of dragging a document icon in this design is inconsistent, but is this inconsistency good or bad? As Kellogg explained, it depends on an assessment of the context of use surrounding dragging; whether the user, when dragging a document to a different folder on the desktop is more frequently seeking reorganization or multiple copies in different locations. If the more frequent task is reorganization as the design assumes, then the inconsistency improves the usability.

According to Grudin [12] interface consistency is a largely unworkable concept and can sometimes work against good design. Also, consistency that supports ease of learning can conflict with ease of use. Grudin explained that consistency can be harmful with *Printing a folder of a directory* example. He considered the situation in which a folder containing several documents is selected and is followed by selection of print operation from the menu. The design question in this case is: What should be printed when the operation is executed?

Informal user studies suggested that documents in the folder should be printed. However, system architects argued that a list of documents within the folder should be printed. A folder in this system was a list of pointers to documents. Since in this case, printing a document produced the contents of the documents and it was argued that printing a folder should similarly produce a copy of its contents i.e. list of documents in the folder. Although developers argued that consistency which was based on software architecture was important, the design was rejected in favor of consistency within system architecture. Therefore wrong dimension of consistency was chosen but was consistent with what users wanted. The bottom line is that the interface should be consistent with the user's task and fulfill the requirements of the targeted users.

GUI Terminology: Another important component of consistency is terminology used in the interface. Terminology used in interactive dialogs for text editing can sometimes be problematic for the user [25]. Inconsistencies in low-level interactions can be frustrating, for example programs that differ in the names of important commands (e.g., "quit" and "exit") [12]. A study done by Long, Hammond, Barnard and Morton [25] showed that users in most text editing tasks refer first to objects in the text or in the system and then perform actions on them, so the use of proper terminology is an important factor in interactive dialog design. Several issues associated with the structuring of arguments with a set of commands were examined [3] in three studies on consistency and compatibility in human-computer interaction. It was found that software specialists tend to prefer positionally consistent systems in which a recurrent argument was most frequently made the first argument in the command string. Also, the users learned positionally consistent systems more readily when recur-

rent arguments were placed first. Users in different tasks involving the same information showed marked preferences for structure analogous to the natural language, with the direct object of the command verb emerging most frequently as the first argument.

Studies in the past comparing "natural language" terminology with computer oriented terminology for better user performance have given mixed results making the empirical evidence useless for designing guidelines for system developers. But, it has been shown that statistical performance differences in interactive dialog occur with command sets that vary in their structural and semantic attributes [14].

Abbreviations are constructed to reduce typing and optimize the use of screen space, but can impose significant cognitive demands on a user in a new application. To create internally consistent design, one abbreviation algorithm should be used [12]. In spite of some evidence that rule-based abbreviation can assist learning, if these rules are inconsistent, users may have to remember the structure and content of individual abbreviations and face multiple cognitive demands. Clearly, the use of abbreviation should be considered carefully in the design of GUI applications.

Tools for Consistent Design: While powerful tools have been developed enabling designers to create systems rapidly, not much work has been done on developing tools to create consistent interfaces. An important step in GUI design was taken by an Interactive Transition Systems (ITS) project [48]. The ITS approach was to generate consistent interfaces automatically by the use of executable style rules. ITS provided a set of software tools to support four application development roles: an application expert, a style expert, an application

programmer, and a style programmer. The ITS architecture divided the application into three parts, namely: application functions, a dialog manager, and views supporting user interfaces. This architecture helped to create a consistent interface for a family of applications and to create multiple consistent interfaces for a given application. ITS supported designers in generating consistent interfaces because it separated the application from the interface and linked them using executable style rules. ITS has been defined to go beyond toolkits, beyond traditional user interface management systems, to generate user interfaces automatically. ITS has generated interfaces for demonstration applications, but has not been used to create real-world applications.

Interface Evaluation Methods: Interface evaluation is a difficult and cumbersome process. Various techniques are used in design evaluation, including expert reviews, cognitive walk-throughs, heuristic evaluations, interface guidelines and many more. Evaluation of a software product's user interface using four techniques, namely heuristic evaluation, usability testing, guidelines and cognitive walk-throughs, showed that each has advantages and disadvantages [18]. For instance, heuristic evaluation identifies more problems than any other method, but it requires UI expertise and several evaluators. Similarly, usability testing identifies serious and recurring problems, but requires UI expertise and has a high cost. The requirements for these powerful methods, which may include availability of working prototypes, test users, expert evaluators and time constraints are hindrances in applying these methods more frequently. The study using these four evaluation methods also showed that usability testing, a powerful and effective evaluation method, is not good in evaluating design consistencies and missed consistency problems in its evaluation technique. Therefore,

consistency checking tools are necessary to aid usability testing.

Furthermore, usability testing works best for smaller applications. It is practically infeasible to analyze every dialog box in an application with hundreds of dialog boxes with the current evaluation methods. Finding anomalies or differences while reviewing hundreds of dialog boxes is even hard for expert reviewers, who may fail to detect some flaws and inconsistencies. In contrast, automated evaluation tools can be used in early prototypes (or late iterations) and can detect anomalies across thousand of dialog boxes. These automated tools, in addition to detecting anomalies, can make interfaces cleaner and easier to use. Although many organizations are adopting more stringent usability testing standards to monitor quality and layout of the design, better automated evaluation tools are needed which would scan for inconsistencies in the interface layout at early design and development stages, thereby providing an aid to the usability testing.

2.4.3 Evaluation Tools for Visual Design and Textual Properties

Automated tools for consistency checking are meant to replace the current manual consistency checking process which is complex, expensive, error prone, and time consuming. These tools can be made independent of platform and development environment, as visual and textual properties of the interface are independent of these factors.

Spatial organization of widgets in the interface is the central aspect of interactive visual design. In developing a system to evaluate alphanumeric displays, Tullis [43] derived six measures to describe screen formats of spatial arrays of

characters.

- **Overall density:** The number of characters on the screen, expressed as a percentage of the number of character spaces available.
- **Local Density:** The number of characters within a 5-degree visual angle of each character, expressed as a percentage of the number of spaces within that area. A measure of how highly packed the screen is.
- **Number of groups:** The number of distinct groups of characters detected using a proximity clustering technique.
- **Size of groups:** Average visual angle subtended by the groups of characters.
- **Number of items:** The number of distinct labels or data values on the screen.
- **Layout Complexity:** The amount of information, in bits, conveyed by the horizontal and vertical positions of the display items. A measure of how poorly aligned the screen elements are within each other.

These measures were later incorporated into a Display Analysis Program to analyze displays on IBM PC and PC-compatible computers, to develop a tool for objectively evaluating the usability of any alphanumeric display format [44]. This Display Analysis Program was developed with two systems created specifically to test these programs and then tested with the displays of seven previous studies [45]. The results indicated that given a set of alternative screen formats to present some alphanumeric data, this program can accurately predict the relative search times and subjective ratings for the formats. "Number of groups"

and "Size of groups" were found to be the most important display parameters in determining search time.

Streveler and Wasserman [41] proposed novel visual metrics to quantitatively assess screen formats which have similarities with Tullis's Display Analysis Program. They proposed three basic techniques for analyzing screen formats: "boxing", "hot-spot" and "alignment" analysis.

- **Boxing** is a technique for detecting groups. The boxing algorithm finds groups of characters that are completely surrounded by white spaces and then draws the smallest possible rectangular boxes around them.

- **Hot-Spot** analyzer calculates a moving average of "intensity" at each character position on the screen and is similar to the local density measure of Tullis display analysis program.

- **Alignment** analyzer calculates the number of "alignment points" for each column of the display.

A balance measure was also proposed by them which computed the differences between the center of mass of the array of characters and the physical center of the screen. These proposed metrics were not applied to any system to validate them. The applicability of Tullis's complexity proposition was later applied to the domain of interactive system design with findings strongly supporting their applicability [8]. Choosing screen density as the measure of complexity, it was found that Madd's Modified Discrepancy Hypothesis in psychology is applicable to user interface design, i.e. users performance and preference for screen complexity follows a U-shaped curve, with too little or too much complexity

depressing preference and performance. In other words, humans have preference and affinity for medium complexity. Furthermore, Kim and Foley [21] used metrics as a constant for the design space and the layout style. They developed a tool which generated potential design specifications and guidelines for the metrics. Effectiveness of their metrics has not yet been evaluated. The evolution of GUIs with multiple typefaces, colors, new kinds of widgets etc. means that more analysis is required and new metrics need to be implemented.

The evolution of modern user interfaces, like multimedia interfaces, has sparked research in automated evaluation based on visual techniques. Vanderdonckt and Gillo [47] proposed five visual techniques which identified more visual design properties than traditional balance, symmetry, and alignment:

- **Physical techniques:** balance, symmetry, regularity, alignment, proportion and horizontality.
- **Composition techniques:** simplicity, economy, understatement, neutrality, singularity, positivity and transparency.
- **Association and dissociation techniques:** unity, repartition, grouping, and sparing.
- **Ordering techniques:** consistency, predictability, sequentiality and continuity.
- **Photographic techniques:** sharpness, roundness, stability, leveling, activeness, subtlety, representation, realism and flatness.

Dynamic strategies for computer-aided visual placement of interaction objects on the basis of localization, dimensioning and arrangement have been introduced [5]. These techniques of localization, dimensioning and arrangement

were based on some of the visual techniques introduced by Vanderdonckt and Gillo [47]. Mathematical relationships were defined to improve the practicability, the workability and the applicability of the visual principles into a systematic strategy, but specific metrics and acceptance ranges were not tested. Visual techniques introduced fo multimedia layout frames have only rarely been applied to commercial applications.

Sears [36, 37] has developed a first generation tool (AIDE) using automated metrics for both design and evaluation using Layout Appropriateness metrics. In computing the Layout Appropriateness the designer provides the set of widgets used in the interface, the sequence of actions to be performed by the user and how frequently each sequence is used. The appropriateness of a given layout is computed by weighing the cost of each sequence of actions by how frequently the sequence is performed. Layout Appropriateness can be used to compare existing layouts and to generate optimal layouts for the designer. AIDE has demonstrated its effectiveness in analyzing and redesigning dialog boxes in simple Macintosh applications and also dialog boxes with complex control panels in NASA applications.

Mullet [30] developed a simple layout grid forming the basis of a systematic layout program embodying a set of guidelines that make it easy to position related controls consistently across dialogs. Using this systematic re-structuring and redesign approach he showed that the GUI of the "Authorware Professional", a leading development tool for learning materials in the Macintosh and Windows environments, could be easily redesigned to create a more coherent, consistent and less crowded layout.

2.5 Scientific Experiments on Effects of Interface Inconsistencies

As mentioned earlier, designing a user interface with a consistent visual design and textual properties is hard to achieve in the current generation GUI development process. The problems may include rapidly changing designs with new objects and relationships and with hundreds of dialog boxes designed by multiple designers. Designers often do not adhere to interface development guidelines or in some cases these guidelines may not even exist. In order to help designers identify these inconsistencies before usability testing, automated consistency checking tools have been developed to evaluate visual design and terminology in user interfaces [28].

Chimera and Shneiderman [6] performed a controlled experiment to determine the effects of inconsistency on performance. This experiment used two interactive computer systems at the National Library of Medicine which were an original inconsistent version and a revised consistent version. The original system had many problems including inconsistent wording of screen titles following the menu item selections, menu items being computer-oriented rather than task-oriented, a clutter of function key descriptions at the bottom of the screen etc. The revised version had consistent screen layouts and colors plus use of consistent task and domain oriented phrases for the description of menu items. The results showed that there was a statistically significant difference ($p<.01$) favoring the revised interface for five out of twenty tasks and only one task favored the original interface ($p<.01$). Also, the revised interface received a higher subjective satisfaction ratings . It was concluded that the revised inter-

face yielded faster performance and higher satisfaction due to how information was displayed with respect to location, wording and color choices.

Experimental studies have shown that inconsistencies in the interface reduces user's performance and subjective satisfaction. Bajwa [2] studied the effect of inconsistencies in color, location and size of widgets (in this case buttons) on user's performance and subjective satisfaction. To test the hypothesis that inconsistency deteriorates performance and subjective satisfaction, four versions of a Billing System interface were created in Visual Basic. The original version was consistent in accordance with most windows applications. The other three versions were made inconsistent with respect to color, location, or size, so that every inconsistent version had about 33% of only one type of inconsistency. The experiment was divided into three phases, namely inconsistent color versus consistent interface, inconsistent location versus consistent interface and inconsistent size versus consistent interface. For every phase of the experiment, subjects used both the consistent and inconsistent version with 50% of the subjects using the inconsistent version first and the other 50% using the consistent version first. A total of 60 subjects participated in the experiment with 20 subjects in each phase. The results of the experiment showed that inconsistency in color, location and size of objects significantly effects user's performance by about 5%. Although studies have been done to check the effects of inconsistencies in the interface design on user's performance, no experiments have been done for terminology inconsistencies specifically. From these previous experimental studies we can derive that some of the benefits of consistency in GUI design include:

- Decrease in task completion time.
- Reduction in error rates.

- Increase in subjective satisfaction.
- Reduction in search, response and learning time.
- Reduction in working memory load.

Chapter 3

Description of the Evaluation Tools

3.1 Metrics Evaluation Using Canonical Format

The present research evolved from the concept of converting interface form files generated by Visual Basic into canonical format files and feeding them as input to the *SHERLOCK*. The canonical format is an organized set of GUI object descriptions. These object descriptions are enclosed in curly braces and embrace interface visual design and terminology information in a sequence of attribute-value pairs. The canonical format is advantageous because of its lucidity and extendibility. It can be easily modified to include any new attributes encompassing interface description information in the form files. A canonical format file uses unique keys for all the objects that are constructed via parent-self ID pairs. (If a parent-self ID pair is not unique, then all the non-unique occurrences

get enlarged to be parent-parent-self ID triples, and so on until uniqueness is achieved.) The attribute called "unique-id" is provided in the event that the originating system has the capability to supply unique numbers to objects. If "unique-id" is not supplied, the value may be filled in by expansion programs. A generic object can list any of the following attribute-value pairs. Those marked with an asterisk (*) are required and have been used by *SHERLOCK*.

{*type	Object-value
*parent	ID
*resource-id	String
name	String
variable-reference	String
unique-id	Number
*top	Coordinate (relative to parent)
*left	Coordinate (relative to parent)
*width	Coordinate
*height	Coordinate
border-is-visible	Boolean
*background-color	Color
*foreground-color	Color
*font-family	String
*font-size	Number
*font-is-bold	Boolean
*font-is-italic	Boolean
font-is-underline	Boolean
font-is-serif	Boolean
label-type	Label-value
*label	String
label-background-color	Color
label-foreground-color	Color
label-justification	Justification-value
label-placement	Placement-value
label-font-family	String
label-font-size	Number
label-font-is-bold	Boolean
label-font-is-italic	Boolean
label-font-is-underline	Boolean

label-font-is-serif	Boolean
margin-top	Coordinate (relative to attribute "top")
margin-left	Coordinate (relative to attribute "left")
margin-bottom	Coordinate (relative to calculated "bottom")
margin-right	Coordinate (relative to calculated "right")
navigation-order	Number
navigation-type	Navigation-value
help-text	String
is-default	Boolean (uses window origin)
is-cancel	Boolean (uses window origin)
is-help	Boolean (uses window origin)
mnemonic	Character
menu-accelerator	Key-value (type Menu-item only)
menu-is-pinnable	Boolean (type Menu only)
menu-content	Menu-value (big hierarchical description)
task	String
comments	String

}

The attribute names listed above are self-explanatory, but many of the value types appearing in the canonical format are abstractions. The following table specifies value types that are not completely obvious. For some of the values like Unit-value and Object-value more components can be defined.

```
Number ::= nonnegative integer
ID ::= String | Number
Coordinate ::= real number (uses top-left origin)
Unit-value ::= Unit-pixel | Unit-point | Unit-MS-Windows dialog-unit
Color ::= String | Color-specification
Color-specification ::= RGB-triple | HSB-triple
Label-value ::=  String | Icon | String-and-icon
Justification-value ::= Justify-left | Justify-center | Justify-right
Placement-value ::= Place-left | Place-right | Place-top | Place-bottom
Navigation-value ::= Nav-parent | Nav-window (whether navigation is
relative to parent or entire window
Character ::= a single alphanumeric character value
Modifier ::= Shift-key | Control-key | Alt-key
Key-value ::= Character + Modifier | Function-key + Modifier
Object-value ::= Window | PopupWindow | Modal-PopupWindow |
   Pushbutton | CheckButton | RadioButton |
```

```
   Container | StaticLabel |
   ScrollingListOneSelection | ScrollingListMultipleSelection
   ScrollingListContiguousSelection
   MenuHighestLevel | Menu | MenuItem | MenuSeparator |
   TextPane | TextPaneReadonly |
   TextPaneOneLine | TextPaneOneLineReadonly | ....
   (basically responsibility of metric conversion
   program to know about these values)
Menu-value ::= hierarchical object description of only
 Object-types  Menu | MenuItem | MenuSeparator
```

A translator program was developed to convert the Visual Basic form files into a canonical format. This translator distills relevant information from the form files. Another translator program was created to convert the Visual C++ resource files into the canonical format and is currently being tested. These canonical formats are platform independent and may be created for other interface development tools like Power Builder, Galaxy and XVT by writing a translator program for those tools. Thus, *SHERLOCK* is not specific to Visual Basic or Visual C++ and can be used for evaluating interfaces created in other environments.

3.2 Evaluation Tools

The objective of developing *SHERLOCK* was to provide designers rapid feedback on possible flaws and inconsistencies from early prototyping to advanced development stages. Our research started with analyzing a single dialog box and was extended towards checking consistency across multiple dialog boxes, typical in most modern user interfaces. Our focus was on evaluating only certain aspects of consistency in user interfaces which are task-independent and can be

automated. One of the task-independent features evaluated by our tools is visual design which includes properties such as sizes of similar screens, placement of similar items, screen density, consistency in margins, screen balance and alignment. Consistency in other visual design properties such as fonts, font-sizes, font-styles and background and foreground colors has also been evaluated. Finally our evaluation includes checking for terminology consistency, abbreviations, variant capitalization and spelling errors.

Development of *SHERLOCK* is an extension of previous work [39] in which spatial and textual evaluation tools were constructed. These tools have been modified after evaluating sample applications and new tools have been integrated with them into a family of consistency checking tools leading to the evolution of *SHERLOCK*.

- **dialog box summary table** to give an overview of visual design properties of the interface dialogs.
- **margin analyzer** to provide the designer with feedback on which margins are inconsistent with the most frequently occurring value and how to modify them.
- **interface concordance** to spot variant capitalization and abbreviation in button widgets.
- **button concordance** to spot variant capitalization, distinct typefaces, distinct background colors and variant sizes in all the interface buttons.
- **button layout table** to spot any inconsistencies in height, width and relative position among a given group of buttons.

- **interface speller** to detect terms used in the interface that are nonexistent in the dictionary.

- **terminology basket** to provide the interface designer with feedback on possibly misleading synonymous computer terms.

3.2.1 Dialog Box Summary Table

The dialog box summary table is a compact overview of the visual design of dozens or hundreds of dialog boxes of the interface. Each row represents a dialog box and each column represents a single metric. Typical use would be to scan down the columns looking for extreme values, spotting inconsistencies, and understanding patterns within the design.

Choosing the appropriate metrics set was the most important factor in the design of the dialog box summary table. The researchers at the University of Maryland generated a list of approximately 40 metrics which were constructed after reviewing the relevant previous literature, consulting with colleagues and using their GUI evaluation experience. A similar effort was taken on the GE side, where they brain-stormed and proposed their metric set based on their commercial software development experience. The two lists had many similar items and the lists were grouped into categories such as consistency, spatial layout, alignment, clustering, cluttering, color usage, fonts, attention getting, etc. A second independent brainstorming session was used to choose an ordered list of metrics for initial implementation. The items on the list were ranked with preference given to items which were expected to have high payoff and be easy to implement.

The metric set has been revised several times after evaluating a series of

interfaces. The metrics that were ineffective have been removed and others have been redefined and new metrics have been added. Some of the metrics have been further analyzed and converted to separate evaluation tools. The modified column set of the dialog box summary table is explained below:

Aspect Ratio: The ratio of the height of a dialog box to its width. Numbers in the range 0.5 thru 0.8 are desirable. Dialogs that perform similar functions should have the same aspect ratio. Fig 3.1 shows two dialog boxes which are designed to perform similar tasks, but have different aspect ratio.

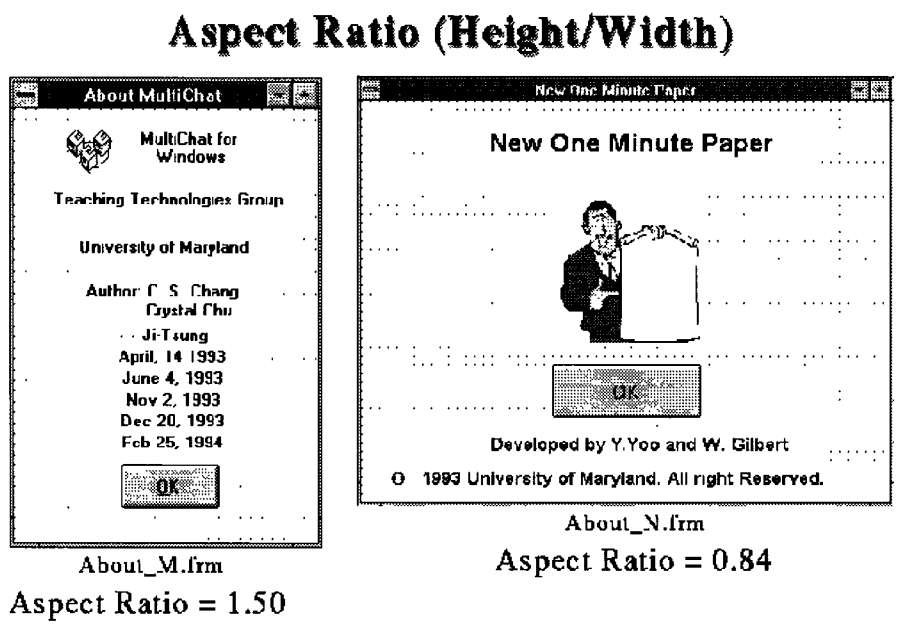


Figure 3.1: Aspect Ratio Calculation

Widget Totals: Count of all the widgets and the top level widgets. In-

creasing difference between all and top level counts indicates greater nesting of widgets, such as buttons inside containers.

Non-Widget Area: The ratio of the non-widget area to the total area of the dialog, expressed as a percentage. Numbers closer to 100 indicate high utilization, and low numbers (<30) indicate possibilities for redesign.

Widget Density: The number of top-level widgets divided by the total area of the dialog box (multiplied by 100,000 to normalize it). High numbers greater than 100 indicate that a comparatively large number of widgets are present in a small area. This number is a measure of the 'crowding' of widgets in the dialog.

Margins: The number of pixels between the dialog box border and the closest widget. The left, right, top and bottom margins should all be approximately equal to each other in a dialog box, and should also be the same across different dialog boxes.

Gridedness: Gridedness is a measure of alignment of widgets. This metric has been refined several times, but we have not been able to find a perfect metric to detect misaligned widgets. In our earlier definition we defined Gridedness to be the ratio of the total number of widgets in a dialog box to the number of distinct X or Y positions that the widgets have. X-Gridedness counts the number of stacks of widgets with the same X coordinates (excluding labels). Similarly Y-Gridedness counts the number of stacks of the widgets with the same Y coordinates. High values of X-Gridedness and Y-Gridedness indicate the possibility of misaligned widgets. An extension of Gridedness is Button Gridedness where the above metrics are applied to button widgets. Gridedness is conceptually similar to Tullis's Layout Complexity metric [45] and Streveler and Wasserman's Alignment metric [41]

Fig 3.2 shows how the Gridedness metric works. The dialog box on the right half of the figure has all the three buttons (OK, No and Cancel) with the same Y-coordinates, so the Y-Gridedness in buttons is 1 and X-Gridedness in buttons is 0. On the otherhand, the dialog box on the left half of the figure has a misaligned Cancel button from the OK and No buttons which have the same Y-coordinate. Thus, the Cancel button has a different Y-value forming a different stack, this misalignment introduces a new stack in both X and Y direction increasing the Y-Gridedness to 2 and X-Gridedness to 1. So every time there is a misaligned button the X- and Y-Gridedness values are increased by 1.

Area Balances: A measure of how evenly widgets are spread out over the dialog box. There are two measures: a horizontal balance, which is the ratio of the total widget area in the left half of the dialog box to the total widget area in the right half of the dialog box; and the vertical balance, which uses top area divided by bottom area. High values of balances between 4.0 and 10.0 indicate screens are not well balanced. The limiting value 10.0 represents a blank or almost blank (for example, a dialog box that has only one widget which is a button) dialog box.

Distinct Typefaces: Typeface consists of a font, font size, bold and italics information. Each distinct typeface in all the dialog boxes is randomly assigned an integer to facilitate quick interpretation. For each dialog box all the integers representing the distinct typefaces are listed so that the typeface inconsistencies can be easily spotted locally within each dialog box and globally among all the dialog boxes. The idea is that a small number of typefaces should be used for all the dialog boxes.

Distinct Background Colors: All the distinct background colors (RGB

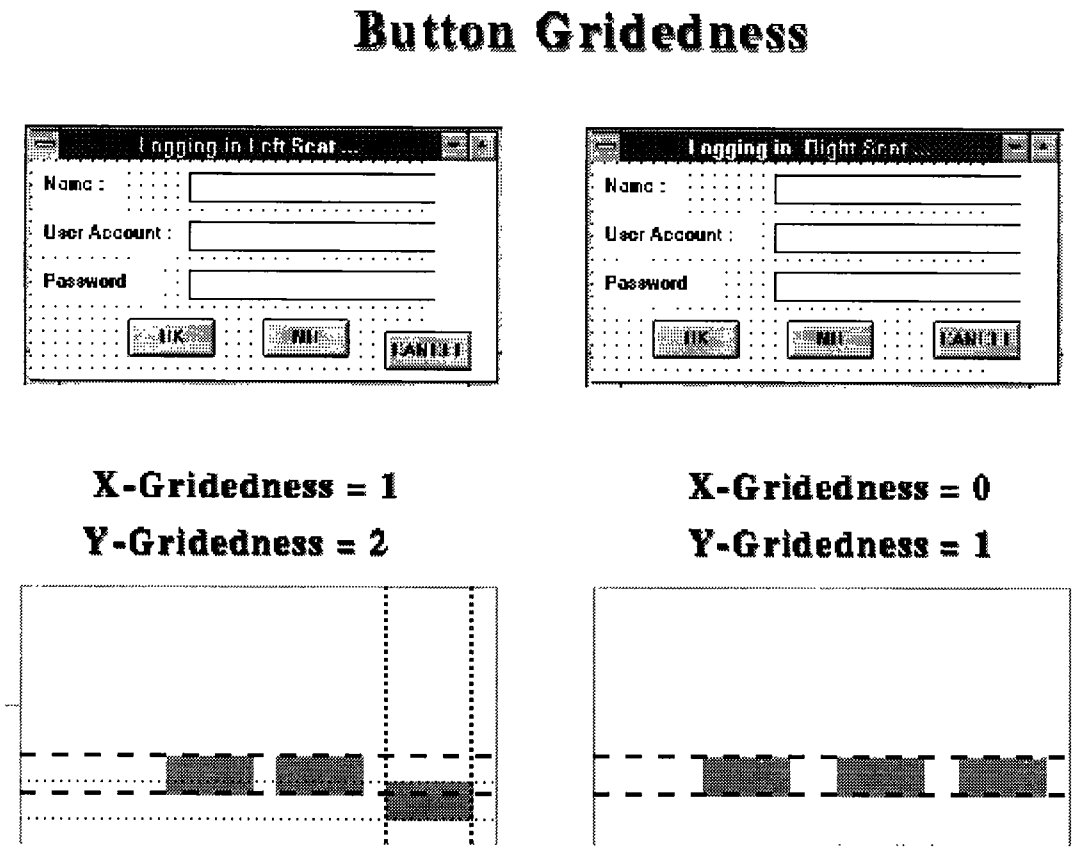


Figure 3.2: Gridedness mechanism

values) in a dialog box are displayed. Each distinct color is randomly assigned to an integer for display and comparison convenience and is described in detail at the end of the table. The purpose of this metric is to check if all the dialog boxes have consistent background colors. Multiple background colors in a dialog box may indicate inconsistency.

Distinct Foreground Colors: Similar to distinct background colors, displays all the distinct foreground colors in a dialog box. The purpose of this metric is to check if all the dialog boxes have consistent foreground colors. The colors in all the analyses are listed as RGB values which would correctly distingusheven

shades of light gray.

3.2.2 Margin Analyzer

Margin analyzer is an extension of the dialog box summary table's margins metric. This analyzer calculates the most frequently occurring values of left, right, top and bottom margins across the interface and then lists margins in every dialog box which are inconsistent with these frequently occurring values. It also calculates what widgets of the dialog box need to be moved by how many pixels to make the margins consistent. The Margin analyzer tool depends on the fact that the most frequently occurring value of margins are the optimum margin values which the designer would have ideally used for consistency.

3.2.3 Concordance

The concordance tool was built to extract all the words that appear in labels, buttons, menus, user messages, etc. in every dialog box. The concordance tool helps designers with appropriate word use such as spelling, abbreviation, tense consistency, case consistency, passive/active voice etc. Occurrences of words in a different case are preserved as unique occurrences of words in a sorted list to point out the use of different case. The sort order used was aAbB...zZ so that the occurrence of "cancel" is not separated from "Cancel" or "CANCEL".

The concordance tool has been broken down further to extract specific information related to spelling, abbreviation and case consistency to expedite the quick evaluation process.

3.2.4 Interface Concordance

The interface concordance tool checks for variant capitalization for all the terms that appear in buttons, labels , menus, etc. in every dialog box of the interface. This tool outputs strings which have variant capitalization, listing all the variant formats of the string and its dialog box sources. These variant forms are spelling differences and may be acceptable, but they may be something that should be reconsidered. For example the words "MESSAGES", "messages", "Messages" and "mesgs" are variant capitalization forms of the same word.

3.2.5 Button Concordance

As buttons are one of the most frequently used widgets performing vital functions like "Save", "Open", "Delete", "Exit" etc., checking consistency in their size, placement, typefaces, colors and case usage becomes more important. This tool outputs all the buttons used in the interface, listing the dialog boxes containing the buttons plus fonts, colors and button sizes. The button concordance identifies variant capitalization, distinct typefaces, distinct foreground colors and variant sizes in buttons.

3.2.6 Button Layout Table

Given a set of buttons that frequently occur together (for example, OK Cancel, Close, Help), if the first button in the set is detected in the dialog box then the program outputs the height, width and position relative to the first button of every button detected in the list. The relative position of every button detected in the set is output as (x + offset, y + offset) to the first button, where offset is

in pixels. Buttons stacked in rows would yield a (x + offset, y) relative position and those stacked in columns would yield (x, y + offset). The Button Layout table identifies inconsistencies in button placement, inconsistencies in button terminology and variant button sizes locally within a dialog box and globally across all the dialog boxes.

Our program initializes by reading an ASCII file containing different sets of detector buttons. These button sets were constructed after analyzing many previously developed interfaces. Variations in terminology were considered while constructing these button sets. Button set (Start Stop Exit) is incomplete as designers may use "Close" , "Done" or "Cancel" instead of "Exit". The set (Start Stop Halt Pause Cancel Close Done End Exit Quit) forms a much better button detector set. Some of the sample button detector sets are:

- OK Cancel Close Exit Quit Help
- Start Stop Halt Pause Cancel Close Done End Exit Quit
- Add Remove Delete Copy Clear Cancel Close Exit
- Help Close Cancel Exit

3.2.7 Interface Speller

Interface Speller is a spell checking tool which reads all the terms used in widgets including menus, buttons, list boxes, combo boxes etc. throughout the interface and outputs terms that are not found in the dictionary. The spell checking operation is performed within the code and all the possible misspelled words are stored in a file. This file can be reviewed by the designer to detect possible

misspelled and abbreviated words which may create confusion for end users. The output is filtered through a file containing valid computer terms and default Visual Basic terms that may be flagged as spelling errors by the dictionary.

3.2.8 Terminology Baskets

A terminology basket is a collection of computer terms including their different tense formats which may be inadvertently used as synonyms by the interface designers. Our goal is to construct different sets of terminology baskets by constructing our own computer thesaurus and then search for these baskets in every dialog box of the interface. The purpose of terminology baskets is to provide interface designers with feedback on misleading synonymous computer terms, like "Close", "Cancel", "End", "Exit", "Terminate", "Quit".

The program reads an ASCII file containing the basket list. These baskets are sorted alphabetically and for each basket all the dialog boxes containing any of the basket terms are output. The list of baskets may be easily updated, as more interfaces are analyzed in the future. Some of the idiosyncratic baskets are:

- Remove Removes Removed Removing Delete Deletes Deleted Deleting Clear Clears Cleared
- Clearing Purge Purges Purged Purging Cancel Cancels Canceled Canceling Refresh Refreshed
- Item Items Entry Entries Record Records Segment Segments Segmented Segmenting Field Fields

- Add Adds Added Adding Insert Inserts Inserted Inserting Create Creates Creating

- Message Messages Note Notes Letter Letters Comment Comments

Chapter 4

Interface Evaluations

4.1 Testing the Evaluation Tools

The effectiveness of *SHERLOCK* tools has been determined by evaluating four commercial prototype applications developed in Microsoft Visual Basic. These applications included a 139 and 30 dialog box GE Electronic Data Interchange Interface, a 75 dialog box Italian Business Application, and a set of University of Maryland AT&T Teaching Theater Interfaces combined together into an 80 dialog box application. The testing method incorporates a sequence of steps beginning with applying the tools to the prototype application, followed by analysis and review of the interface screen shots and output generated by *SHERLOCK*. *SHERLOCK* tools were not created with reference to any particular test prototypes and can evaluate any interface that is converted to the canonical format.

4.1.1 Evaluation Results, GE Interfaces

The 139 dialog box GE Electronic Data Interchange Interface was the first prototypes evaluated. Although this was a well-reviewed and polished design, *SHERLOCK* detected some inconsistencies which may have otherwise been left undetected. Another small 30 dialog box GE interface was evaluated which also revealed inconsistencies.

Dialog Box Summary Table Analysis

A portion of the dialog box summary table is shown in the figure 4.1

Aspect Ratio: Aspect Ratio varied from 0.32 to 1.00 and some dialog boxes which performed the same functionality had different aspect ratio, which was an inconsistency.

Non-widget Area: Non-Widget Area varied from 2% to 97.5%. Some dialog boxes with low Non-widget area (5% to 15%) were candidates for redesign.

Widget Density: Widget Density varied from 14 to 271, but most of the high values were due to exceptions in the metric, as none of the dialog boxes had too many widgets in a small area. The variation in widget density across dialog boxes can be easily seen by graphing the widget density metric [fig. 4.2].

Margins: Left, right, top and bottom margins were inconsistent within a single dialog box and were also inconsistent across the interface. For example, the average value of the left margin was 12 pixels, but the margin value ranged across the interface from 0 to 80 pixels. A graphical representation of left margin inconsistencies is shown in figure 4.3.

Gridedness: Some high values of the Button Gridedness metric helped in detecting dialog boxes with misaligned buttons.

No.	Dialog Name	Aspect Ratio (H/W)	-WIDGET-- TOTALS All	Top-Level	Non-Widget Area (%)	Widget Density widget/area	--- M A R G I N S ------ Left	Right	Top	Bottom (pixels)	----GRIDEDNESS----- Top X	Level Y	Buttons X	Y	-Balances-- Area Ratios Horiz (L/R)	Vert (T/B)	Distinct Typefaces	Distinct Background Colors	Distinct Foreground Colors
129	statupd.cft	0.75	4	3	43 1	64	16	14	16	7	1	1	1	1	1 0	1 4	1	1	2
130	sview.cft	0 77	27	17	13 9	63	0	-1	0	3	1	2	0	1	0 8	0 9	1	1	2 3 8
131	tempsel cft	0.55	9	8	33 9	76	8	9	8	4	1	1	1	0	1 3	0.6	1	1	2 3
132	tpsel cft	0 45	10	9	49 3	91	8	10	8	7	2	1	2	0	1.0	0 5	1	1	2 3
133	trans cft	0 50	10	9	71 2	121	24	24	8	4	2	0	1	0	1 1	0.5	1	1	2 3
134	tsexpl cft	0 62	3	2	50.8	43	16	18	8	21	1	1	1	1	1 0	4.2	1	1	2
135	turnard.cft	0 47	3	2	37 7	70	8	-2	8	8	1	1	1	1	0.9	1 9	1	1	2
136	updates cft	0 57	7	6	71.2	161	24	11	16	6	1	2	0	1	1 1	0.8	1	1	2 3
137	waste.cft	0.68	10	8	12 8	73	0	-1	0	5	1	2	0	1	1 0	1.2	1	1	2 3
138	wastedef.cft	0 65	15	7	35 3	75	8	3	8	2	2	4	1	0	1.4	0.8	1	1	2 3
139	xref.cft	0 65	14	13	7.6	54	0	-1	0	0	1	2	0	1	1.0	0 6	1	1 4	2 3
Maximum		1.00	170	31	97.5	271	80	40	24	27	4	5	5	4	6 2	8.6			
Minimum		0 32	3	2	0.2	14	0	0	0	0	0	0	0	0	0.0	0.0			
Average		0.59	17	8	42.7	86	12	9	7	6	1	1	0	0	1.1	1.4			

DISTINCT TYPEFACES:
1 = MS Sans Serif 8 25 Bold
2 = MS Sans Serif 8.25
3 = MS Sans Serif 9.75 Bold Italic
4 = MS Sans Serif 8.25 Bold Italic
5 = Arial 8.25 Bold
6 = MS Sans Serif 18 Bold
7 = MS Sans Serif 9.75 Bold

DISTINCT BACKGROUND COLORS:

1 = 80000005
3 = ff
4 = ffffff
5 = c0c0c0
6 = 808000
7 = 0
8 = 808080
10 = 8000000c

DISTINCT FOREGROUND COLORS

2 = 80000008
3 = ff
7 = 0
8 = 808080
9 = 80
11 = 3
12 = 8000000e

Figure 4.1: Portion of the Dialog Box Summary Table

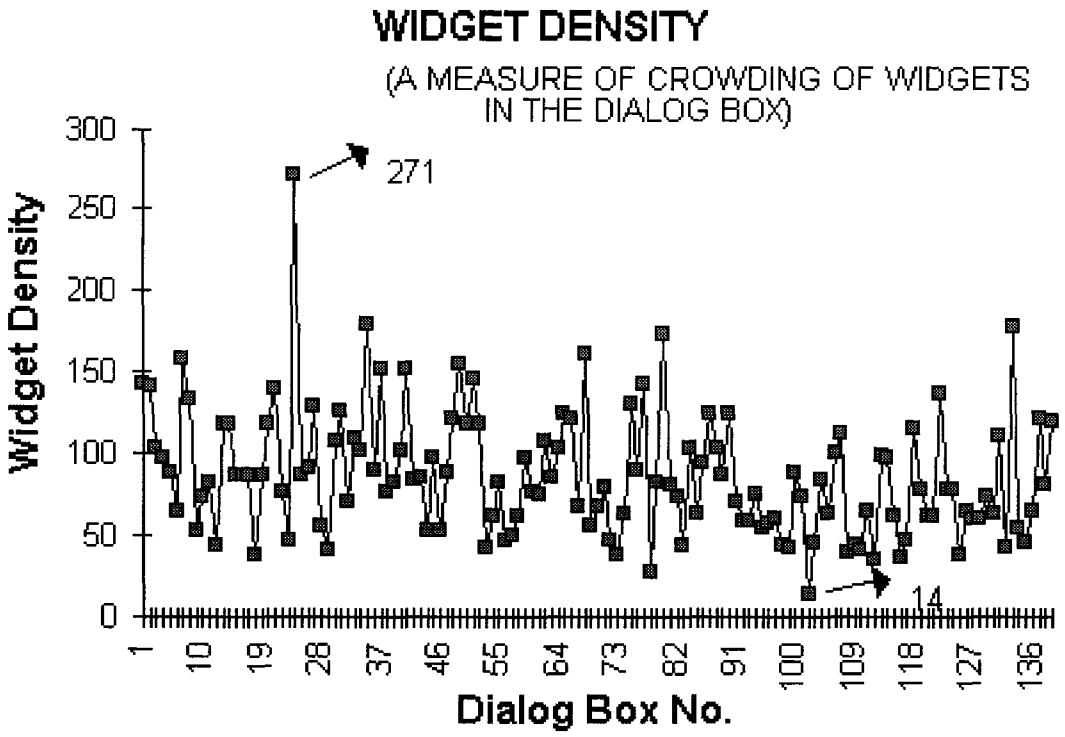


Figure 4.2: Graphical Representation of Widget Density Metric Results

Area Balances: Dialog boxes were well balanced as the average value of Left/Right Balance and Top/Bottom Balance was 1.1 and 1.4 respectively.

Distinct Typefaces: Although most of the dialog boxes used a single typeface (MS Sans Serif 8.25 Bold), there were a couple which used more than three typefaces [fig. 4.4]. Altogether seven distinct typefaces were used.

Distinct Background & Foreground Colors: There was much variation in color usage among different dialog boxes, indicating inconsistency. The interface used a total of eight foreground colors (RGB Values) and seven background colors (RGB Values).

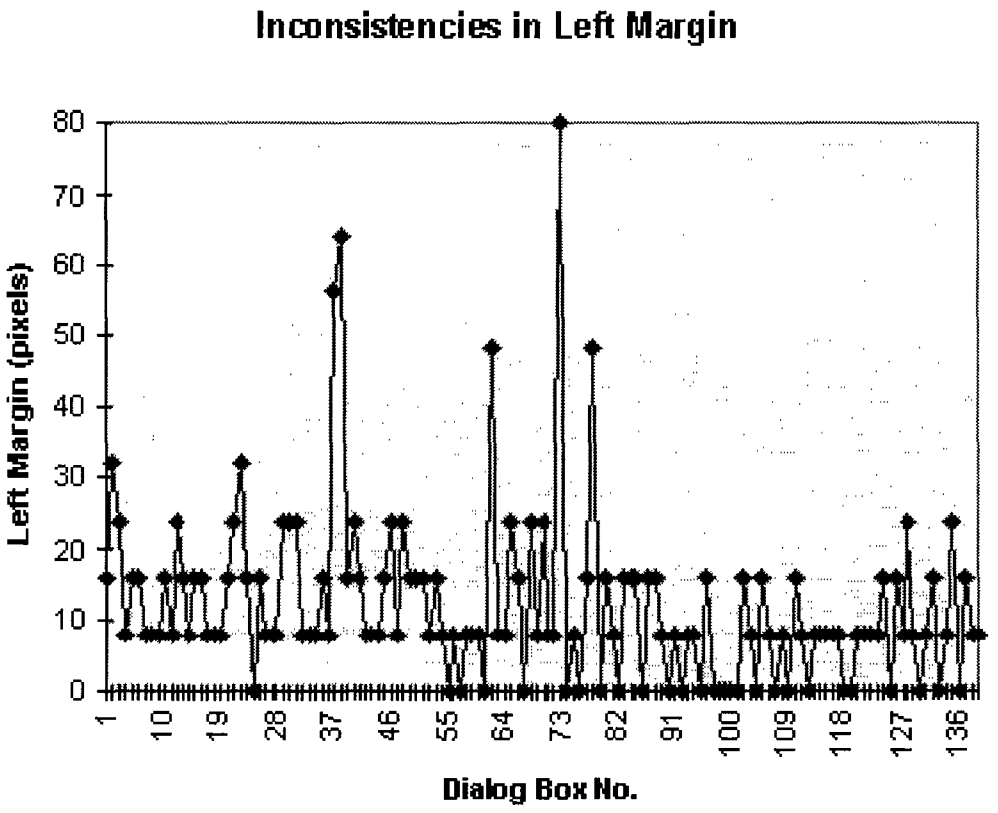


Figure 4.3: Graphical Representation of Left Margin Metric Results

Margin Analyzer

The margin analyzer successfully detected the dialog boxes which had margin values more than two pixels apart from the most frequently occurring value in both the applications. For each inconsistent value, it listed the widgets that need to be moved and by how many pixels to alleviate the inconsistency. A portion of the margin analyzer for the large 139 dialog box application is shown in the figure 4.5. In the case of the smaller application, the margin values were highly inconsistent, since most frequently occurring margin values had a maximum envelope of six to eight dialog boxes, making the other values inconsistent. These

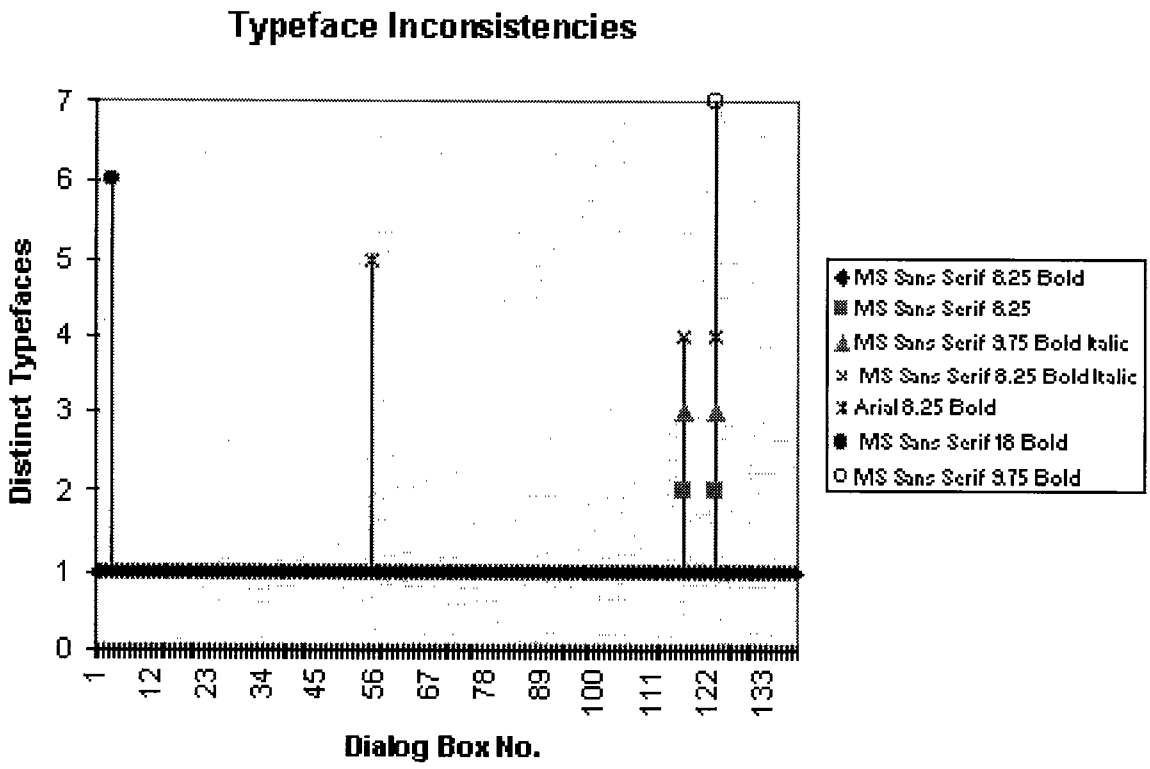


Figure 4.4: Graphical Representation of Typeface Inconsistencies

results showed that the design didn't adhere to any concept of consistent margins. There were some exceptions in Visual Basic (the tool allowing widgets to extend beyond the area enclosed by the dialog box and size of label and text boxes being greater than size of text enclosed by them) were beyond the capability of the tool to handle, leading to negative margin values.

Interface Concordance

The interface concordance tool spotted the terms that used more than one case across the application. For example, terms like "Messages", "MESSAGES" and "messages" were detected by the interface concordance tool. Some of the other inconsistencies included terms like "Item", "item" and "Item", "Open", "OPEN" and "open".

Button Concordance

The output from the button concordance tool show that the GE Interface did not have any button labels which used more than one case. All the button labels used the title format and were therefore consistent. Also, all the buttons used the same typeface and foreground color. Button Concordance detected inconsistency in height and width of the buttons across the interface. For example, the width of the "OK" button varied from 57 to 105 pixels, with very few buttons having the same width. Similarly the width of the "Find" button varied from 63 to 153 pixels, with no two buttons having the same width. Also, inconsistencies in placement of buttons across the interface were detected by the tool, using the left, right and top button position metric. The table below shows a portion of the button concordance output for the "Archive" button. Browsing across the columns of the table, we can see that the width of the "Archive" button varies between 65 and 105 pixels. All the buttons have a top margin of 0 pixels except one (file.find.cft) which has a top margin of 312 pixels. This is an inconsistency, since all the "Archive" buttons are placed at the top right corner of the dialog box except one which is placed at the bottom right corner. Button placement inconsistencies were detected in many other buttons including "OK", "Cancel",

No	DIALOG NAME	MARGINS LEFT	RIGHT	TOP	BOTTOM	COMMENTS THE FOLLOWING MARGINS MAY BE CHANGED BY MOVING THE LISTED WIDGET BY THE GIVEN NO OF PIXELS FOR CONSISTENCY
10	admpwd.cft	16	5	8	5	Left Margin: Widget container-9 to the left by 8 pixels
11	adrmsg.cft	8	5	8	7	
12	adrmsg2.cft	8	5	8	6	
13	adrmsg3.cft	8	-3	8	7	
14	advsched.cft	16	7	16	13	Left Margin: Widget label-4 to the left by 8 pixels Top Margin: Widget label-4 up by 8 pixels Bottom Margin: Widget button-2 down by 6 pixels
15	afile2.cft	16	17	8	13	Left Margin: Widget label-4 to the left by 8 pixels Right Margin. Widget label-3 to the right 10 pixels Bottom Margin: Widget button-2 down by 6 pixels
16	alert1.cft	8	2	8	4	Right Margin: Widget label-5 to the left by 5 pixels Bottom Margin: Widget label-5 up by 3 pixels
17	archive.cft	8	17	8	26	Right Margin: Widget list-box-7 to the right 10 pixels Bottom Margin: Widget label-21 down by 19 pixels
18	archok.cft	8	-5	8	6	
19	asgnfam.cft	16	15	8	3	Left Margin: Widget label-11 to the left by 8 pixels Right Margin: Widget button-8 to the right 8 pixels Bottom Margin. Widget label-9 up by 4 pixels
20	autoff.cft	16	7	8	4	Left Margin: Widget label-10 to the left by 8 pixels Bottom Margin: Widget label-8 up by 3 pixels

Figure 4.5: Portion of the Margin Analyzer Table (GE Interface)

"Clear", "Close", "Find", "Forward", "Print" and "Open".

BUTTON LABEL	DIALOG BOX	BUTTON TYPEFACE	BUTTON FG_COLOR	BUTTON (H , W)	BUTTON POSITION LEFT	RIGHT	TOP
Archive	xref.cft	1	1	25,105	208	311	0
	file.cft	1	1	25,89	448	87	0
	file2.cft	1	1	25,73	360	72	0
	filefind.cft	1	1	25,73	408	142	312
	hold.cft	1	1	25,65	320	55	0
	in.cft	1	1	25,81	464	79	0
	out.cft	1	1	25,73	304	55	0
	sent.cft	1	1	25,81	344	78	0

DISTINCT TYPEFACES IN BUTTONS:

1 = MS Sans Serif 8.25 Bold No Label

DISTINCT FOREGROUND COLORS IN BUTTONS:

1 = Default Color

Interface Speller

The tool detected few misspelled terms, but many potentially confusing incomplete and abbreviated words such as "Apps", "Trans", "Ins" , "Oprs" were found in both the applications. A portion of the output of the spell checking tool for the large 130 dialog box GE Interface is shown below:

DIALOG BOX	TERMS NOT FOUND IN THE DICTIONARY			
addfamdf.cft	Doc	Msg		
addr.cft	App	EDI	HDLR	UNDA
admpwd.cft	ADMIN	EDI		
contacts.cft	Provence	Quik		
docsearc.cft	ILOG	Interchg		
dp.cft	NAD	Trans		

```
profile.cft     Ctrl            Provence
```

Terminology Baskets

The basket browser revealed some interesting terminology anomalies after analyzing the large 130 dialog box interface that led to reconsideration of the design. As shown below terms like "record", "segment", "field" and "item" were used in similar context in different dialog boxes. Other interesting inconsistencies included the use of "start", "execute" and "run" for identical tasks in different dialog boxes . Also, the small 30 dialog box interface had terminology inconsistencies, such as using the terms like "Show", "View", and "Display" to perform similar tasks.

```
---------------------------------------------------------------------------
Basket: Entries, Entry, Field, Fields, Item, Itemized, Itemizing, Items
        Record, Records, Segment, Segmented , Segmenting, Segments
---------------------------------------------------------------------------
BASKET TERM        FORM CONTAINING THE BASKET TERM
Field              search.cft
Items              reconly.cft       reconly.cft       reconly.cft
                   reconly.cft       sendrec.cft       sendrec.cft
                   sendrec.cft       sendrec.cft       wastedef.cft
Record             ffadm.cft         profile.cft
Segment            addr.cft          search.cft
```

Button Layout Table

The Button Layout Table revealed inconsistencies in button sizes, terminology, and placement within a dialog box and across the interface in both GE applications. The most common button positional and terminology inconsistency was in

the button set [OK Cancel Close Exit Help], the button labels "Cancel", "Close" and "Exit" were used interchangeably. Sometimes these buttons were stacked in a column on the top left corner of the dialog box and in other cases they were stacked in a row at the bottom of the dialog box and were either left, right or center aligned.

Output of the button detector set (OK Cancel Close Exit Quit Help) tested with the small 30 dialog box GE application is shown below. Inconsistency in height and relative button positions within a button set can be checked by moving across the rows of the table. Inconsistency in height and relative position for a particular button can be spotted by moving down the columns. For example, browsing the "OK" button column we found that the height of the "OK" button varied between 22 and 26 pixels and the width varied between 62 and 82 pixels. Also, scanning across the rows, we found that the relative position between "OK" and "Cancel" buttons varied in all the three dialog boxes in which they occurred together. In the dialog boxes "nbatch.cft" and "systinp.cft" the "Cancel" button was 20 pixels and 13 pixels below the "OK" button, but in the dialog box "admprof.cft" the buttons occurred next to each other in the same row. Also, both the buttons "Cancel" and "Exit"[fig. 4.6] were used with the "OK" button essentially to perform the same task. This is a terminology inconsistency.

DIALOG BOX	OK	Cancel		Exit		Help	
	(H,W)	(H,W)	Rel. Pos.	(H,W)	Rel. Pos.	(H,W)	Rel. Pos.
admprof.cft	22,68	22,68	x+16, y			22,68	x+98, y
checkpsw.cft	25,82			25,82	x+18, y	25,82	x+116, y+1
nbatch.cft	25,62	25,62	x-1, y+20			25,62	x, y+66
systinp.cft	26,72	26,72	x+1, y+13			25,73	x+2, y+48

Output of the button detector set (Add Remove Delete Copy Clear Cancel Close Exit) tested with the 130 dialog box GE interface is shown below. The heights of all the "Add" buttons were constant (25 pixels) but the width varied from 65 pixels to 97 pixels. Also, the relative position between the "Add" and "Remove" buttons varied in all three dialog boxes in which they occurred together. In the dialog box "archive.cft" and "autoff.cft" the "Remove" button was 15 pixels and 39 pixels below the "Add" button, but in the dialog box "dp.cft" the buttons occurred next to each other in the same row. Also, both the buttons "Remove" and "Delete" were used with the button "Add". This is another terminology inconsistency.

DIALOG BOX	Add	Remove		Delete		Cancel		Close	
	(H,W)	(H,W)	Rel. Pos.	(H,W)	Rel. Pos.	(H,W)	Rel. Pos.	(H,W)	Rel. Pos.
archive.cft	25,65	25,65	x, y+15			25,73	x-9,y+151		
autoff.cft	25,73	25,73	x, y+39					25,73	x, y+71
dp.cft	25,97	25,89	x+1,y					25,81	x+351,y
famdef.cft	25,89			25,89	x+1, y			25,97	x+87, y
standrd.cft	25,89			25,89	x+1, y			25,89	x+175,y

4.1.2 Evaluation of University of Maryland Interface

The 80 dialog box University of Maryland AT&T Teaching Theater Interface was a combination of different applications all designed for the students to use. Evaluation of this interface highlighted the intra-application inconsistencies that may exist among applications designed for the same user. A portion of the dialog box summary table is shown in figure 4.8.

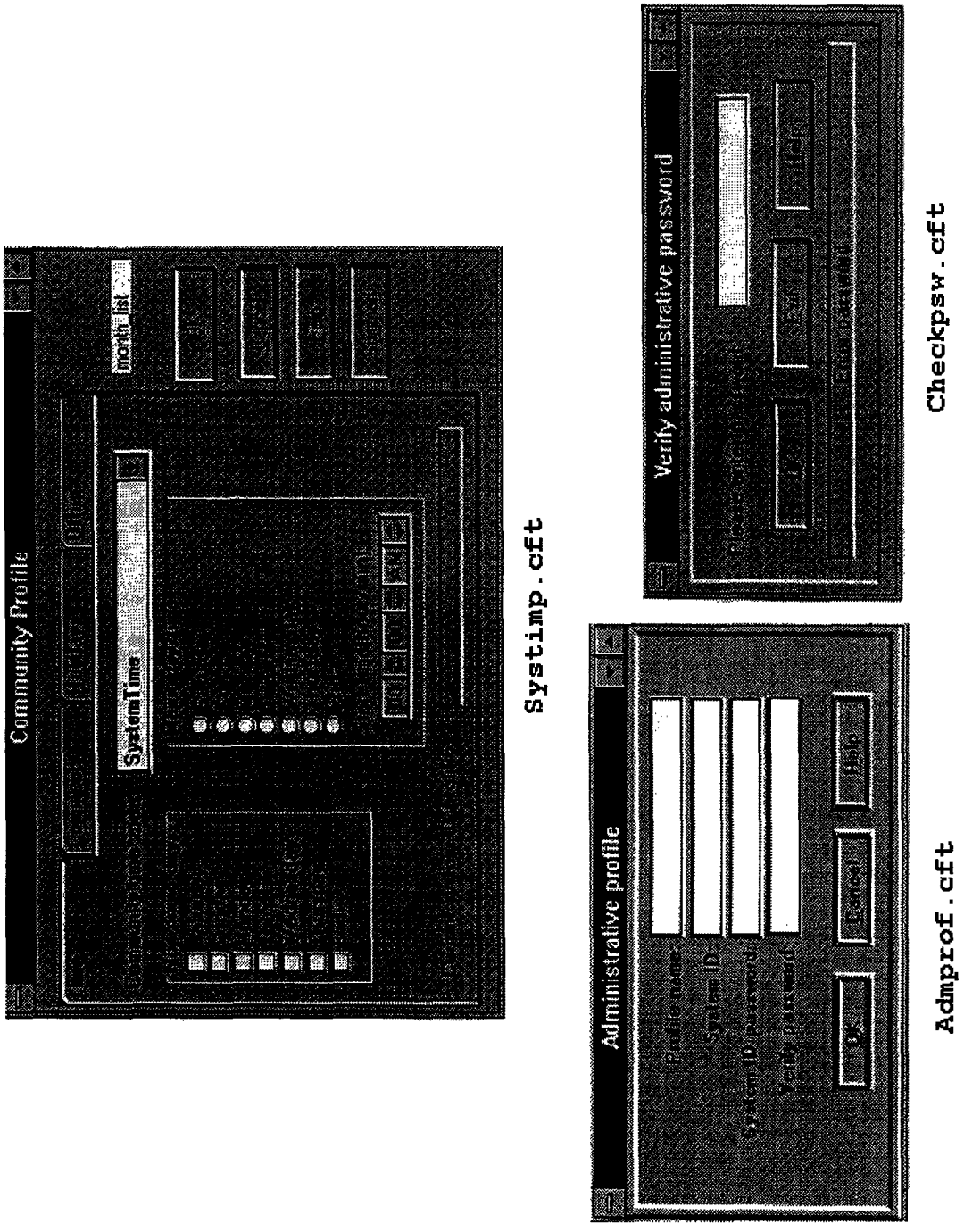


Figure 4.6: Screen Shots from Analysis of Button Layout Table

Dialog Box Summary Table Analysis

Aspect Ratio: Aspect Ratio, which is ratio of the height to width of a dialog box, in general varied between 0.5 and 0.8. There were a few dialog boxes whose Aspect Ratio were on the lower side, (left_or_right.cft, ratefrm2.cft and zoom.frm.cft) and a few had a high aspect ratio of 0.9 or more. (about1.frm.cft, about2.frm.cft, delete.frm.cft and notice.frm.cft) Also, the dialog boxes performing similar functions had different Aspect Ratios. All the *About, Cover and Exit* dialog boxes had different aspect ratios. Although most of these dialog boxes belong to a different application, these applications have been designed for the same set of users and these inconsistencies in Aspect Ratio, especially in the dialog boxes with the same functionality, should not exist.

Widget Totals: Some dialog boxes had a high value of widget totals i.e. 70 or more widgets. This may indicate complexity in the dialog box.

Non-Widget Area: High values of Non-Widget area (above 90%) were found in some of the dialog boxes including main.frm.cft, syllabus.frm.cft, vd-mdi3.frm.cft, winstat.frm.cft. This indicates that the use of screen space may not be optimum in the above cases. Low values of Non-Widget area make the screen appear cluttered. Dialog boxes filelist.frm.cft, winchat8.frm.cft and zoom.frm.cft had low Non-widget areas.

Widget Density: Widget Density is another measure of the crowdedness of widgets. High values of widget density (around 150 or more) in the dialog boxes like ibm_az.frm.cft, rate.frm.cft and seat_uaz.frm may indicate that too many widgets are present in a small area. Those dialog boxes which had high widget density, but had Non-widget area of 40% or more may be acceptable. Low values of Widget Density (less than 20) in the dialog boxes like frmcompaz.frm

and frmcompu.frm may indicate that the use of screen space is not optimum.

Margins Left margins varied from 0 to 192 pixels, although the most frequently used margin values were between 8 to 16 pixels. A quarter of the dialog boxes had left margin values of 0 pixels and a few had high values above 70 pixels (feed.frm.cft, frmcompaz.frm.cft, frmcompu.frm.cft, frmlogin.frm.cft and frmquesu.frm.cft). Right margins varied from 0 to 381 pixels. In some cases high values of the right margins were not a problem, like the case when the dialog box only had labels and buttons which are in general center aligned. The high values of right margin (above 50 pixels) in frmcompaz.frm.cft, frmcompu.frm.cft, frmlogin.frm.cft, frmquesaz.frm.cft, frmquesu.cft, passwd.frm.cft represent inconsistency. Top margin varied from 0 to 56 pixels and was more consistent than left and right margins, but still a consistent value of 8 to 16 pixels should be used as a guideline for all the margins in the design of the interface. Similarly, bottom margins were more consistent than left and right margins with values clustered between 8 and 30 pixels. Very high values of bottom margins (above 100 pixels) were mostly exceptional cases which are acceptable (Cases where the dialog box only had menu items). Figure 4.7 shows two dialog boxes from the application. The dialog box on the left has consistent margins and the one on the right has inconsistent margins. This figure shows, the effect of margins on the layout of the interface.

Gridedness: Most dialog boxes had well aligned widgets, with low X-Gridedness and Y-Gridedness values (1 or 2). Some dialog boxes which had high values of Gridedness (4 or more) like picture3.frm, ratefrm2.frm, seat_uaz.frm, topic.frm, tqmain.frm, winstat.frm required minor alignment changes. Most Button Gridedness values were low, indicating that buttons were properly aligned in rows or columns. A small number of dialog boxes like cover.frm, qmain3.frm, mulq.frm, omp.frm had higher values of Gridedness due to misalignment of buttons by a few pixels.

Area Balances: Dialog boxes like dynaset.frm, dyngrid.frm, form9.frm and winstat.frm had high balance values indicating that they may not have well balanced screens.

Distinct Typefaces: Although the dialog boxes analyzed in this application belong to various small applications, these applications are designed for the same set of users. Therefore, the number of typefaces used should be kept small. In total 19 distinct typefaces were used which is very high. This shows that different designers worked on different applications without following any guidelines. It is recommended that the applications should be modified to decrease the use of too many typefaces. Some of the dialog boxes that used four or more different typefaces are About.frm, cover.frm, coveraf.frm, coveruf.frm and frmlogo.frm.

Distinct Background Colors: In addition to using too many typefaces, this application uses eight distinct background colors. This may be attributed to the fact that different dialog boxes had different styles which means specific groups of designers worked on particular applications.

Distinct Foreground Colors: Altogether the application used 15 different colors (both background and foreground). The number of distinct foreground

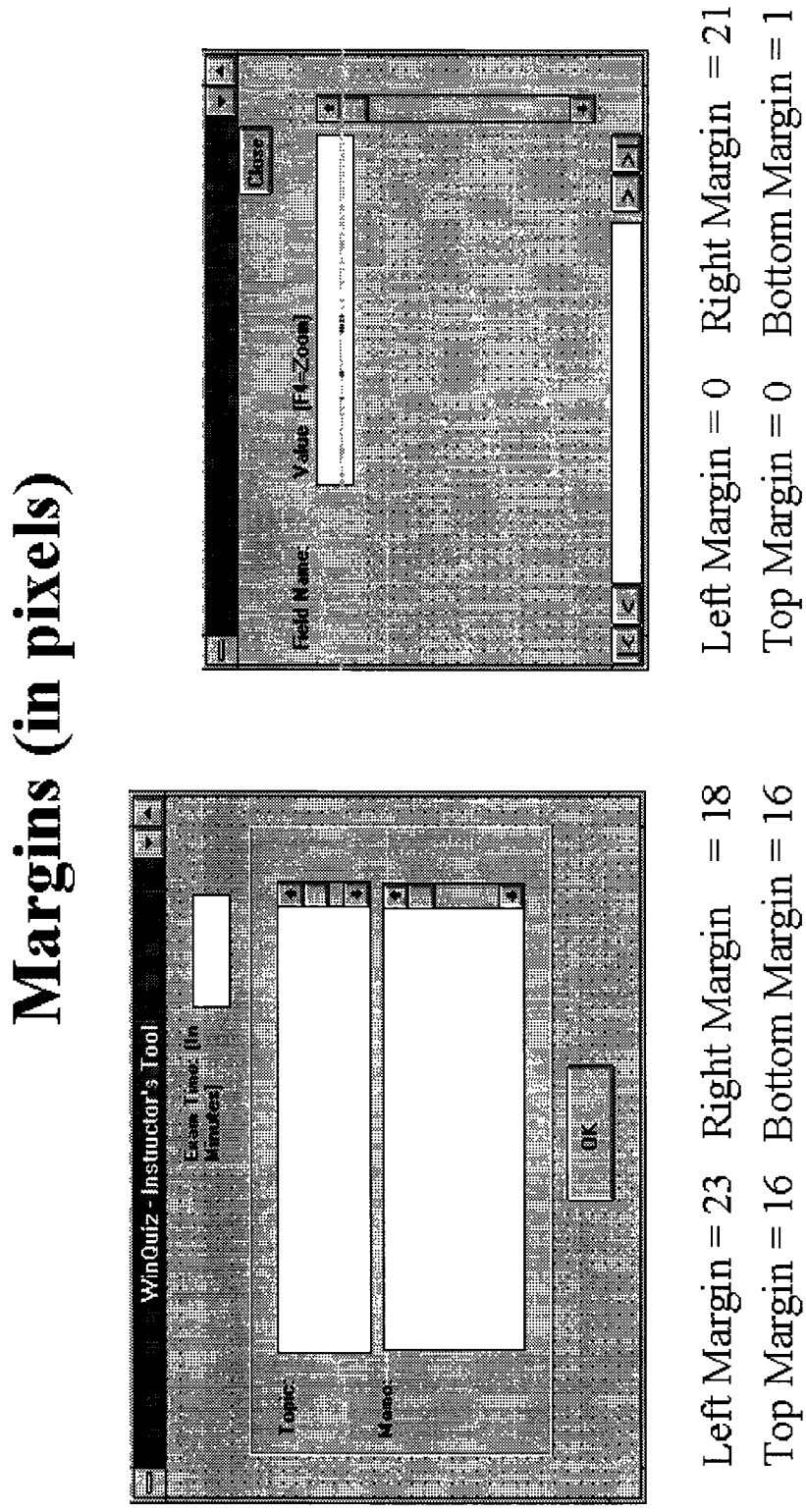


Figure 4.7: Margin Inconsistencies.

colors used was 10 which is high.

Interface Concordance:

There are a few terms which used different case across the application like Cancel, cancel, CANCEL, Delete, DELETE and more. Designers need to check whether these are inconsistencies or not.

Button Concordance:

The following are the inconsistencies detected by the Button Concordance Tool:

- Designers used six distinct typefaces in Button Labels which is inconsistent. A single typeface should be used for all button labels.
- Designer used three distinct foreground colors in button labels which may be an inconsistency. Like the typefaces, a single foreground color should be used for all the button labels.
- Button sizes are inconsistent across the application. For example, the height of the "Cancel" button varies between 24 and 49 pixels and width varies between 57 and 122 pixels. Other buttons that have inconsistencies in button sizes include "OK", "Done", "Exit", "No", "Previous" and "Start" and more.
- All buttons should use the same case across the application. Buttons like "OK", "Cancel", "Done", "Exit", " No" used different case across the application which is an inconsistency. Also, the designers used the button labels "Save Left" and "Save Right" in some dialog boxes and "Left Save" and "Right Save" in others which is an inconsistency. Designers also

used abbreviations in button labels like "Close w/o Changes" which is not recommended. Also the number of terms on button labels should be kept to a minimum.

- The left, right and top button positions from the dialog box helped in detecting inconsistencies in button placement. For example, the "Cancel" button had a different right button position for every dialog box, so the distance between the right end of the button and the right end of the dialog box is different for every dialog box. This doesn't represent a good layout. In the case of the "Close" button, the left position was 8 pixels in two dialog boxes and was 291 in the third, indicating that the "Close" button is left aligned in the first case and right aligned in the second case. Similar inconsistencies existed in buttons like "Done", "Exit", "OK" and more.

BUTTON LABEL	DIALOG BOX	BUTTON TYPEFACE	BUTTON FG_COLOR	BUTTON (H , W)	BUTTON POSITION LEFT	RIGHT	TOP
Exit	attapp94.cft	1	2	56,120	464	56	240
	cover.cft	2	2	57,153	680	182	536
	coveraf.cft	3	2	41,89	448	0	392
	coveruf.cft	3	2	41,89	448	85	392
	frmhand.cft	4	3	49,105	384	31	136
	frmlogin.cft	4	3	41,97	248	96	256
	winstat.cft	6	1	49,113	368	70	424
EXIT	delete.cft	1	1	41,97	280	45	352
	syllabus.cft	1	1	33,137	856	7	512
Left SAVE	feed.cft	5	2	33,81	272	647	448
Left Save	omp.cft	5	3	33,97	112	796	456
Right SAVE	feed.cft	5	2	33,89	648	263	448
Right Save	omp.cft	5	3	33,97	544	364	456
SAVE Left	mulq.cft	5	2	33,97	256	653	488
SAVE Right	mulq.cft	5	2	33,97	504	405	488

DISTINCT TYPEFACES IN BUTTONS:

1 = MS Sans Serif 8.25 Bold

2 = MS Sans Serif 18

3 = MS Sans Serif 13.5

4 = MS Sans Serif 12 Bold

5 = MS Sans Serif 9.75 Bold

6 = MS Serif 12 Bold

DISTINCT FOREGROUND COLORS IN BUTTONS:

1 = Default Color

2 = ffffffff80000005

3 = 0

4 = ff0000

No.	Dialog Name	Aspect Ratio (H/W)	-WIDGET-- TOTALS All	Top-Level	Non-Widget Area (%)	Widget Density widget/area	-----M A R G I N S------ Left	Right	Top	Bottom	----GRIDEDNESS----- Top Level X	Y	Buttons X	Y	-Balances-- Area Ratios Horiz (L/R)	Vert (T/B)	Distinct Typefaces	Distinct Background Colors	Distinct Foreground Colors
30	form9.cft	0 74	19	13	88.7	69	0	381	0	17	3	3	0	0	10.0	2 4	4 9	1 2 5 6	3 4 6
31	frmcompaz cft	0 79	5	4	67 5	11	104	97	56	69	1	2	0	1	1 0	1.4	9 13	2	3 4 7
32	frmcompu.cft	0.79	5	4	67 5	11	104	101	56	69	1	2	0	1	1 0	1.4	9 13	2	3 4 7
33	frmhand cft	0 48	6	5	67 5	33	32	31	32	31	1	2	0	1	1 0	1 5	9 13	2	3 4 7
34	frmlogin cft	0 85	8	7	77 1	25	96	96	40	64	1	1	0	1	0 6	1.7	9 13	2	3 4 7
35	frmlogo.cft	0.38	9	8	18 3	25	0	223	0	0	2	2	1	1	0 8	1 3	4 9 16 17	1 2 10	2 3 11
36	frmmatch.cft	0.50	7	6	70.0	60	16	50	24	43	1	1	0	1	0 8	1.3	4	2	3 4
37	frmquesaz.cft	0.42	6	5	75 6	25	56	54	48	61	0	1	0	1	1.1	1.5	9 13	2	3 4 7
38	frmquesu.cft	0.45	6	5	73.9	23	72	14	48	74	0	1	0	1	0 8	1.3	9 13	2	3 4 7
39	graph.cft	0.55	14	4	57.0	22	0	0	0	7	2	3	0	1	1 0	1.0	7 8	2 5 8	2 3 7
40	grid3.cft	0.89	5	4	42.5	23	21	15	13	60	2	1	1	0	1 5	1.2	4	2	3
Maximum		1.60	102	101	100.0	184	192	381	56	276	9	13	2	3	10.0	10.0			
Minimum		0.13	0	0	0 0	0	0	0	0	0	0	0	0	0	0.0	0.0			
Average		0.73	14.	9	57.5	48	19	52	11	29	2	2	0	0	1.8	1.6			

DISTINCT TYPEFACES:

1 = Arial 13.5 Bold
2 = Symbol 9.75 Bold
3 = Arial 8.25 Bold
4 = MS Sans Serif 8.25 Bold
5 = System 9.75 Bold
6 = Arial 15.75 Bold
7 = MS Sans Serif 9.75 Bold
8 = MS Sans Serif 16.5 Bold
9 = MS Sans Serif 12 Bold
10 = MS Sans Serif 13.5
11 = Symbol 13 5 Bold
12 = MS Sans Serif 24 Bold Italic
13 = MS Sans Serif 13.5 Bold
14 = MS Sans Serif 18
15 = MS Serif 30 Bold
16 = Arial 18 Bold
17 = Symbol 8.25 Bold
18 = Times New Roman 24 Bold Italic
19 = Times New Roman 30 Bold Italic

DISTINCT BACKGROUND COLORS:

1 = ffffff
2 = ffffffff80000005
5 = c0c0c0
6 = ff
8 = e0ffff
10 = c00000
12 = 404040
14 = ffffffff8000000f

DISTINCT FOREGROUND COLORS:

2 = ffffffff80000005
3 = ffffffff80000008
4 = 0
6 = ff
7 = ff0000
9 = c000c0
10 = c00000
11 = ffff
13 = 808080
15 = c000

Figure 4.8: Portion of the Dialog Box Summary Table

Interface Speller:

The spell checking tool detected various abbreviations and a few misspelled terms. The use of abbreviation should be kept to a minimum in interface design. The following were the spelling errors detected: "qiz", "veryfying", "peronal" and "btrieve". All the other terms in the output except abbreviations were valid terms, but not found in the dictionary.

Terminology Baskets:

The output from the basket [Browse, Display, Find, Retrieve, Search, Select, Show, View] shows that "Display", "View" and "Show" have all been used in this application. Also, both "Find" and "Search" have been used. Similarly the output from the basket [Cancel, Clear, Delete, Purge, Refresh, Remove] indicates that the terms "Cancel", "Delete", "Clear", "Refresh" and "Remove" were all used in the application. Designers need to check whether these are inconsistencies.

Button Layout Table:

The Button Layout Table revealed inconsistencies in button sizes and placement within a dialog box and across the interface. For example, the button set [OK Cancel ... Exit Help] revealed inconsistencies in the sizes of the "OK" "Cancel" and "Help" buttons. Also,the"Cancel" and "Help" buttons were in some cases placed next to OK buttons in a row or at other times stacked below the "OK" button in a column, with the distance between these buttons varying from 0 to 40 pixels. Figure 4.9 shows the dialog boxes in which button placement inconsistencies of "OK" and "Cancel" buttons were detected by the Button

Layout Table and Button Concordance Tool.

4.1.3 Evaluation of Italian Business Application

This application had 75 dialog boxes and is a prototype for a business client of of an Italian company. The terminology used in this application is in Italian and the analysis was restricted to tools that do not involve the use of a dictionary or the evaluation of terminology.

Dialog Box Summary Table

Aspect Ratio: Aspect Ratio was mostly consistent across the interface with values between 0.5 and 0.7. There were a few dialog boxes which had an aspect ratio on the lower side, around 0.4 (dataidiot.frm, db_selal.frm, il.frm, riepsel.frm, seleff.frm and ultsel.frm). Also, a few dialog boxes had high values of aspect ratio, around 0.9, (sedi.frm, selanoma.frm, sel.frm, st_pun.frm) which may be reconsidered.

Widget Totals: Some dialog boxes had high values of widget totals, around 100, which may indicate too much complexity. (econ.frm, statg.frm, histab.frm)

Non-Widget Area: High values of Non-Widget area (around 90%) were found in the dialog boxes like anag.frm, conn.frm, correl.frm and db_selal.frm. This may indicate that screen space was wasted in those dialog boxes. Dialog boxes like graf1.frm, graf1b.frm, outcorr.frm and statg.frm had Non-Widget area less than 20% implying that more space between widgets should be created for better screen design.

Widget Density: Dialog boxes like seleffarm.frm, st_punn.frm and statg.frm had high values of Widget Density (> 150). This may indicate that too many

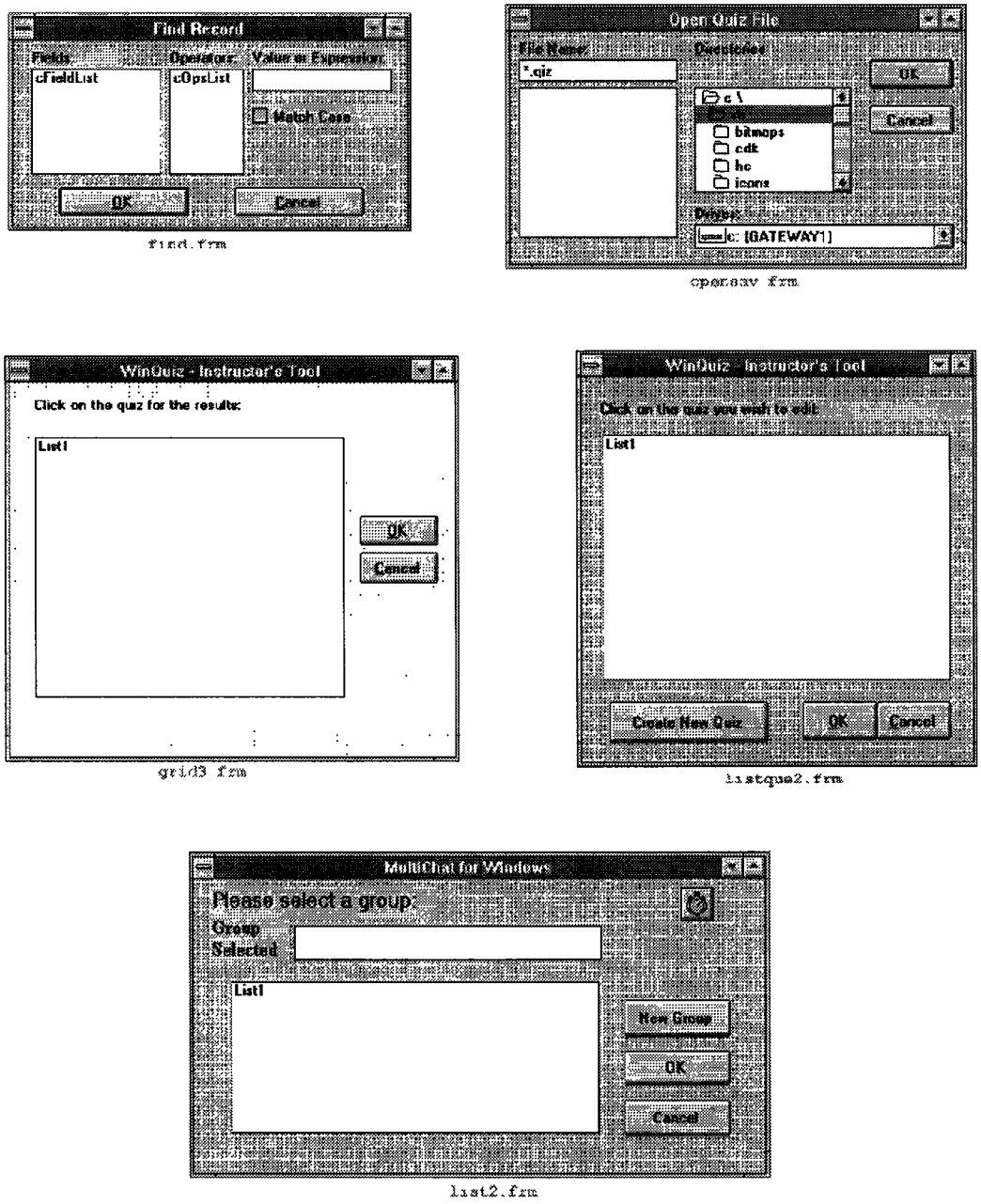


Figure 4.9: Button placement inconsistencies in OK and Cancel buttons.

widgets are present in a small area.

Margins: Left margin varied from 0 to 60 pixels, although most of the dialog boxes had a left margin value around 20 pixels. Dialog boxes like conn.frm, il.frm, infiva.frm had left margin values greater than 35 pixels which is inconsistent. Right margin values varied from 0 to 284 pixels. Many dialog boxes had right margin values around 20 pixels, but dialog boxes like atteast.frm, attiv.frm, riepsel2.frm and ultsel.frm had right margin values around 100 pixels or more which is inconsistent. Top margin varied from 0 to 36 pixels and was mostly consistent around 16 pixels. Similarly, bottom margins were mostly consistent between 16 and 20 pixels, but some dialog boxes like conn.frm had higher values around 45 pixels.

Gridedness: High values of X-Gridedness and Y-Gridedness in all the *graf dialog boxes* indicate that widgets in those dialog boxes were not properly aligned. No misaligned buttons were detected by the Button Gridedness metric.

Distinct Typefaces: This application used too many different typefaces. The number of typefaces should be small, ideally between three and five. In this case 15 distinct typefaces were used. Designers should check the dialog boxes that used too many typefaces like diag.frm, econ.frm, f770.frm, raffan.frm, statg.frm and many more.

Distinct Background Colors: This application used 21 distinct background colors, indicating the use of too much color in screen design. Dialog boxes like outcorr.frm, outsel.frm, histab.frm, and contr.frm used six or seven background colors. Designers should check whether the use of these many different colors is appropriate for the task. After looking through the output more carefully, it seemed that certain sets of dialog boxes had same background colors

which were entirely different from other dialog boxes, leading to the conclusion that different designers worked on different parts of the application.

Distinct Foreground Colors: As mentioned earlier, the use of typeface and color is very extensive in this application. There were 18 distinct foreground colors used in this application.

Concordance Tool

The concordance tool output all the terms used in the application. Because, the terms were in Italian, no analysis was done. Designers should go through all the terms to check the proper use of terminology.

Interface Concordance

There are many terms that used different case in dialog boxes across the application, for example terms like codice, CODICE, Codice or Provincia, PROVINCIA, provinciale and Provinciale belonged to this set of terms. Since, the outputs were in Italian, designers need to check which of these represent inconsistency, if any.

Button Concordance

Consistencies

- Designers used the same typeface MS Sans-Serif 8.25 Bold for all the button labels which shows that some consistency guidelines were followed.
- There is no variant capitalization in button labels.

Inconsistencies

- Most button labels used the same foreground color listed as color 1 for buttons, but there are few which used different foreground color, listed as color 2.

- Some buttons like "MENU" had the same height and width across the application, but most of the buttons had different height and width in different dialog boxes. Taking an example, the height of the "OK" button varied between 25 and 29 pixels and the width varied between 53 and 85 pixels.

- There were many button placement inconsistencies across the interface. For example, in all the dialog boxes, the left position of the "ANNULA" button varied between 200 and 300 pixels and right position between 150 and 200 pixels, but for the dialog box outtcorr.frm, the left position was 552 pixels and right position was 8 pixels. The same is true with the "OK" button which in general had a left position between 100 and 200 pixels (buttons were centered), but in dialog boxes like corr.frm, db_stats.frm, grafgrid.frm, id.frm, riepsel.frm, and sel.frm the "OK" buttons had a left position of 16-40 pixels (buttons were left aligned). Additionally, the dialog boxes like graf1b.frm, graf2b.frm, graf3b.frm, raffani.frm and ross.frm had a left position of the "OK" button around 550 pixels and a right position around 10 pixels (buttons were right aligned). These button positioning inconsistencies also existed in other button labels and can be noticed at a glance.

Button Layout Table

Since the outputs were in Italian, the default button sets didn't work in this application. The output from the easily identifiable button set [OK Cancella Annula Uscita] corresponding to [OK Cancel .. Help] was analyzed.

Consistencies

- Button sizes were consistent within the same dialog box.
- Sizes of Cancella and Annula were mostly consistent across the interface
- Buttons were stacked horizontally in a row at the bottom of the dialog box throughout the application which is consistent.

Inconsistencies

- The size of "OK" button was different in various dialog boxes across the application.
- Both "Cancella" and "Annula" buttons were used with "OK" button which may be an inconsistency.
- Inconsistencies existed between relative button positions. For example, the "Annula" button in most cases had a relative position between x + 19, and x + 50 pixels from the OK button (horizontal distance of 19 to 50 pixels between the two buttons). But in the case of the dialog box riepsel.frm, the relative position was x + 180 pixels(horizontal distance of 180 pixels between the two buttons) which was an inconsistency.

4.2 Conclusion

Evaluation of the four applications using *SHERLOCK* tools helped us to determine those tools which were most successful in detecting inconsistencies and those that were less successful. The dialog box summary table had limited success in detecting inconsistencies. Only certain metrics of the dialog box summary table like aspect ratio, margins, distinct typefaces, distinct foreground and background colors were successful in finding inconsistencies. Most of the extreme values computed by the metrics like Non-Widget Area, Widget Density and Area Balances were due to the limitations of *SHERLOCK* or the Visual Basic development tool and were not real inconsistencies.

These metrics were modified several times to deal with exceptions and further work is required to validate these metrics. The Button Concordance and the Button Layout Table proved to be the most useful tools and were able to detect inconsistencies in the size, position, typeface, color and terminology used in buttons. The Interface Concordance and the Interface Speller tools were successful in detecting terminology inconsistencies like variant capitalization, abbreviations and spelling errors. The Terminology Basket tool helped in detecting misleading synonym terms in many cases. In summary *SHERLOCK*, was successful in detecting major terminology inconsistencies and certain inconsistencies in visual design of the evaluated interfaces.

4.3 Limitations of *SHERLOCK*

SHERLOCK evaluation tools were designed to aid the interface evaluation process by providing a compact overview of possible inconsistencies and anomalies

in certain visual design characteristics and textual properties of the interface. The designer must decide what to do, if anything about these possible inconsistencies. Certain issues like efficiency in screen layout including proper placement of widgets on the dialog box, violations of design constraints, and the use of inappropriate widgets types are not evaluated by *SHERLOCK*. Other evaluation methods, such as usability testing and heuristic evaluation, are needed to locate typical user interface design problems such as inappropriate metaphors, missing functionality, chaotic screen layouts, unexpected sequencing of screens, misleading menus, excessive demands on short-term memory, poor error messages, or inadequate help screens.

Chapter 5

Feedback From Designers

Output from the *SHERLOCK* tools and the screen shots of the interface along with the analyses were forwarded to the developers and designers to elicit feedback.

5.1 GE Interfaces

We worked closely with the people at GE Information Services to get feedback on the effectiveness of *SHERLOCK* tools, as these tools were being iteratively refined. The feedback was positive on the evaluation output with suggestions for modifications at every refinement stage.

The feedback suggested that the outputs of the dialog box summary table were simple for the designers to interpret, as they were able to detect inconsistencies by scanning down the columns for extreme values, indicated by the statistical analysis at the bottom of the table. They recommended that we develop some "goodness" measures for the metrics after analyzing more applications. We have succeeded partly in assigning measures to certain metrics after

analyzing the four applications. The detection of the use of multiple typefaces and colors by *SHERLOCK* is one of the inconsistencies they otherwise would have missed. Inconsistent margins was another dimension that would have been slipped through testing, if not detected by *SHERLOCK*. They plan to incorporate the proper usage of fonts, colors and margins in their future guidelines. Although, most of the dialog boxes in the GE interface neither had a cluttered or crowded layout, Non-Widget Area and Widget Density were two of the metrics which the GE designers agreed were important in determining the layout of the interface.

The incorporation of a spell checking tool in *SHERLOCK* had a positive response from the designers, since none of the current GUI building environments on the PCs have a built in spell checker. Separate detailed analysis of each metric of the dialog box summary table was recommended by the designers for the future implementations. One of the steps taken in this direction was the development of the Margin Analyzer tool to indicate inconsistencies in margins and the way to rectify those inconsistencies. Another quick evaluation tool that had positive results in detecting inconsistencies was the button concordance tool. The extreme variations in button sizes detected by this tool that included instances with no two same label buttons having the same size across the application, raised concerns within GE design team. The button typeface, size and placement inconsistencies detected by the Button Concordance tool were corrected by the GE designers after reviewing the evaluations performed by *SHERLOCK*.

Inconsistencies in relative positioning of button labels and button terminology detected by the Button Layout table were modified by the designers using the output. Use of misleading terminology was another dimension explored by

the terminology basket tool which helped GE designers in rectifying a few terminology inconsistencies which would have been missed otherwise. Controlling the use of potentially misleading synonym terms in the user interface specifically in buttons, was something they wanted to do in the near future. Overall the use of *SHERLOCK* helped to modify the layout, visibility and terminology of GE interfaces by detecting many small inconsistencies. The identification of these inconsistencies led to reconsideration of design issues for current and future implementation and will lead to interfaces with improved "look and feel".

5.2 University of Maryland Interface

Since this application was a combination of various small applications, the output was given to two different design groups to elicit a broader spectrum of feedback. The first group included the developers of a portion of the interface and the other was the designers responsible for all the applications together.

Developers' feedback on the dialog box summary table was positive for some metrics. They showed interest in the ability of the dialog box summary table to detect the typeface and color inconsistencies in their application. When asked for a possible explanation of these inconsistencies, they explained that different designers worked on different portions of the application, with very few guidelines on visual design. Similar reasons were given for other inconsistencies including different aspect ratio's for functionally similar screens and the use of inconsistent margins.

Designers liked the statistical analysis at the end of the metric table with mean, maximum and minimum values and wanted an additional function that

listed the optimum values for the metrics. They liked the idea of the spell checking tool so much that they wanted this tool for their future development projects. Many of the terminology inconsistencies detected by the Button Layout Table and the Terminology Basket tool were valid inconsistencies which they will take into consideration for the next version of the application.

Chapter 6

Software Design

6.1 Software System Architecture

SHERLOCK is a set of 7 user interface consistency checking programs which were implemented in about 7000 lines of C++ code, and developed on the SUN SPARC Stations/UNIX platform. In order to evaluate a Graphical User Interface using *SHERLOCK*, its interface description files need to be converted to a canoncial format [see section 3.1]. These canonical format files are the only input required by the *SHERLOCK* evaluation tools. Each of the *SHERLOCK* evaluation tools provide separate output for the designers to review, with a total of 7 outputs. Figure 6.1 shows a top level design view of the system. *SHERLOCK* was designed to be a generic GUI consistency and evaluation tool.

6.1.1 Translator and Canonical Format Design

Translator programs are specifically designed for a particular GUI development tool and converts its interface description(resource) file to a canonical format.

The canonical format is a sequence of GUI object descriptions in the form of attribute value pairs enclosed in curly braces. Design of the data structure for the translator depends on the format of the interface resource file. Two translators were created, one for Visual Basic 3.0 and the other for Visual C++ 4.0 using a lexical scanner generated by FLEX (Fast Lexical Analyzer Generator) which is a tool for generating programs that perform pattern matching on text. Using the lexical scanner, attribute value strings are detected and converted to the appropriate format as defined by the canonical format. All the dimensional coordinates are converted to pixels and other platform and application dependent values. [Section 3.1] All the parent child relationships of the widgets are kept intact by the translator and also each object is provided with a parent-ID attribute and unique-ID to help in processing data read by *SHERLOCK* tools. Every object is enclosed by curly braces and is indented appropriately to identify parent child relationships. Figure 6.2 shows a portion of a Visual Basic .frm file and its corresponding canonical format.

The translator program only extracts specific interface description information from the resource file. If any attributes corresponding to this specific information do not exist in a given object, they are not listed. For example, the menu objects do not have any attributes like height, width, left and top, therefore these attributes are ignored instead of listing null values. The attributes with null values are not listed to facilitate the data processing of *SHERLOCK* tools. Some GUI development tools like Visual C++ store all the interface description information in a single file. Others like Visual Basic store interface description of each form in a separate file. *SHERLOCK* requires a separate canonical format file for every form, so the translator programs may need to perform additional

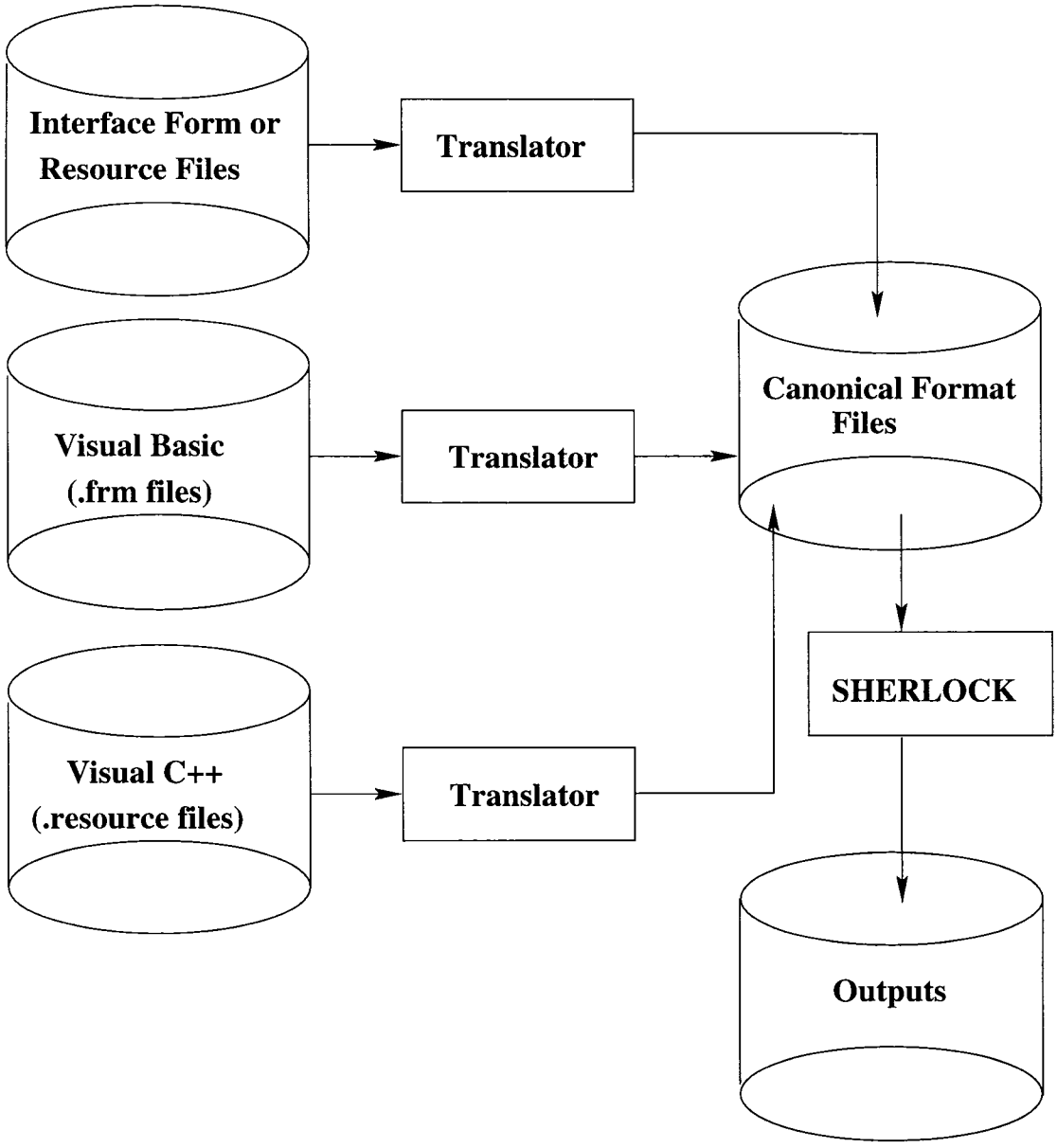


Figure 6.1: Top Level Design View

Visual Basic .frm file

```
Begin Form frmHelp
BackColor          =  &H00C0C0C0&
BorderStyle        =  1  'Fixed Single
Caption            =  "MultiChat Help"
ClientHeight       =  5460
ClientLeft         =  3120
ClentTop           =  3315
ClientWidth        =  5085
ForeColor          =  &H00808080&
Height             =  5865
Left               =  3060
LinkTopic          =  "Form1"
MaxButton          =  0  'False
MinButton          =  0  'False
ScaleHeight        =  5460
ScaleWidth         =  5085
Top                =  2970
Width              =  5205
Begin CommandButton Command1
BackColor          =  &H00404040&
Caption            =  "&OK"
Height             =   495
Left               =  2040
TableIndex         =  1
Top                =  4800
Width              =  1095
End

Begin TextBox  Text1
Height             =  4455
Left               =  120
MultiLine          =  -1  'True
TabIndex           =  0
Top                =  120
Width              =  4815
End
End
Sub Command1_Click()
   Unload frmHelp
End Sub
Sub Form_Load ()
   Call HelpFile
End Sub
Sub Text1_Change ()
      Call HelpFile
End Sub
```

Canonical Format file

```
{
   type                      window
   parent                    NULL
   resource-id               window-1
   name                      frmHelp
   label                     "MultiChat Help"
   unit-of-measure-width UNIT_PIXEL
   unit-of-measure-height UNIT_PIXEL
   top                       198
   left                      204
   height                    391
   width                     347
   client-width              339
   client-height             364
   foreground-color          808080
   background-color          c0c0c0
   {
      type                   edit-box
      parent                 window-1
      resource-id            edit-box-1
      name                   Text1
      top                    8
      left                   8
      height                 297
      width                  321
      foreground-color       ffffffff80000008
      background-color       ffffffff80000005
      font-name              "MS Sans Senf"
      font-size              8.25
      font-bold              True
   }
   {
      type                   button
      parent                 window-1
      resource-id            button-1
      name                   Command1
      label                  "OK"
      top                    320
      left                   136
      height                 33
      width                  73
      background-color       404040
      font-name              "MS Sans Senf"
      font-size              8.25
      font-bold              True
      mnemonic               0
   }
}
```

Figure 6.2: Conversion of Interface Description file to Canonical Format File

file parsing to fulfill this requirement.

6.1.2 *SHERLOCK* Design

The family of consistency checking tools was implemented using different sets of classes in the C++ programming language. The data structure for these tools was designed in accordance with the canonical format input files. The *SHERLOCK* data structure was designed to be flexible, extensible and customizable to changes that may be made by expansion of the canonical format files. Although data in canonical format is stored in attribute value pairs, GUI development tools (Visual Basic, Visual C++) may exclude some of these attributes in their resource files or may include additional attributes. *SHERLOCK* is designed to handle these possibilities.

SHERLOCK has a sequential modular design and can be divided into the following subsystems.

Widget Store Subsystem

The modules of this subsystem parse and store the widget description information from all the canonical format files and create a tree structure on top of the stored information. Since every widget of the dialog box has a different size (number of attributes value pairs), an object "Widget" was created to store these attribute value pairs as "Str" a string object, so that memory can be allocated dynamically according to the size of attribute value pairs and according to the size of the widget.

```
class str {
private:
    struct srep {
        char*   s; // pointer to data
```

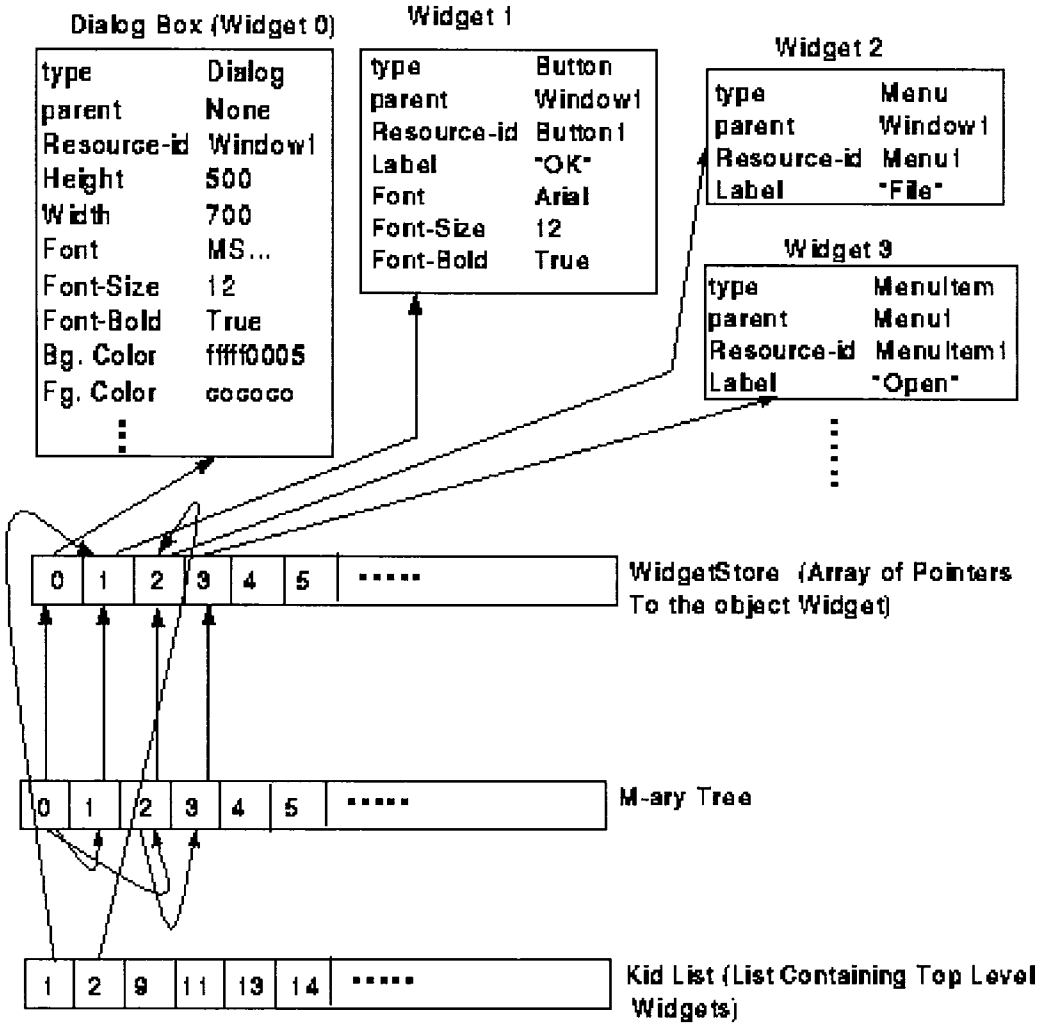


Figure 6.3: Data Structure View

```
        int     n; // reference count
        srep() {n = 1;}
    };
    srep *p;
public:
    char* GetStr();//To return the string
    str (const char*);  // constructor for input format: string x = ``abc''
    str();              //  constructor string x
    ~str();//destructor
    str (const str&);// constructor for input format: string x = string...
    str& operator= (const char*);//overloaded operator =
    str& operator= (const str &);//overloaded operator =
    void release_memory();//destructor for special cases
    char& operator[] (int i);//overloaded operator[]
    const char& operator[] (int i) const;//overloaded operator[]
                .
                .
                .
    friend ostream& operator<< (ostream&, const str&);
    friend istream& operator>> (istream&, str&);
    friend int operator== (const str &x, const char* s);
    friend int operator== (const str &x, const str &y);
    friend int operator!= (const str &x, char* s);
    friend int operator!= (const str &x, const str &y);
}

class widget {
 // A widget class object.
private:
    struct pair {
    str attribute, value;
    };
    pair *vec;
    int max, count;
    widget(const widget&);   // prevent copying
    widget& operator= (const widget&); // prevent copying

public:
    widget(int);//constructor
    ~widget();//destructor
    str&        operator[] (const str);
    const str&  operator[] (const str) const;
    const str&  operator[] (const char*) const;
    friend      ostream& operator<< (ostream&, const widget&);
                .
                .
                .

}; //end class widget
```

The "Widget" objects are stored using "Widget Store" an array of pointers to the object "Widget" [fig. 6.3]. The String Class, Widget Class and all the other Classes incorporated in *SHERLOCK* use operator overloading to simplify the processing of strings and other abstract data types. "WidgetStore" stores the input widget data in the format it is input from the canonical format files. A high level ordering mechanism of M-ary tree and a Kid list (list containing all the top-level widgets of the dialog box) is built on top of the "WidgetStore". The M-ary tree and the Kid list are constructed as pointers to the "WidgetStore".

All the *SHERLOCK* tools share the same data structure programs for parsing and storing the widget data. In addition to the above set of class objects, every *SHERLOCK* program had class objects specific to the data being processed. This design methodology was followed to make *SHERLOCK* flexible and extensible, so that more tools can be constructed on top of the initial set of programs.

Dialog Box Subsystem

This subsystem is uniquely defined for dialog box summary table and encapsulates various functions which define metrics of dialog box summary table and those used for statistical analysis. These functions access the "Widget Store" using both the M-ary Tree and the Kid List. Kid List is used for those calculations where the top level widgets are only required. For example, the Kid List is used with Widget Density metric. Additional string processing functions are used to process the typeface, background color and foreground color information. Therefore the modules are linked to "String Processing Subsystem"

String Processing Subsystem

All the *SHERLOCK* programs share the string mapping subsystem since the "Widget Store Subsystem" stores all the widget information as string data type. It incorporates a module that defines various string manipulation functions. Also the subsystem incorporates string search and a dictionary building module. This module uses various templates like a Stack template and a Map template. Each dictionary entry is a tuple <string, list>. The list is a generic list and can be a list of filenames (list of indexes to array containing the filenames) or a list of strings. For example, this module is used by the Interface Concordance tool to search string labels with variant capitalization and count the number of occurrences and lists all the dialog boxes (filenames) in which those strings were found. It has a similar usage in the Terminology Basket tool. The Button Concordance tool inherits this module and builds a class with additional functionality related to button typeface, color and position on top of it.

Spell Checker Subsystem

Spell Checker subsystem module uses built in UNIX system libraries and tools to perform the spell checking operation and is linked to module used to build the string dictionary to store those results.

Button Processing Subsystem

This subsystem defines hierarchy of classes varying from button "Attribute Class" which defines all the attributes of buttons to the "Button" class which incorporates functions to calculate the button location with respect to the dialog box and button position relative to other buttons in the dialog box. A partial

set of classes of this system are defined below:

```
class AttributeClass
{
        friend class Buttons;
        str BLabel;// BLabel is of type string class
        struct attributeset {
                char * RelativePosition;
                char * file_name;
                int Bheight, Bwidth, Btop, Bleft;
                bool presence;
         };// A structure containing height, width, top, left coordinates of
              the button plus its relative position to the first button.
        attributeset * buttonfamily;
        int file_count,max;
        public:
        .
        .
};

class Buttons {
        struct  set {
                        AttributeClass * ButtonsArray;
                        int Button_count;
        };
        set * ButtonList;
        int Max, Count;
        public:
        Buttons (int);//Constructor
        ~Buttons ();//Destructor
        void NewButtonList();//Creating Button List
        void ReadButtons(const char*);//Reads Buttons in the Button list
        int  ButtonExistence(const str&, char *);//Checks for existence of
                                                  button in the list.
        void FindButtonPositions(widget*, char *);//Calculates the button size
                                                  //and position
        void ButtonInfo(int ButtonIndex,int tempheight, int tempwidth ,int
                        templeft, int temptop, int location);//Reads button info
        void FindRelativePosition();//Finds Relative Position of the Button
        .
        .
};//class Buttons
```

Chapter 7

Effects of Terminology Inconsistencies on User's Performance and Subjective Satisfaction

7.1 Introduction

An experiment was designed to assess the effects of inconsistent terminology on user's performance and subjective satisfaction. The experiment considered only one aspect of inconsistency, misleading synonyms. It was conducted to test the hypothesis that terminology inconsistencies reduce performance speed and subjective satisfaction. We developed a GUI in Visual Basic for the students to access the resources of the Career Center at the University of Maryland. Three versions of the interface were created, the first one with no terminology incon-

sistency. The second version had a medium level of terminology inconsistency (on average, one inconsistency in terminology was introduced for each task and each task had an average of four screens). The third version had a high level of terminology inconsistency (50% more inconsistent terms than the medium inconsistent version). The resulting 2 X 3 experiment had two independent variables which were level of expertise and the type of the interface (no inconsistency, medium level of inconsistency and high level of inconsistency). The level of expertise were no prior training and five minutes of training. For all the six phases of the experiment, users were given the same task list and their task completion time and subjective satisfaction was evaluated. For each treatment 10 subjects were selected, with a total of 60 subjects for the whole experiment. The results showed that the user's performance is significantly affected by terminology inconsistencies.

7.2 Interface Design

All the screens of the Career Center interfaces had a consistent visual design. Only terminology inconsistencies were introduced in the medium and high inconsistency version. Sample screen shots are shown in figure 7.1.

- The no inconsistency version had consistent terminology between labels, menus and titles across the interface. For example button labels "OK" and "Abort" were used consistently across the interface.

- In the medium inconsistency version, on average one terminological inconsistency was introduced for every task. Since the subjects were told to perform seven tasks, the interface had seven terminology inconsistencies.

These inconsistencies included changing the heading of the dialog box from "Questions" to "Inquiries" or

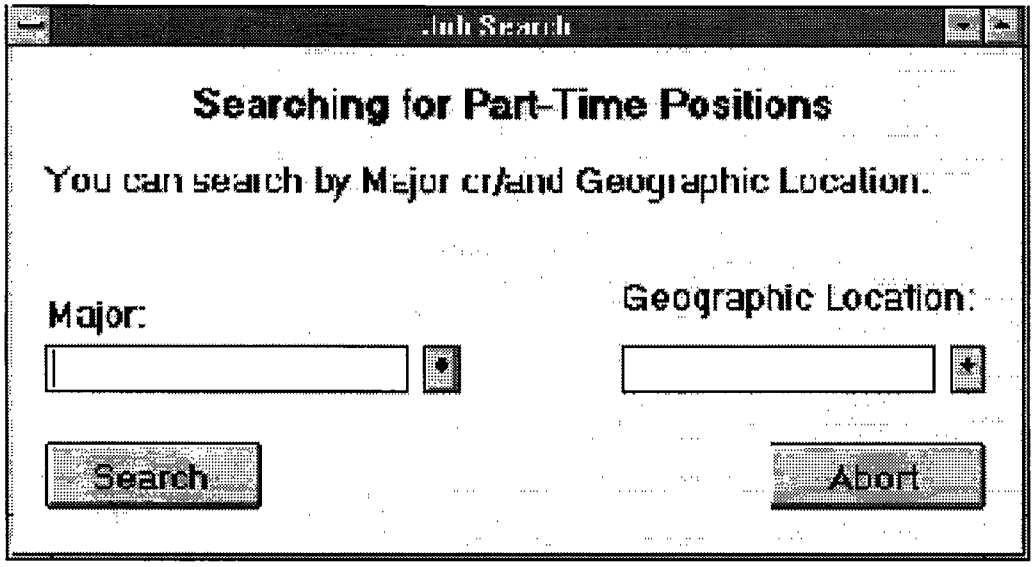
changing the widget labels from "Workshops" to "Seminars". Also, menu items were changed from "Career Counseling" to "Career Advising" and "View" to "List". Inconsistency in button labels were also introduced by changing "OK" and "Abort" to "Forward" and "Discard" in the case of a particular task.

- In the high inconsistency version, 50% more terminology inconsistency was introduced than the medium inconsistent version. On average, the high inconsistency version had one or two terminology inconsistencies per task with a total of 11 terminology inconsistencies. These inconsistencies included changing the "OK" button to "Done", "Return" or "Forward" depending upon the task being performed and the "Abort" button to "Discard" ,"End" or "Suspend", and "View Interviews" to "Interviews". Inconsistencies in menu items were introduced by changing "Workshops" to "Seminars" plus inconsistencies in widget labels were introduced by changing "Major" to "Area".

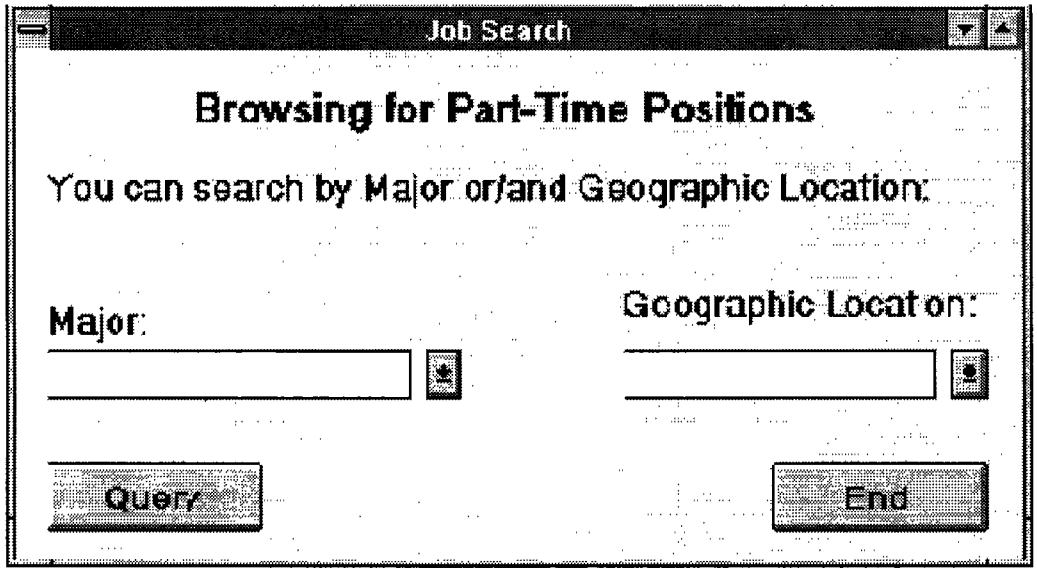
7.3 Hypothesis

In this experiment we tested two dependent variables which were task completion time and subjective satisfaction. The hypothesis were:

- The task completion time for the interface with no terminology inconsistency will be significantly lower than the interfaces with medium and high



No Terminology Inconsistency



High Terminology Inconsistency

Figure 7.1: Sample Screen Shots of the Experimental Interface

levels of terminology inconsistency in the case of subjects with no prior training of the system.

- The difference in task completion time for the interface with no, medium and high levels of terminology inconsistencies would decrease when subjects are given five minutes of training with the system prior to the execution of the tasks. the execution of the tasks.

- The subjective satisfaction will be significantly higher for the interface with no terminology inconsistency as compared to those with medium and high levels of terminology inconsistency.

7.4 Subjects

A total of 60 subjects were used in the experiment out of which 30 were given 5 minutes training on the no inconsistency interface. The subjects chosen were students at the University of Maryland and were frequent computer users and were comfortable using the mouse and Windows 3.1 operating system. To minimize the impact of typing on the experiment, subjects with a typing speed of 30 words per minute or more were chosen. Most of the subjects were computer science and electrical engineering undergraduates and graduate students.

7.5 Materials

The experimental interfaces were created in Microsoft Visual Basic 3.0 and the experiment was run on a 100MHz Pentium machine with a 17" color monitor having a 1024 X 768 pixel resolution with 256 colors. In order to ensure vol-

untary participation of subjects, all the subjects were asked to sign a consent form. Subjects were screened to meet certain requirements listed in the previous section. A set of instructions was provided in writing and was also explained to the subjects verbally before beginning the experiment. The subjects were asked to fill out a Subjective Satisfaction Questionnaire after completing the experiment. The questionnaire used in this experiment was a modified portion of the standard Questionnaire for User Interface Satisfaction (QUIS) [7]. The subjects performed the seven tasks listed below.

Task List

- Register for Workshop II
- Set appointment with any one of the career counselor for Nov. 8 for 10:00 to 10:30 am.
- View Part-time job openings in Computer Science Major in Maryland state.
- Submit the following question to the Career Center:
  Do counselors at the career center perform mock interviews ?
- Request Graduate School information for Masters program in Business Management in any one of the listed universities.
- Register for an interview with any one of the listed companies on Nov. 20th using any one of the open slots.
- Cancel Registration for Workshop II

7.6 Procedure

7.6.1 Administration

The experiment was set up to test one subject at a time and was conducted by two administrators. The first administrator held the task list and administered the tasks while the second administrator recorded the task times and errors. Subjects were asked to read the instructions before starting the experiment and to sign the consent form. All the subjects were shown the different functionalities of the interface by browsing through the list of menu items. The subjects who were trained were shown the functionality of all the dialog boxes by opening each of the menu items. They were also allowed to use the interface for two minutes to experience the interface. The subjects in the training group were all trained using the same version (no terminology inconsistency) of the interface and then were tested using appropriate versions. Subjects were given time to ask questions or raise any doubts they had at the beginning or while performing the experiment.

7.6.2 Grading

As mentioned earlier two dependent variables, task completion times and subjective satisfaction ratings, were measured in this experiment. Task completion times for each task were measured using an electronic stop watch with an accuracy of 1/100th of a second by subtracting the start time from the finish time. These values were accumulated to get the total task completion time. Any time taken by the subject to ask questions during the experiment was excluded from the total task completion time. Error rates were observed to facilitate the deriva-

tion of results, but no measurement ratings were done for errors. The subjective satisfaction questionnaire had 19 questions whose ratings were summed up to get the total rating.

7.7 *SHERLOCK* Analysis

The terminology inconsistencies in the Career Center application can be detected by the *SHERLOCK* Terminology Baskets tool. These baskets with user predefined synonym terms are used to find misleading synonyms in the application. For example, shown below are the output of terminology baskets tool using the basket "Browse", "Browsing", "Query", "Search" and "Searching" when applied on the no and high inconsistency versions respectively. Comparing the outputs for this basket for both versions we find that the no inconsistency version of the interface only uses the term "Search" and "Searching", but in the high inconsistency version terms "Browse", "Browsing" and "Query" were used instead of the term "Search" in various parts of the application.

```
-------------------------------------------------------------------------------
Browse          Browsing        Query           Search          Searching
-------------------------------------------------------------------------------
TERM              DIALOG BOXES CONTAINING THE TERM
Browse
Browsing
Query
Search
                  coop.frm.cft      fulltime.frm.cft     intern.frm.cft
                  parttime.frm.cft
Searching
                  coop.frm.cft      fulltime.frm.cft     intern.frm.cft
                  parttime.frm.cft
```

Output: No Inconsistency Version

```
--------------------------------------------------------------------------------
Browse          Browsing        Query           Search          Searching
--------------------------------------------------------------------------------
TERM              DIALOG BOXES CONTAINING THE TERM
Browse            coop.frm.cft
Browsing
                  parttime.frm.cft
Query
                  parttime.frm.cft
Search
                  fulltime.frm.cft      intern.frm.cft
Searching
                  coop.frm.cft          fulltime.frm.cft      intern.frm.cft
```

Output: High Inconsistency Version

Similar results were obtained using the basket "Abort", "Discard", "End", "Exit", "Suspend". Although the term "Abort" was used consistently in the button labels for the no inconsistency version, the terms "Discard", "End", "Exit" and "Suspend" were used in some parts of the high inconsistency version instead of "Abort". These outputs are shown below:

```
--------------------------------------------------------------------------------
Abort           Discard         End             Exit            Suspend
--------------------------------------------------------------------------------
TERM              DIALOG BOXES CONTAINING THE TERM


Abort
                  canappt.frm.cft   carstapp.frm.cft      cnwkshps.frm.cft
                  coop.frm.cft      credent1.frm.cft      fellshps.frm.cft
                  fulltime.frm.cft  gradschl.frm.cft      intern.frm.cft
                  intsgn.frm.cft    parttime.frm.cft      pttmsrch.frm.cft
                  ques.frm.cft      resume.frm.cft        sgnwks.frm.cft
Discard
End
Exit
Suspend
```

Output: No Inconsistency Version

```
--------------------------------------------------------------------------------
Abort          Discard        End            Exit           Suspend
--------------------------------------------------------------------------------
TERM             DIALOG BOXES CONTAINING THE TERM
Abort
                 canappt.frm.cft   coop.frm.cft         credent1.frm.cft
                 fellshps.frm.cft  fulltime.frm.cft     intern.frm.cft
                 intsgn.frm.cft    ques.frm.cft         resume.frm.cft
                 sgnwks.frm.cft
Discard
                 gradschl.frm.cft
End
                 intsgn.frm.cft    parttime.frm.cft
Exit
                 pttmsrch.frm.cft
Suspend
                 carstapp.frm.cft
```

Output: High Inconsistency Version

Following are the other terminology baskets which helped in detecting these terminology inconsistencies:

- Major Field Area Program
- OK Done Apply Submit Send Forward
- Question, Questions, Inquiry, Inquiries
- Counselor, Counseling, Advisor, Advising
- Workshop, Workshops, Seminar, Seminars

Level of Expertise	No Inconsistency	Medium Inconsistency	High Inconsistency
Without Training	239.0 sec. (61.0)	287.4 sec (42.6)	312.7 sec (88.25)
With Training	204.0 sec. (41.7)	217.4 sec (50.6)	270.7 sec (30.5)

Table 7.1: Average Task Completion Time and Standard Deviation

7.8 Results

The experimental results summarized in Table 7.1 and 7.2 show that the no inconsistency version had a faster average task completion time than the medium and high inconsistency versions. Also, the average subjective satisfaction ratings for the no inconsistency version was higher than the medium and the high inconsistency versions of the interface. Overall, the performance improved when training was administered to the subjects, as the average task completion time was lower for all the three versions of the interface when training was provided. Also, the subjective satisfaction ratings were slightly higher in all the three versions of the interface when training was administered.

The 2 X 3 ANOVA (Analysis of Variance) was used to determine whether the interface types (versions) and the level of expertise have statistically significant effects on the task completion time and subjective satisfaction measured across the three treatments(no, medium and high level of terminology inconsistency) and two training levels (with prior training and without prior training). The ANOVA was conducted at the 5% significant level and the results are summarized in Table 7.3 and 7.4

Level of Expertise	No Inconsistency	Medium Inconsistency	High Inconsistency
Without Training	142 (14)	130 (14)	134 (13)
With Training	142 (12)	139 (11)	138 (13)

Table 7.2: Average Subjective Satisfaction Rating and Standard Deviation

Source of Variation	Sum of Squares	Degrees of Freedom	Variance Estimate	F-Statistic	F-Critical P=(.05)
Between	85530.60	5			
Level of Expertise	36015.00	1	36015.00	12.38	4.02
Interface Type	49515.60	2	24757.80	8.51	3.17
Level of Expertise X Interface Type	3430.00	2	1715.00	0.59	3.17
Residual (Error)	157057.00	54	2908.46		
Total	246017.00	59			

Table 7.3: ANOVA Task Completion Time

Source of Variation	Sum of Squares	Degrees of Freedom	Variance Estimate	F-Statistic	F-Critical P=(.05)
Between	2141.70	5			
Level of Expertise	866.40	1	866.40	2.19	4.02
Interface Type	1275.30	2	637.60	1.62	3.17
Level of Expertise X Interface Type	537.70	2	268.50	0.68	3.17
Residual (Error)	21332.20	54	395.04		
Total	24011.60	59	406.98		

Table 7.4: ANOVA Subjective Satisfaction

For the task completion time, we determined that the F-statistic value for the interface type was greater than the corresponding F-critical value (P= 0.05). Thus, the interface type had a statistically significant effect on the task completion time. Also, the F-statistic value for the level of expertise (training vs no training) was greater than the corresponding F-critical value, indicating that the level of expertise had a statistically significant effect on the task completion time. The F-statistic value for the interaction effect between interface types and the level of expertise was less than the corresponding F-critical value, indicating no interaction effect between the two variables. This implied that for the task completion time, the effect of the level of expertise is not dependent on the inconsistency level of the interface. Interpreting it further, we can say that the interface with terminology inconsistencies has higher task completion time

(lower performance) compared to the consistent interface and this performance is independent of whether the user is novice or expert. Therefore, performance of both the experts and novices reduces with increase in terminology inconsistencies. In summary, terminology inconsistencies decrease user's performance regardless of the user's level of expertise.

For the subjective satisfaction, the F-statistic values for both the level of expertise and the interface type were less than the corresponding F-critical values, implying that the interface type and the the level of expertise didn't have statistically significant effect on the subjective satisfaction of the user. There was no statistically significant interaction effect observed between the level of expertise and the interface type.

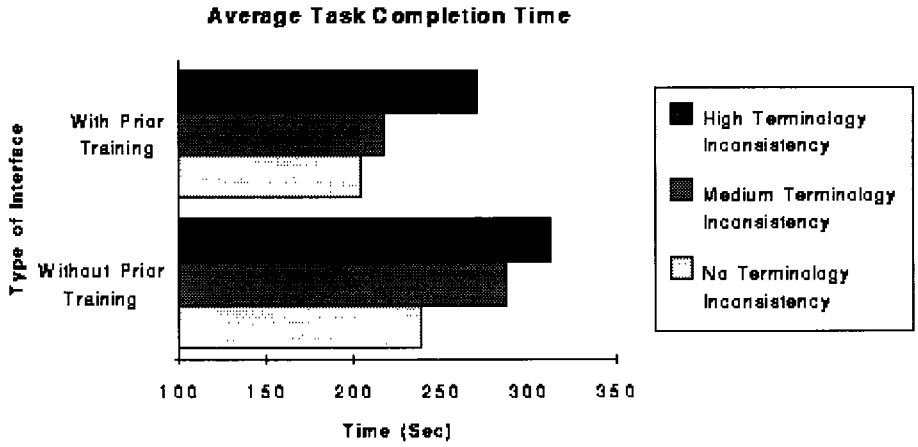


Figure 7.2: Average Task Completion Time

7.9 Discussion

The 2X3 ANOVA, as reported in the previous section enabled us to specifically answer the questions raised by the hypotheses set forth in this experiment. In

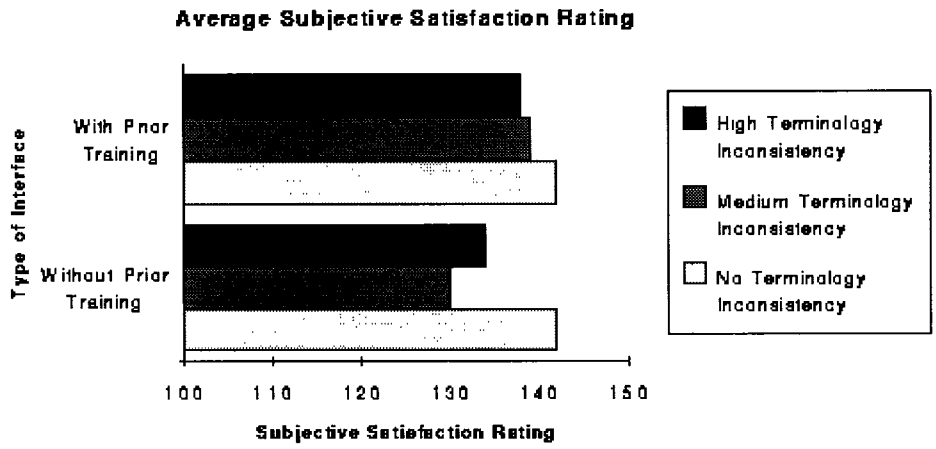


Figure 7.3: Average Subjective Satisfaction Rating

relation to the task completion time, the ANOVA identified that the terminology inconsistencies introduced in each version of the interface significantly affected the user's performance. In other words, increase in the level of inconsistency in the interface significantly reduced user's performance. These effects can also be observed in the task completion time difference between the averages of each treatment. In the treatments with no training the average task completion time for the medium and high inconsistency treatments were 20% and 30% [fig. 7.2] more than the no inconsistency treatment. Similarly in the block with training administered, the average task completion time for medium and high inconsistency treatments were 7% and 34% more than the no inconsistency treatment [fig. 7.2].

The level of expertise, according to ANOVA significantly effected the user's performance. In other words, users who were provided training had significantly better performance than the users with no training. On average, the training decreased the task completion time by 14%, 24% and 13% in no, medium and

high inconsistency versions respectively. Although the subjective satisfaction ratings for the medium and the high inconsistency versions were less than the no inconsistency version [fig. 7.3], the ANOVA analysis didn't give statistically significant results. It is difficult to obtain statistically significant differences in preference scores for between groups design, because subjects do not see the other versions. A future within subjects study might elicit stronger preference differences. Some subjects who were given training stated in their comments that training prior to utilizing the interface gave them a sense of comfort, confidence and ease of use, and as one subject stated "user training helped me in getting a preview of the interface, and in identifying the location of items".

7.10 Conclusion

In conclusion, this experiment supports the thesis that terminology inconsistencies in user interfaces significantly decreases user's performance. Also, prior training provided to the user before conducting the experiment, significantly increases user's performance.

Guidelines for Interface Designers

- Terminology within menu hierarchies should be consistent.
- Terminology in menu items should be consistent with the title of the dialog boxes which they open and with the labels of the widgets contained in those dialog boxes.
- Button Labels should be consistent across the interface, for example synonyms like "Abort", "Cancel", "Close" and "Exit" should not be used for similar tasks.

- Terminology should be consistent within the sequence of dialog boxes which are connected through button clicks.

- In addition to these consistencies within the dialog boxes, the interface should be consistent in terminology with the current commercial applications running on that system, but in accordance with the user's task domain.

The results of this experiment, along with the experiment done by Bajwa [2] support the encouragement to "strive for consistency" and including consistency as one of the prime guidelines when designing user interfaces. Also, the terminology inconsistencies introduced in the design of the experimental interfaces were detected by the *SHERLOCK Terminology Basket Tool.* This verifies the tool's capability in detecting misleading synonyms in user interfaces designed in Visual Basic.

Chapter 8

Conclusion

This chapter highlights the major contributions of this work and suggests possible future directions.

8.1 Contributions

The major contributions of this work are as follows

- Consistency Checking Tools: *SHERLOCK*, a family of user interface consistency checking tools, was developed to evaluate visual design and textual properties of user interfaces. The effectiveness of *SHERLOCK* was established by analyzing four commercial applications developed in Microsoft Visual Basic and numerous inconsistencies were detected in those applications.

- Scientific Experiment: A scientific experiment was conducted to study the effects of GUI terminology inconsistencies on user's performance and subjective satisfaction. The results showed that the user's performance is

significantly effected by GUI terminology inconsistencies.

8.2 Recommendations

8.2.1 Recommendations for GUI Application Developers

Following guidelines should be followed for a step towards creating consistent interfaces.

- Consistent Aspect Ratio especially for dialog boxes having similar visual design and functionality.
- Non-Widget Area (white space) should be atleast 20% of total area enclosed by the dialog box.
- Consistent margins within and across dialog boxes.
- Widgets within a dialog box should be both horizontally and vertically aligned as far as possible.
- Widgets within a dialog box should be properly aligned.
- Design with too many widgets in a small area should be avoided.
- Consistent background colors, foreground colors and typefaces.
- Consistent location and size of frequently used widget.
- Consistent terminology across dialog boxes [Section 7.10]

8.2.2 Recommendations for Designers Looking for GUI Evaluation Metrics

- To evaluate screen complexity, use metrics like Non-Widget Area and Widget Density of the dialog box summary table. Explore the use of other metrics for dialog box crowdedness like Tullis's Local Density metric [45] and Streveler and Wasserman's Hot-Spot metric [41].

- Use metrics like Aspect Ratio, Margins, and Balance to evaluate size of dialog boxes and create a more coherent and consistent layout.

- Metrics evaluation using additional visual techniques of Regularity, Proportion, Neutrality, Transparency and Grouping defined by Vanderdonck and Gillio [47] should be explored.

- Develop metrics to check consistency in typefaces and colors across dialog boxes in general and for every widget type in particular.

- Develop a better metric for detecting misaligned widgets similar to the dialog box summary table gridedness metric and previously developed Layout Complexity and Alignment metrics [45, 41].

- Develop metric to check consistency in size and location of specific widget types across dialog boxes. These metrics may be similar to those used in *SHERLOCK's* Button Concordance and Button Layout Table tools.

- Expand the Terminology Basket tool to detect misleading synonyms for specific widget types and across all the widgets.

- Implement tools similar to Interface Concordance and Interface Speller to detect variant capitalization, spelling errors and abbreviations.

8.2.3 Recommendations for Designers of New GUI Tools

The applicability of *SHERLOCK's* canonical format approach to other GUI development tools beyond Visual Basic was explored. The following are recommendations for designers of new GUI tools after analyzing existing tools like Visual Basic (limitations and exceptions), Visual C++, Galaxy, and Tk/Tcl.

- Visual design and textual properties of dialog boxes including widget labels, widget coordinates, widget sizes, typefaces, colors and more should be stored in an ASCII resource file. Many existing GUI development tools store this information as binary files.

- If the GUI development tool allows the user to dynamically change the widget size and position within the code, these changes should be updated to the corresponding resource files.

- The GUI development tool should not allow widget in the dialog box to extend beyond the area of the dialog box.

- If possible, the GUI development tool should create default left, right, top and bottom margins for dialog box beyond which widgets may not be extended.

- The GUI development tool should promote consistency by creating default dialog box templates so that developers are aware of the positioning of frequently used widget.

- A spell checking tool should be incorporated in the GUI development tool.

8.3 Future Directions

In this section, future directions are suggested. These include direct extensions to *SHERLOCK* and also other issues in research involving consistency checking in Graphical User Interfaces. After an year of effort trying to find commercial applications to be analyzed by *SHERLOCK* we were only able to find four Visual Basic applications. Therefore more work in the future needs to be done to further validate *SHERLOCK* tools.

8.3.1 Extension to *SHERLOCK*

- Analyze more interfaces developed in Visual Basic to further modify the dialog box summary table metrics like widget density, non-widget area, area balances and gridedness. Furthermore, remove those metrics that are not successful in detecting any inconsistencies.

- Analyze interfaces developed in Visual C++ to validate the canonical format approach on which the design of *SHERLOCK* is based.

- Subdivide the dialog box summary table into smaller tools. This subdivision would expand the analysis of each metric by reporting any exceptions or additional information along with metric values to facilitate the interpretation of results.

- Link the dialog box summary table to a spread sheet software like Microsoft Excel that can graph the inconsistencies for every metric.

- Add a generic tool to *SHERLOCK* which can detect visual design and textual inconsistencies in any type of widget, including combo boxes, drop down boxes, sliders, text boxes, similar to the inconsistencies detected by button layout table and button concordance tools in button widgets.

- Expand the terminology thesaurus of *SHERLOCK* in button sets of the button layout table and baskets of the terminology basket tool. This expansion would help in detecting more button inconsistencies and potentially misleading synonym terms.

- A new tool needs to be added to *SHERLOCK* that can separate dialog boxes with similar layout features and then use the other *SHERLOCK* tools to find inconsistencies between those similar dialog boxes. This tool would be useful because *SHERLOCK* analysis on the four commercial applications showed that too many inconsistencies existed between the dialog boxes with similar visual design and functionality.

8.3.2 Extension beyond *SHERLOCK*

SHERLOCK analysis is static and provides limited feedback to the user. Steps need to be taken to design a tool that incorporates *SHERLOCK* tools, but is dynamic in nature, where users can interactively modify the inconsistencies highlighted by *SHERLOCK*. Also, the future tool should be able to indicate the severity of the inconsistency, so that the designers can prioritize the modifications suggested by those tools. Perhaps the greatest challenge is to provide a "goodness" measures for the metric values. Similar problems exist in optimization-based engineering design, where algorithms need to evaluate the

severity of a constant violation. There is no "hard-and-fast" method for doing this, not yet, anyway. We have laid down the foundation for building the next generation generic GUI consistency checking tool to evaluate visual design and textual properties, but more work needs to be done to build a complete structure on this foundation.

References

[1] Apple Computer, Inc. (1992), *Macintosh Human Interface Guidelines*, Addison-Wesley Publishing Co., Reading, MA.

[2] Bajwa, S.(1995), Effects of inconsistencies on users performance and subjective satisfaction, *Unpublished, Department of Computer Science, University of Maryland.*

[3] Barnard, P., Hammond, J. , Morton, J., Long, J. and Clark, I. (1981), Consistency and compatibility in human-computer dialogue, *International Journal of Man-Machine Studies 15*, 87-134.

[4] Blake, T. (1986), Introduction to the art and science of user interface design, *Intuitive Software and Interactive Systems*, CA.

[5] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., and Vanderdonckt, J. (1994), Towards a dynamic strategy for computer-aided visual placement, In Catarci, T., Costabile, M., Levialdi, S., and Santucci, G. (Editors), *Proc. Advanced Visual Interfaces Conference '94*, ACM Press, New York, 78-87.

[6] Chimera, R. and Shneiderman, B. (1993), User interface consistency: An evaluation of original and revised interfaces for a videodisk library, In Shnei-

derman, B. (Editor), *Sparks of Innovation in Human-Computer Interaction*, Ablex Publishers, Norwood, NJ, 259-271.

[7] Chin, J.P., Diehl, V.A. and Norman, K. (1988), Development of an instrument measuring user satisfaction of the human-computer interface. *Proc. of the Conference on Human Factors in Computing Systems, CHI'90*, ACM, New York, 213-218.

[8] Coll, R., and Wingertsman, A. (1990), The effect of screen complexity on user preference and performance, *International Journal of Human-Computer Interaction 2,3*, 255-265.

[9] Comber, T. and Maltby, J. (1995), Evaluating usability of screen design with layout complexity, *Proc. of the CHISIG Annual Conference*, Melborne, Australia (in print).

[10] Firth, C. and Thomas, R. (1991), The use of command language grammar in design tool, *International Journal of Man-Machine Studies 34*, 479-496.

[11] Frederiksen, N., Grudin, J. and Laursen, B. (1995), Inseparability of design and use: An experimental study of design consistency, *Proceedings of Computers in Context'95*, Aarhus, Aarhus University, 83-89.

[12] Grudin, J. (1989), The case against user interface consistency, *Communications of the ACM 32,10*, ACM Press, New York, 1164-1173.

[13] Grudin, J. (1991), Interactive systems: Bridging the gap between developers and user, *Computer 4*, April 1991, 59-69.

[14] Grudin, J. and Barnard, P. (1984), The cognitive demands of learning and representing command names for text editing, *Human Factors 26,4*, 407-422.

[15] Grudin, J., Ehrlich, S. and Shriner, R., Positioning human factors in the user interface development chain. *Proc. of CHI+GI'87*, ACM, New York, 125-131.

[16] Grudin, J., and Norman, D. (1991), Language evolution and human-computer interaction, *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, Hillsdale, New Jersey, 611-616.

[17] Harrison, M.D. and Thimbleby, H.W. (1985), Formalizing guidelines for the design of interactive systems. *In Proc. of BCS HCI Specialist Group Conference HCI85*, 161-171.

[18] Jeffries, R., Miller, J., Wharton, C. and Uyeda, K. (1991), User interface evaluation in the real world: A comparison of four techniques, *Proc. of CHI 1991*, ACM, New York, 119-127.

[19] Kellogg, W. (1987), Conceptual consistency in the user interface: Effects on user performance, *In Proc. of Interact '87 Conference on Human-Computer Interaction*, Stuttgart.

[20] Kellogg, W. (1989), The dimensions of consistency, *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J., Academic Press Inc., London, 9-20.

[21] Kim, W. and Foley, J. (1993), Providing high-level control and expert assistance in the user interface presentation design, *Proc. of CHI 93*, ACM, New York, 430-437.

[22] Koritzinsky, I. (1989), New ways to consistent interfaces, *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J., Academic Press Inc., London, 93-106.

[23] Karat, C-M., (1990), Cost-benefit analysis of usability engineering techniques, *Proc. of Human Factors Society*, 34th Annual meeting, 839-843.

[24] Karat, C-M., (1992), Cost-justifying human factors support in development projects, *Human Factors Society Bulletin 35, 8.*

[25] Long, J., Hammond, N., Barnard, P., Morton, J. and Clark, I. (1983), Introducing the interactive computer at work: The user's view, *Behavior and Information Technology, 2*, 39-106.

[26] Lynch, P. (1994), Visual design for the user interface, pt. 1 design fundamentals, *Journal of Biocommunications 21, 1*, 22-30.

[27] MacIntyre, F., Estep, K.W. and Sieburth, J.M. (1990), Cost of user-friendly programming, *Journal of Fourth Application and Research 6, 2*, 103-115.

[28] Mahajan, R. and Shneiderman, B. (1995), A family of user interface consistency checking tools: Design & analysis of Sherlock, *Proc. of NASA Twentieth Annual Software Engineering Workshop*, (In print).

[29] Moran, T.P. (1981), The command language grammar: A representation for the user interface of interactive computer systems, *International Journal of Man-Machine Studies 15*, 3-50.

[30] Mullet, K.(1995), Organizing information spatially, *Interactions*, July 95, 15-20.

[31] Nielsen, H.(1989), *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J. Academic Press Inc., London.

[32] Norman, D.A. and Draper, S.W. (1986), *User Centered System Design: New Perspectives in Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, N.J.

[33] Polson, P., Muncher, E. and Engelbeck, G. (1986), A test of a common elements theory of transfer, *Proc. of CHI'86*, ACM, New York, 78-83.

[34] Reisner, P. (1990), What is consistency?, *Proc. of the IFIP Third International Conference on Human-Computer Interaction, Interact '90*, Elsevier Science Publishers, B.V., North-Holland, 175-181.

[35] Rosenberg, D. (1989), Cost benefit analysis for corporate user interface standards: What price to pay for consistent look and feel?, *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J. Academic Press Inc., London, 21-34.

[36] Sears, A. (1993), Layout Appropriateness: A metric for evaluating user interface widget layouts, *IEEE Transactions on Software Engineering 19, 7*, 707-719.

[37] Sears, A. (1994), AIDE: A step towards metric-based interface development tools, *Proc. of UIST '95*, ACM, New York, 101-110.

[38] Shneiderman, B. (1992), *Designing the user interface: Strategies for Effective Human-Computer Interaction: Second Edition*, Addison-Wesley Publ. Co., Reading, MA.

[39] Shneiderman, B., Chimera, R., Jog, N., Stimart, R. and White, D. (1995), Evaluating spatial and textual style of displays, *Proc. of Getting the Best from State- of the-Art Display Systems '95*, London.

[40] Smith, D.C., Irby, C., Kimball, R., Verplank, B. and Harslem, E. (1982), Designing the star user interface, *Byte 7, 4*, 242-282.

[41] Streveler, D. and Wasserman, A. (1987), Quantitative measures of the spatial properties of screen designs, *Proc. of INTERACT '87*, Elsevier Science, Amsterdam, 125-133.

[42] Tognazzini, B.(1984), Consistency for the Macintosh, *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J. Academic Press Inc., London, 89-92.

[43] Tullis, T.S. (1983), The formatting of alphanumeric displays: A review and analysis, *Human Factors 25*, 657-682.

[44] Tullis, T. S. (1988a), Screen design, In Helander, M. (Editor), *Handbook of Human- Computer Interaction*, Elsevier Science, Amsterdam, The Netherlands, 377-411.

[45] Tullis, T. S. (1988b), A system for evaluating screen formats: Research and application, In Hartson, H. Rex and Hix, Hartson (Ed.),*Advances in Human-Computer Interaction: Volume 2*, Ablex Publishing Corp., Norwood, NJ, 214-286.

[46] Turek, R. and Greenstein, J. (1991), Stars, polygons and clusters; An investigation of polygon displays, *International Journal of Man-Machine Studies, 35*, 807-824.

[47] Vanderdonckt, J. and Gillo, X. (1994), Visual techniques for traditional and multimedia layouts, In Catarci, T., Costabile, M., Levialdi, S. and Santucci, G. (Editors), *Proc. Advanced Visual Interfaces Conference '94*, ACM Press, New York, 95-104.

[48] Wiecha, C., Bennett, W., Boies, S. and Gould, J. (1989), Tools for generating consistent user interfaces, *Proc. of CHI'89, Human Factors in Computing Systems.*

[49] Wolf, R. (1989), Consistency as process, *Coordinating User Interfaces for Consistency Checking*, Ed. Nielsen, J. Academic Press Inc., London, 89-92.

[50] Yourdon, E. (1989), *Modern Structured Analysis*, Yourdon Press, New York.