Universidad Nacional Mayor de San Marcos

(Universidad del Perú, DECANA DE AMÉRICA)

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA DE POST GRADO

Adaptación del algoritmo Grasp en el diseño eficiente de la interfaz gráfica de usuario

"Tesis presentada para optar el Grado Académico de Magister en Ingeniería de Sistemas e Informática con mención en Ingeniería de Software"

Presentado por

Bach. Juan José Zamudio Díaz

Orientador: Dr. Luis Rivera Escriba

UNMSM – LIMA
Octubre, 2007

DEDICATORIA

Este trabajo está dedicado a toda mi familia, en especial a mi madre.

Con todo mi corazón a Techy, la mujer que cambio mi vida.

AGRADECIMIENTOS

- Al profesor Luis Rivera, por su orientación y dedicación para que este trabajo cumpla con los objetivos trazados.

- Al profesor David Mauricio, co-asesor, por su orientación, consejos y revisiones del presente trabajo.

- A mis colegas y amigos de maestría Erick Vicente y Manuel Tupia, por sus observaciones y porque en todo momento me incentivaron para que culmine este trabajo.

- A los profesores de la UNMSM, principalmente al Dr. Angel Coca por sus observaciones teóricas que me sirvieron de mucho.

- A todas aquellas personas que indirectamente me ayudaron para culminar este trabajo y que muchas veces constituyen un invalorable apoyo.

- A Dios, porque sin él nada es posible.

ÍNDICE

Adaptación del algoritmo Grasp en el diseño eficiente de la interfaz gráfica de usuario

RESUMEN

Existen algunos métodos en la actualidad para el mejoramiento de la interfaz gráfica de usuario (GUI), métodos que, muchas veces, se basan en la recopilación de información vía *test* de usuarios y en la creatividad del diseñador; de acuerdo a las funcionalidades que debe proporcionar la GUI para el procesamiento de los datos. En el presente trabajo desarrollaremos una metodología de optimización para el diseño de una GUI utilizando un algoritmo de procedimiento de búsqueda adaptativa aleatoria (GRASP), que tendrá como base un algoritmo goloso de tipo FFD (*first fit decreassing*), que nos permitirá encontrar un diseño óptimo de una GUI, basado en la eficiente distribución de objetos visuales. Este diseño óptimo servirá para la correcta comunicación hombre-maquina en el eficiente procesamiento de los datos. Cabe recalcar que siempre un diseño de una GUI guarda una estrecha relación con el desarrollo cognitivo del usuario. Esto puede ocasionar una pequeña variación en el diseño final de una GUI.

Palabras clave: GRASP, GUI, usuario, interfaz, testing.

Adaptation of Grasp algorithm in the efficient design of graphic user interface

ABSTRACT

There are ome methods at the present time for the improvement of the Graphic User Interface (GUI), methods that, often, are based on the compilation of information through *test* of users and on the creativity of the designer; according to the functionalities that must provide the GUIs to data processing. In the present work we will develop to a methodology of optimization for the design of a GUI, using a Greedy Randomized Adaptative Procedures Search (GRASP), which it will have as it bases a goloso algorithm of FFD (*first fit decreassing*) type, that will allow to find an optimal design of a GUI optimal, based on the efficient distribution of visual objects. This optimal design will be used for the man machine right communication in the efficient data processing. It is important to stress that a design of a GUI always keepsa narrow relation with a user's cognitive development. It can cause a little variation in the final design of a GUI.

Key words: GRASP, GUI, user, interface, testing.

CAPÍTULO I: INTRODUCCIÓN

Para establecer una interacción usuario-computadora (*user-computer interaction* - UCI), se requiere de una interfaz fácil de aprender, usar y recordar, eficiente, con bajo coeficiente de error en su uso y que generen satisfacción en el usuario. Esto es lo que comúnmente conocemos en ingeniería de software como "Usabilidad". Hoy en día, la industria del *software* requiere usabilidad para desarrollar *software* con un interfaz apropiado para usuarios que realizan tareas con la ayuda del *software*. Este es un componente fundamental de la calidad del producto, porque asegura que el software va a ser usado y explotado tal como fue proyectado. Los beneficios concretos que reporta esta disciplina varían de acuerdo a la naturaleza del *software* desarrollado.

El problema del diseño de la interfaz gráfica de usuario (*Graphic User Interface* – GUI), es un problema que básicamente pasa por la correcta o incorrecta distribución de los elementos de interfaz, comúnmente conocido como objetos, sobre la pantalla que define la interfaz de comunicación con el usuario, también llamado diálogo. Por mucho tiempo se ha buscado introducir técnicas que aseguren una correcta distribución de esos objetos sobre una GUI, de forma a permitir entablar una comunicación natural usuario-computadora que asegure el correcto procesamiento de la información. La meta es reducir al mínimo la distancia recorrida entre los objetos en una interfaz que implementa una tarea específica, de forma que el programa de interacción pueda ser amigable y óptimo respecto al tiempo.

Algo parecido ocurre en el campo industrial, donde los especialistas han utilizado por más de 20 años el análisis de acoplamiento para realizar una correcta distribución de los objetos y equipos en un ambiente [1]. El análisis de acoplamiento depende de una descripción de la localización de los objetos y el valor de la distancia entre ellos. Los valores de acoplamiento de los objetos pueden ser la frecuencia de recorrido de un objeto a otro multiplicado por la distancia entre dichos objetos. Será más eficiente la distribución de esos objetos cuando la separación entre ellos sea mínima. El esquema mostrado en la Figura 1-1 (b) es un ejemplo de una oficina que se reorganizó completamente para acortar distancia entre las diferentes secciones. Los funcionarios de la oficina deben moverse con frecuencia de una localización para otra; si bien es cierto que no todos los acoplamientos entre objetos se pudieron reducir al mínimo, pero los acoplamientos con frecuencias más altas se redujeron dramáticamente (Figura 1-1 (b)) a comparación de la configuración

inicial, mostrada en la Figura 1-1(a). Similarmente, el problema de organizar varios departamentos y equipos de un edificio puede ser enfocado de la manera de poder distribuir las instalaciones o la manera de la disposición de los objetos. Ese problema de alguna manera podría ser tratado por una métrica basada en la conveniencia de la disposición [2].

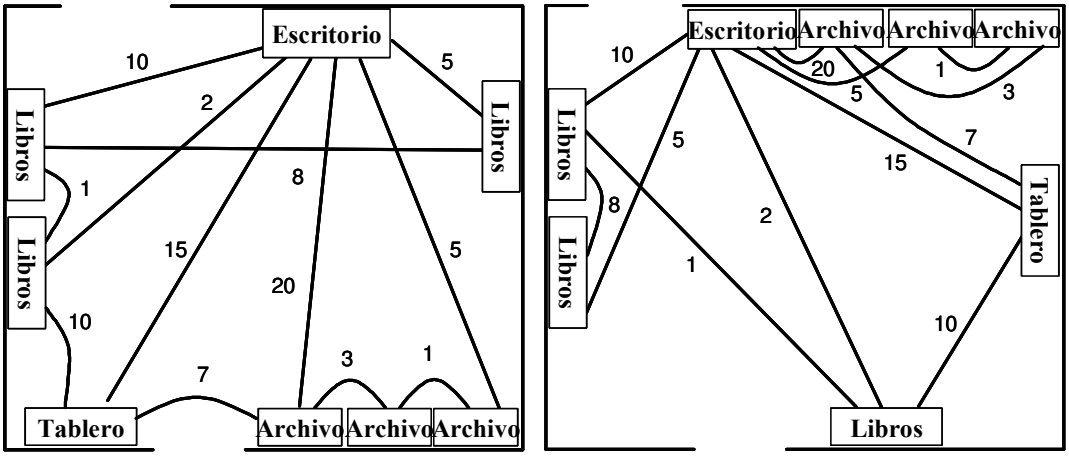


Figura 1-1: Distribución y acoplamientos entre objetos en un ambiente espacial: (a) distribución original; (b) objetos redistribuidos de la mejor forma.

La distribución de los objetos en espacios geométricos, descritos arriba, es inspirada para la distribución de objetos de interfaz en un ambiente de GUI en el área de Computación, ya que la frecuencia de manipulación de un objeto (botón, barra, menú, etc.) en una interfaz puede crear ciertas actitudes insatisfactorias en los usuarios finales. Por otro lado, la distribución de los elementos de interfaz sobre un diálogo tiene bastante que ver con el comportamiento de los usuarios finales. Entonces, las actividades de manipulación de los objetos de GUI pueden ser conceptuadas como tareas, que pueden ser descritas analizando el aspecto cognitivo de los usuarios. En este aspecto hasta se podría delegar al usuario la distribución de los objetos en su diálogo de trabajo.

1.1 Tipos de usuarios

A nivel operacional, podemos distinguir tres tipos de usuarios. Cada uno de estos tipos de usuarios utiliza la computadora de acuerdo a los conocimientos que sobre él se tiene y a la necesidad que estos usuarios tienen de la computadora. Los tres tipos de usuario son:

A) Usuarios esporádicos.- Son aquellos usuarios que utilizan la computadora de manera limitada, mayormente entablan una comunicación con la computadora, sólo para solucionar determinados problemas y no ven a la computadora como una herramienta necesaria con múltiples utilidades.

B) Usuarios intermedios.- Son aquellos usuarios que, con mayor experiencia en el manejo de la computadora, la utilizan para solucionar múltiples problemas. Generalmente, son muy asiduos al uso de internet y herramientas de cuarta generación (excel, word, power point, access, etc.). Estos rápidamente ven a la computadora como una solución factible, ya que tienen un nivel de operario avanzado.

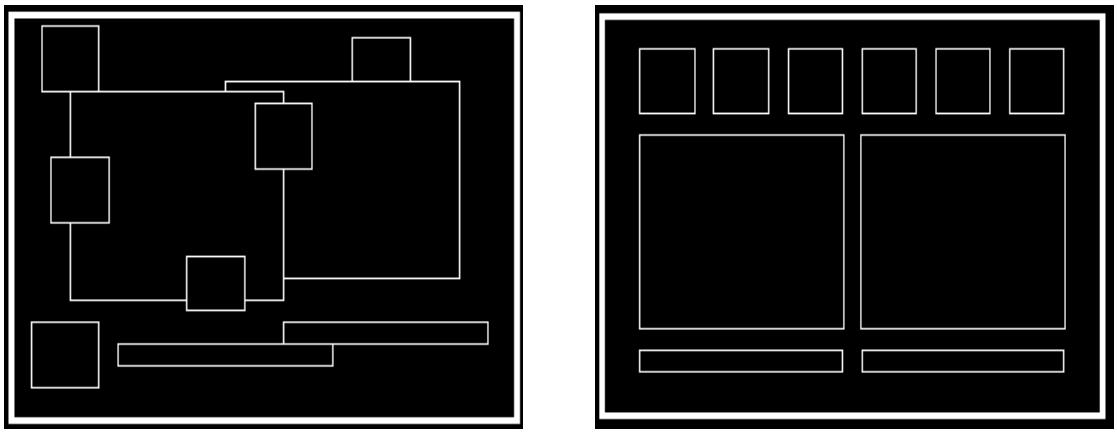
C) Usuarios expertos.- Son aquellos usuarios de mucha experiencia, mayormente son personas que se dedican al desarrollo de aplicaciones, manejo de bases de datos, desarrollo web, dominan 2 o 3 lenguajes de programación, se pueden adecuar rápidamente a diferentes plataformas, caso Linux, Windows, etc. Son usuarios que tienen un manejo óptimo de la computadora, e inclusive muchos de ellos tienen conocimientos de ensamblaje y reparación de equipos de cómputo.

1.2 Trabajos relacionados

Según Martin [35], existen tres principios fundamentales relacionados para el diseño de una GUI: organizar (proveer al usuario de una estructura conceptual, clara y consistente), economizar (hacer lo máximo con la mínima cantidad de elementos), y comunicar (ajustar la presentación a las capacidades del usuario). Si hablamos de organizar, debemos decir que el diseño de una GUI debe tener los siguientes conceptos: consistencia, disposición de la pantalla, relaciones y navegabilidad.

En la Figura 1-2, podemos observar dos tipos de organización en una GUI. Cuando los objetos visuales no están bien distribuidos vemos una organización caótica que dificulta la comunicación usuario-computadora. Si consideramos el principio de economizar, debemos tener en cuenta tres factores importantes: simplicidad, claridad y singularidad. Con la simplicidad incluimos únicamente los elementos que son más importantes para la comunicación; con la claridad podemos obtener interfaces de usuario que tengan componentes no ambiguos que lleven a una comunicación errónea entre el usuario y la

computadora; y con singularidad, la GUI debe poseer elementos fácilmente percibidos, ser legible y rápidamente perceptible para el usuario.



(a) (b)

Figura 1-2: Organización de objetos en un ambiente: (a) organización caótica; (b) organización ordenada.

La comunicación es un principio fundamental en la interacción usuario-computadora; por ende, los elementos u objetos visuales que se utilizan para el diseño de una GUI deben provocar en el usuario una rápida idea del trabajo operativo que estos realizan y de la función que cumplen dentro de una GUI. En consecuencia, la simbología (imágenes, gráficos, comandos, etc.) que usemos para la comunicación es de suma importancia para la interacción usuario-computadora. Un mal uso de esos símbolos provocará un defecto en esa comunicación [36]. A pesar que hoy en día existen herramientas para diseñar GUI de forma rápida, el costo final podría ser caro si esa GUI fue diseñada sin criterios básicos para estos definidos.

Casner y Larkin [20] desarrollan un sistema que utiliza descripciones de las tareas, para tal motivo se realiza exhibiciones gráficas para las tareas de extracción de la información. En ese sistema se puede observar que los operadores visuales son sustituidos automáticamente por los operadores lógicos. Al usar estos operadores, se intenta reducir las demandas cognoscitivas puestas en los usuarios. Es un método basado en tareas independientes, por lo tanto no se puede evaluar un plan de tal manera que los usuarios de dichas tareas la realicen normalmente.

Goms [7, 8] formula un método que requiere las descripciones de la tarea detallada. Este método puede tomar muchos aspectos de las interacciones en consideración. Las

descripciones de la tarea detallada pueden ser difíciles de obtenerse, y los cambios en las tareas pueden requerir un análisis bastante extenso. A parte de ello, el análisis de la tarea detallada puede requerir bastante entrenamiento que muchas veces puede ser impráctico y costoso. Este tipo de método (tarea independiente) puede proporcionar información útil, pero no puede evaluar la adecuación de una interfaz para las tareas que los usuarios realmente realizcen.

Tullis [9] investiga métodos de construcción de GUI prediciendo el despliegue de los usuarios, explorando la relación entre varias métricas, que incluye densidad global de la pantalla, la densidad de la pantalla agrupada de objetos, la complejidad del diseño y el tiempo que los usuarios demoran en extraer la información. Dirige varios experimentos para predecir tiempos de búsqueda, pero no toma en cuenta la descripción de las tareas que los usuarios realizan, con eso el no puede evaluar la adecuación de la interfaz para las tareas que los usuarios realmente realizan. Proporciona los valores recomendados para cada métrica, para guiar a los diseñadores, pero no explora los beneficios de la descripción de las tareas.

Perlman [10] desarrolla un modelo axiomático de presentación de la información. Este modelo permite ver a los diseñadores la relación que existe entre los datos de entrada y la información a ser presentada, incorpora un prototipo que proporciona la regeneración de la interfaz frente a los problemas potenciales que se presentan. El problema de este método es que no hay un mecanismo que compare los diseños alternativos. La única contribución es que este método empieza con las "intenciones del diseñador", pero no considera incorporar las tareas del usuario a las intenciones del diseñador. Ya que enfatiza las intenciones del diseñador y no las tareas del usuario, aunque él intenta generar las reglas y relaciones que reflejan las tareas de los usuarios. Ninguna descripción de las tareas está explícitamente incorporada al modelo.

Lohse [11] desarrolla un modelo para predecir la cantidad de tiempo necesario para extraer la información de un objeto. Este trabajo se enfoca sobre las actividades que los usuarios realicen y las tareas cognoscitivas que requiriesen cada uno, pero este modelo solo predice la cantidad de tiempo necesario para realizar una tarea o un conjunto de tareas, y no da énfasis al diseño óptimo ni a los diseños existentes evaluados. Lohse se refiere a tareas cognitivas, principalmente, a las tareas que van a ser realizadas, las sucesiones de acciones que se van a realizar para completar una tarea. Una vez que estos pasos se

completan, ellos evalúan la facilidad de aprender la interfaz propuesta. El problema es que, quizás, aplicar este modelo resulta en una inversión considerable de tiempo que, muchas veces, la gerencia no esté dispuesta a otorgar.

Sears [16] formula una métrica llamada "diseño apropiado". En esta métrica compara el diseño propuesto con el diseño óptimo. El diseño propuesto es preparado por el diseñador y el diseño óptimo es hallado en base a un algoritmo de búsqueda en árbol, en la cual redistribuye los objetos sobre la GUI. El área de la interfaz es convertida en una grilla con una determinada cantidad de celdas uniformes. Estas celdas representan una posible posición para cada uno de los objetos visuales sobre la interfaz, esto quiere decir que mientras más celdas haya en una grilla, mayor cantidad de posibles posiciones existen para los objetos. Una vez que los objetos son redistribuidos se pasa a calcular el costo de la interfaz; aquí es donde este modelo incorpora las tareas de usuario. Cada tarea de usuario supone dirigirse con el puntero del mouse o algún otro dispositivo de entrada de un objeto visual a otro, esta distancia que recorre el puntero del mouse es multiplicada por la frecuencia (número de veces que realiza esta tarea), el resultado es el costo de realizar dicha tarea. La interfaz de usuario tiene un número determinado de tareas de usuario, la sumatoria de los costos de estas tareas es el costo total de la interfaz, en el algoritmo de búsqueda en árbol. Sears redistribuye los objetos hasta obtener la interfaz con menor costo. Este es comparado con los costos de la interfaz de usuario propuesta, y de esta manera se puede ver que tan eficiente es la interfaz propuesta y obtener un mejor diseño.

La desventaja de este modelo es que no considera para nada el aspecto semiótico (impacto de las imágenes y objetos visuales en la mente del usuario), que proporciona crear mejores GUIs, ya que es el usuario final el que va a interactuar con la computadora. Por otro lado, podemos decir que el algoritmo de búsqueda en árbol de Sears genera recursividad, y con una determinada cantidad de objetos que ocupen varias celdas de una grilla y con numerosas posiciones en las cuales puedan ser ubicados estos objetos, los costos de procesamiento pueden ser muy altos.

1.3 Aplicativos o *softwares* comerciales

Existen algunas herramientas para modelar GUI, pero siempre tienen algunas desventajas, debido a que, entre otros aspectos, no consideran las descripciones de las tareas que los usuarios deben realizar. Esto da una ventaja significativa al método de Sears que sí

contempla descripción de las tareas. De esta manera, podemos contemplar y optimizar una GUI, teniendo en cuenta las acciones de los usuarios y como optimizar estas acciones que a la larga es lo que debe ser para una correcta comunicación usuario-computadora.

A seguir mencionados, como ejemplo, dos aplicativos existentes:

AIDE: Se basa principalmente en la metodología de Andrew Sears. Da prioridad a la preferencia de los diseñadores. Está desarrollado en Java de Sun Microsystem. Se basa principalmente en el algoritmo que él mismo utiliza para rediseñar una GUI. Este *software* fue desarrollado en la Escuela de Ciencias de la Computación de la Universidad Depaul en Chicago [47].

ESSI PIE 24306: *Software* basado en la parte funcional de los objetos visuales y en las acciones que estos realizan. Trata de descubrir los errores en las rutinas, subrutinas, *bucles* que estos ejecutan, tiene una característica muy interesante de colocar *checkpoints* en el *script* de los objetos y, de esta manera, realizar un *testing* personalizado. Fue desarrollado en Alemania [49].

1.4 Propuesta del trabajo

Habiendo realizado un preámbulo de lo que a control de calidad se refiere y tomando como referencia a Ben Shneiderman [12], decimos lo siguiente "Las interfaces graficas ventanas, íconos y selecciones mediante ratón han eliminado muchos de los terribles problemas con la interfaz. Pero incluso en un mundo de ventanas todos encontramos interfaces de usuario difíciles de aprender, difíciles de utilizar, confusas, imperdonables y en muchos casos totalmente frustrantes. Sin embargo hay quien dedica tiempo y energías construyendo estas interfaces, y es posible que estos problemas no los crean a propósito".

En este trabajo, partimos del método formulado por Sears [16] en la evaluación de una GUI mediante la utilización de la descripción de las tareas de usuario. Se propone una Grasp basado en un criterio goloso (voraz), FFD (*First Fit Decreasing*) para determinar una redistribución eficiente de los objetos sobre una GUI existente. Para eso consideramos los aspectos semióticos de los objetos que componen la interfaz. Para validar nuestro método realizamos simulaciones con objetos sintéticos expresados por sus dimensiones, posiciones en una GUI y matrices de valores asociados a los significados de los objetos.

Podemos decir que tener una correcta interfaz de usuario o desarrollar una que se ajuste a la "realidad", de lo que el usuario pretende, no es una tarea fácil. Nuestro trabajo consiste básicamente en rediseñar una GUI, redistribuyendo sus objetos visuales de tal manera que la comunicación usuario-computadora sea más eficiente y la GUI cumpla mejor los objetivos para la cual fue diseñada.

Por los tipos de usuario que hemos definido anteriormente, la GUI debe permitir el aprendizaje rápido y continuo al usuario, sea este el nivel que tiene, no vamos a pretender que nuestra GUI vaya a ser utilizada solo por usuarios expertos. Esta tiene que ser diseñada para todo tipo de usuario (esporádicos, intermedios y expertos). Muchos productos de *software* han sido desechados porque la comunicación o la interacción usuario-computadora mediante las GUIs de estos productos de *software* estaban tan mal diseñadas, que el usuario cuando se sentaba frente al computador no sabía aa donde ir, ni con qué entidad (llamamos entidad a los objetos sobre la GUI) actuar primero, esto conlleva a que muchas compañías desecharan *softwares* que, internamente, eran muy buenos (tiempo de respuesta adecuada, estructura de datos buena, almacenamiento continuo óptimo, capacidad de responder a fallas inmediatas, etc.), pero eso de nada vale si es que la comunicación usuario-computadora, por un mal diseño de la GUI, entorpece la comunicación usuario-computadora.

1.5 Organización del trabajo

El trabajo esta organizado en seis capítulos. Dedicamos el Capítulo II a la interfaz gráfica de usuario, la comunicación usuario-computadora, los tipos de comunicación y sus principios fundamentales. En el Capítulo III, se aborda el desarrollo de la métrica para una GUI, los costos de distribución de los objetos y el diseño óptimo. En el Capítulo IV, describiremos la metaheurística en la evaluación de una GUI, un algoritmo goloso en la distribución de los objetos y la adaptación del Grasp. En el Capítulo V, lo tenemos dedicado a los experimentos numéricos, los requerimientos de *hardware* y *software*, el *testing* y los resultados numéricos. Finalmente, en el Capítulo VI, presentamos las conclusiones y trabajos futuros.

CAPÍTULO II: GUI EN LA INTERACCIÓN USUARIO-COMPUTADORA

Los elementos de una GUI están configurados de tal manera que cumplan con la función de procesamiento de la información. Esa función no se realiza por sí sola, sino que es producto de una interacción entre el usuario y la computadora, teniendo justamente como enlace de comunicación una GUI. El usuario es quien decide cómo y cuándo realizar el procesamiento de la información mediante la activación de los elementos de la GUI. El usuario es el "actor" que interactúa con la computadora que hospeda los aplicativos que va atender sus necesidades, es aquí donde la GUI cumple una función primordial. Si la GUI está mal diseñada generará consecuencias funestas en la comunicación usuario-computadora, y por ende el procesamiento de los datos se realizará en forma errada o demandará mayor tiempo de operatividad.

Ante la demanda de uso de computadoras en las actividades diarias del hombre, las GUI se han vuelto en un "punto crítico" del *software*, y es por ello que los diseñadores de interfaces cada vez están preocupados en mejorar sus diseños para permitir que la comunicación usuario-computadora sea más efectiva, precisa y las tareas operacionales se realicen con mayor rapidez.

2.1 Interacción usuario-computadora

La interacción entre dos o más agentes[1] puede ser definida como el "intercambio de símbolos entre las partes, asignando a los participantes en el proceso comunicativo los significados de estos símbolos" [32]. Esta definición nos permite identificar a los participantes (usuario y la computadora), al mensaje y a su significado. El significado y el símbolo usado para el mensaje pueden ser interpretados de manera distinta por el usuario, esto va a depender de su estatus, conocimiento y emociones. Es por ello que los objetos visuales que usemos sobre la superficie de una GUI, deben tener la simbología adecuada (imágenes, forma de los objetos, distribución adecuada), de esta manera lograremos un

[1] Agentes: elementos que sean capaces de realizar una tarea. En este caso, el agente es parte del proceso de comunicación.

mejor impacto en el usuario, este interpretará de manera más eficiente el diseño de la GUI y el objetivo para la cual fue construida.

Las técnicas de interacción usuario-computadora (IUC), conocido también como interacción hombre-máquina, ayudan a entender cómo la gente interactúa con las nuevas tecnologías. Además, esta interacción puede ayudar a mejorar las posibilidades de las nuevas tecnologías en la enseñanza en dos importantes aspectos: primero, puede guiar en el análisis cuidadoso y sistemático sobre qué información, herramientas y capacidades la gente necesita para conseguir sus objetivos; y segundo, puede proporcionar herramientas y técnicas para evaluar y quitar defectos que estorban en una interacción tranquila entre la gente y las nuevas tecnologías.

Es decir, es necesario que se profundice en los factores que dificultan la interacción. Este tipo de investigación permitirá la generación de guías para el diseño de interfaces de diálogos entre usuarios y computadoras.

En los últimos años, se ha ido incrementando el interés en el estudio de los usuarios como parte del sistema hombre-máquina [37]. No obstante, la mayoría de los estudios han sido dirigidos hacia los usuarios con experiencia con la computadora, o más específicamente a programadores. Sólo algunos de los más recientes estudios se ocupan más específicamente de los usuarios casuales o principiantes, un caso palpable es el diseño o el constante rediseño de las interfaces de los cajeros automáticos, donde el apuntador son las manos del usuario y estas interfaces están destinadas a usuarios sin experiencia o casuales.

Los usuarios principiantes y experimentados generalmente manifiestan maneras de comportamiento bastante diferentes. Los principiantes normalmente se dedican a actividades de resolver problemas que son mayormente entender los objetos visuales que existen sobre la GUI, mayormente se le presentan las siguientes interrogantes: ¿por cuál objeto visual iniciar la interacción con la computadora?, ¿estas imágenes qué representan?, ¿para qué servirá esta opción del menú?, ¿cómo termino este programa?, etc. Mientras que los experimentados son hábiles en la interacción con la computadora. La interacción es para el usuario experto una destreza cognitiva de rutina. Además, junto al nivel de experiencia del usuario es necesario que la GUI posea un diseño intuitivo, es decir a medida que el usuario explora la interfaz, esta debe facilitar su aprendizaje.

La investigación en IUC está tratando de desarrollar nuevos dispositivos y estilos de interacción que incorporen las capacidades del lenguaje entre personas. La comunicación tiene dos partes importantes un emisor y un receptor. Estos necesitan encontrarse en un mismo contexto. Un ejemplo práctico son los famosos programas de reconocimiento de voz, en el cual el computador reconoce la palabra emitida por el emisor y el receptor, que es el computador, ejecuta la orden emitida por el emisor que en este caso es el usuario.

La interfaz de usuario está constituida por una serie de dispositivos, tanto físicos como lógicos, que permiten al hombre interactuar de una manera precisa y concreta con un sistema. El planteamiento del área de IUC es estudiar las relaciones que se establecen entre las personas y la tecnología computacional. De esta forma, la IUC se relaciona con el diseño de sistemas para que las personas puedan llevar a cabo sus actividades productivamente, con unos niveles de "manejabilidad", "usabilidad" o "amigabilidad" suficientes. Esto se concreta en términos de simplicidad, fiabilidad, seguridad, comodidad y eficacia [6].

La IUC estudia:

- el *hardware*, el *software* y la repercusión de ambos en la interacción,
- los modelos mentales de los usuarios frente al modelo del sistema,
- las tareas que desempeña el sistema y su adaptación a las necesidades del usuario,
- el diseño dirigido al usuario, no a la máquina (*user-centered design*), y
- el impacto organizacional.

Cada vez se plantea con más auge aplicar también estos principios de IUC a la construcción de otras aplicaciones, entre ellas las de recuperación de información; de hecho, la evolución de los videojuegos, desde su aparición hasta la actualidad, nos confirma el gran desarrollo que han sufrido estos sistemas dirigidos al usuario final en cuanto a su interfaz, que ha favorecido la facilidad de manejo.

2.1.1. Interfaces de usuario y programas

El usuario interactúa con la máquina a través de los objetos distribuidos en la GUI. Lógicamente que detrás de esos objetos, como del diálogo, están los programas cuyas fuentes están en un lenguaje de programación específico (java, visual basic, power builder, delphi, etc.), que realizan la secuencia de tareas que el objeto visual representa para el

usuario. Esos programas, siguiendo una metodología de programación, en particular orientado a eventos, están asociados a los objetos gracias a la propiedad de encapsulamiento de esos lenguajes de programación modernos. Cuando un usuario manipula un elemento de la GUI genera un evento que activa y/o desactiva una secuencia de acciones realizadas por los segmentos de programas asociados a la GUI y el sistema que el usuario desea manipular.

La Figura 2-1 ilustra el clásico modelo de la IUC, en la cual mediante una interfaz de usuario definida, la persona establece comunicación con la computadora.

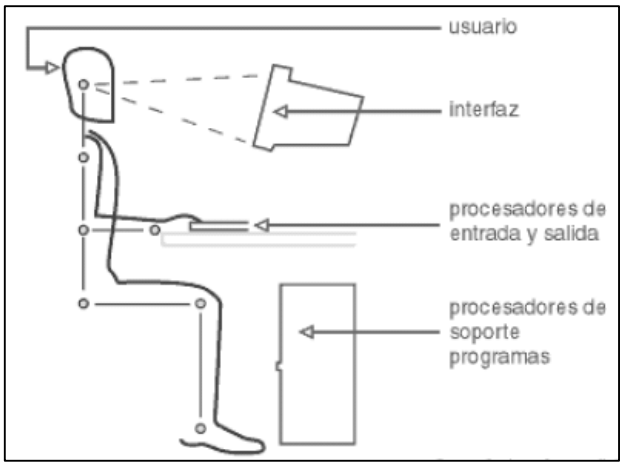


Figura 2-1: Modelo clásico de la interacción usuario-computadora.

La IUC, a través de la GUI, envuelve tres grandes partes del recurso computacional: los dispositivos de entrada de información, los de proceso y los de salida. El usuario se enfrenta con los que le permiten comunicar a la máquina lo que desea hacer y con los que este utiliza para responderle, pero nada de esto sería posible si no existiera una unidad de procesamiento de información. Los dispositivos de entrada y salida tendrán especial importancia en los aspectos ergonómicos y de diseño, que afectarán a algunos factores importantes para la IUC, como la velocidad de ejecución de tareas por parte del usuario, la precisión, los errores, el tiempo necesario de aprendizaje, etc.

2.1.1.1 Dispositivos de entrada

Se distinguen dos tipos de dispositivos de entrada: los de introducción de texto y los de indicación de posición y selección o *pointing-selecting*. En el primer tipo, se encuentra el teclado, hasta ahora el más utilizado, si bien el auge de los dispositivos de *pointing* en esta

década está situando a algunos de ellos en una posición también privilegiada. Estos últimos permiten una manipulación directa de los objetos que aparecen en la pantalla, con la ventaja de que el usuario no tiene que aprender comandos, no se equivoca al teclear y centra la atención directamente en la pantalla. Las interacciones que se producen con este tipo de dispositivos han sido estructuradas por Foley [33], y las presentamos a continuación con algunas modificaciones que nos han parecido convenientes para adaptarlas al contexto actual:

- Selección de un objeto o un item entre varios,
- Posición de un punto donde comenzar una acción o colocar un objeto, y
- Orientación y ruta de un objeto que se va a hacer mover.

En algunos casos, el usuario ejerce un control directo sobre la pantalla, en otros el control es indirecto. Dentro del primer grupo se encuentran aquellos en los que el usuario tiene un contacto físico con la pantalla: la pantalla táctil y el lápiz óptico. En el segundo consideraremos el ratón, el *trackball*, el *joystick*, la tableta gráfica y el *touchpad*, además de otros que están todavía en desarrollo. En el tercer grupo, podemos considerar los dispositivos adaptados al computador para el reconocimiento de voz, los lectores ópticos que realizan reconocimiento ocular, los cascos con principios de realidad virtual que, por ejemplo, utiliza la fuerza aérea americana para el entrenamiento de sus pilotos de combate, en el cual pueden simular un combate virtual con aviones enemigos que se asemeja mucho a la realidad.

Shneiderman [12] presenta el resultado de un estudio sobre el tiempo que tarda el usuario en responder a los estímulos con esos dispositivos, descubrimiento ya hecho por Paul Fitts en 1954, de ahí la ley de Fitts que se tratará en el Capítulo III. Este autor detectó que el tiempo que se tarda en apuntar a un objeto determinado va en función de la distancia a la que se encuentra del cursor en ese momento y del tamaño del objeto.

Veamos los dispositivos de entrada más utilizados y que se pueden aplicar a computadoras con sistemas de recuperación de información en el sentido que entendemos dentro de las Ciencias de la Documentación (ciencia que estudia la mejor manera de recuperar, catalogar y ordenar información, en estos tiempos modernos podemos agregar usando tecnologías de la información).

a) Teclado. Se encuentra entre los dispositivos de entrada discreta, ya que es sensible a una posición concreta, en contraposición a los de entrada continua, que perciben un rango de posiciones. Es el dispositivo más indicado para la introducción de texto en cantidades no excesivamente grandes. En su diseño físico, en relación con la facilidad de uso, hay que tener en cuenta varios factores. Para ser manejable, debe tener poco grosor y estar algo inclinado (entre 10 y 25 grados), así la postura de las manos será más cómoda. También es importante su tamaño total y el de las teclas, las medidas idóneas reveladas en algunos estudios y recogidas por Shneiderman [12].

Otros datos de interés para la comodidad del usuario son la leve concavidad de las teclas que permite mejor la adaptabilidad de los dedos y las marcas en relieve de las teclas de la línea central correspondientes a los dedos índices (la *f* y la *j* en teclados *Qwerty*) para indicar la correcta colocación de las manos.

Además de las letras del alfabeto, un teclado de ordenador cuenta con otros tipos de teclas: para dar instrucciones básicas (retorno, borrar, otros), para combinar con otras (control, alternativa, otros), el bloque numérico, habitualmente con el orden de las calculadoras (7-8-9 en la fila superior) en lugar del que acostumbramos a ver en el teléfono (1-2-3 en la fila superior), las teclas de función y los cursores. Las teclas de función cuentan al mismo tiempo con la ventaja de reducir el error al teclear y con las desventajas de tener que recordar su significado y de obligar a mover los dedos de su posición en el teclado. Las aplicaciones modernas las han sustituido por combinaciones de teclas usando control, alternativa, mayúscula, etc.

b) Pantallas táctiles. Se trata de un dispositivo ventajoso y problemático a la vez, ya que sus peculiaridades lo hacen ser idóneo para unos usuarios y desaconsejable para otros. Nos referimos a su facilidad de uso (que no precisa aprendizaje y es altamente intuitivo e interactivo) frente a su falta de precisión, que gracias a los avances técnicos se está saldando y hoy contamos con pantallas capaces de captar un solo píxel.

También la incomodidad física de su utilización es un inconveniente, además de que produce manchas en la pantalla y precisa separar las manos del teclado. Están especialmente indicadas para contextos con usuarios que las van a usar en ocasiones puntuales y durante poco tiempo, como en los cajeros automáticos de los bancos, los puntos de información, etc.

c) Ratón. Este dispositivo indica partes de la pantalla de forma indirecta, por lo que no se produce tanta fatiga en el brazo, ni se oculta con éste la pantalla, pero se precisa una coordinación entre el ojo y la mano. El ratón permite mover el puntero por toda la pantalla, así como seleccionar una parte presionando alguno de sus botones. Por tanto, combina la entrada continua con la discreta. Su uso está ya tan difundido como el del teclado. Es muy aconsejable para tareas que requieran movilidad en la pantalla.

d) Reconocimiento del habla. De todos los sistemas de comunicación hombre-máquina, este es el más natural para las personas y el más fácil, ya que no precisa ninguna formación. El objetivo principal de crear sistemas capaces de reconocer el habla humana y de reproducirla es facilitar al hombre el trabajo con la máquina, hacerlo más rápido a la vez que se baja la tasa de error por introducción de datos por el teclado.

Resultan de gran utilidad cuando el usuario tiene las manos ocupadas, debe moverse o sus ojos están leyendo o controlando algo. Otra ventaja palpable es la que supone para las personas con discapacidades físicas que les impiden manejar el ordenador de la forma usual (teclado y ratón). La tecnología no ha avanzado lo suficiente como para ofrecer sistemas muy completos que admitan el discurso oral como forma de introducción de información en la computadora. Se encuentran dificultades cuando existe ruido de fondo, cuando el orador sufre alguna enfermedad que afecta a la voz o cuando dos palabras tienen una fonética similar.

Hasta hace poco tiempo las aplicaciones han estado limitadas a vocabularios pequeños y a unos usuarios determinados que deben leer algunas palabras para que el sistema capte sus características en el habla. Poco a poco comienzan a comercializarse productos que admiten discurso continuo y funcionan con diferentes usuarios sin necesitar un entrenamiento previo. La inteligencia artificial estudia cómo conseguir sistemas que resuelvan conflictos lingüísticos de forma inteligente en relación con el contexto [42, 43, 44, 45, 46].

e) Movimiento de los ojos y de la cabeza. Estos sistemas pensados especialmente para personas con discapacidades físicas en los miembros superiores son también útiles para quienes deban tener las manos ocupadas mientras utilizan la computadora. Consiste en una vídeo cámara conectada a la máquina que detecta hacia dónde mira el usuario e interpreta el significado. Existen dispositivos que son capaces de interpretar qué letra se desea

escribir con tan sólo fijar la mirada en un teclado preparado para ello. Resulta altamente complejo, ya que el movimiento de los ojos es demasiado rápido y las personas no siempre lo controlamos. Este tipo de comunicación de usuario con el sistema pertenece a la categoría de "interfaz basada en atención" [39, 40, 41], fuertemente estudiada en los últimos tiempos.

2.1.1.2 Dispositivos de salida

Son dispositivos destinados a brindar o mostrar la información deseada por los usuarios. Podemos distinguir dos tipos: visuales y sonoros. Aquí podemos distinguir un dispositivo de salida muy conocido "el monitor", en donde se va a plasmar gráfica o textualmente la información deseada por el usuario, el usuario recibirá los mensajes visualmente del computador a través del monitor; estos mensajes que pueden ser gráficos, textos, causaran un impacto en el cerebro del usuario de acuerdo a la forma en que estén diseñados. Ese impacto originará una interpretación en el usuario, dependerá en gran medida de la "forma de los mensajes". Hoy en día la mayoría de sistemas operativos soportan *softwares* que estén diseñados a reproducir sonoramente lo que textualmente está escrito. Por ejemplo, ya no es un problema que las personas invidentes puedan leer su correo electrónico, el computador puede reproducir con voz humana lo que está escrito en un *e-mail*, y de esta manera los usuarios invidentes pueden recibir sus mensajes sin ningún problema. A parte de ello, hay sonidos que identifican claramente una acción o quien no reconocería el sonido de inicio de Windows, no hay que visualizarlo simplemente bastaría con escuchar dicho sonido para saber que el sistema operativo Windows se ha iniciado normalmente (Esto ocurre cuando encendemos el computador y el sistema operativo Windows se ejecuta normalmente).

a) Visuales. Los aspectos físicos como el brillo, el contraste entre los caracteres y el fondo, la combinación de colores, el tamaño de los caracteres, los íconos y el orden de los ítems en un menú, y los aspectos técnicos como la resolución y el parpadeo producido por un lento refresco de pantalla, hacen que la lectura o visualización de la información sea más o menos cómoda. Una baja calidad en estos requisitos harán de las tareas algo fatigante para el ojo y, posiblemente, provocan una interpretación errónea de lo que percibe.

Las aplicaciones en tres dimensiones comparten esas características y añaden alguna otra, ya que son capaces de mostrar un objeto desde distintas perspectivas, con distintos enfoques de luz, distintas texturas, por lo que habrá que prestar mayor atención a sus

aspectos físicos y técnicos si se quiere que la interacción sea cómoda y no dé lugar a equívocos.

b) Sonoros. Esta forma puede resultar de gran utilidad si los sonidos representan determinados mensajes y el usuario conoce el significado de cada uno. Los sistemas operativos actuales incluyen una gama de sonidos que el usuario elige según la finalidad para los que los requiera: abrir un documento, eliminarlo, enviar un mensaje, avisar de un error. El efecto de estos sonidos en el usuario suele ser de satisfacción por la confirmación de una acción determinada. En entornos de trabajo en los que hay varias personas a menudo resulta molesto; en esos casos pueden utilizarse auriculares, si bien habrá que valorar hasta qué punto es incómodo llevarlos y qué medida resulta útil trabajar con sonidos. La utilidad es mayor en aplicaciones en las que el usuario debe tener la vista fijada en algún punto en concreto sea o no de la pantalla, desatendiendo por tanto el espacio donde pueden aparecer mensajes, o también en usuarios invidentes, que recibirán de este modo las contestaciones del ordenador a sus peticiones.

Otra forma de recibir información de la computadora por medios auditivos es el discurso. La capacidad de que una computadora reproduzca la voz humana se consiguió hace tiempo. Los primeros sistemas se usaban para repetir mensajes en un contexto muy determinado, como las máquinas expendedoras que dan las gracias por utilizar su servicio; el siguiente paso fue conseguir la lectura de textos, tan útil para las personas ciegas. Pero se trata sin duda de algo complicado, ya que el sistema debe poseer conocimientos similares a los que necesita para el reconocimiento del habla: estructura gramatical, contexto, significado de la entonación. Edwards [34] dedica un artículo a las características y posibilidades del discurso. Por el momento, la síntesis de voz está recomendada para la lectura de mensajes simples y cortos. Si es posible usar sistemas visuales, es preferible, ya que los sintetizadores de voz todavía no están preparados para reproducir discursos complejos y extensos.

2.1.1.3 Elementos de entrada/salida usados en el trabajo

En nuestro trabajo usaremos uno de los más populares dispositivos de entrada, el *mouse*. Actualmente, las interfaces de usuario diseñadas con objetos visuales son más sensibles a entablar una interacción usuario-computadora, con la ayuda de este dispositivo encontraremos los costos en términos de tiempo de recorrido de un objeto visual otro o distancia de estos objetos visuales de la interfaz gráfica de usuario.

El dispositivo de salida que lógicamente usaremos será la pantalla del computador que nos mostrará la interfaz gráfica de usuario mejorada con la metodología que estamos proponiendo como veremos más adelante.

2.1.2 Estilos de interacción

En un término genérico, se incluye todas las formas que los usuarios tienen de comunicarse con la computadora [12]. Los estilos de interacción más destacados son los sistemas de menús, lenguaje de comandos, lenguaje natural, etc.

a) Sistema de menús: Es el objeto más popular en el diseño del *software.* Cuando el *software* que se está desarrollando requiere múltiples opciones que el usuario debe elegir para el correcto procesamiento de la información, existen varios tipos de menús, podemos encontrar los menús desplegables, tanto en forma horizontal como vertical, menús diseñados en base a imágenes, esto lo podemos encontrar mayormente en los diseños web, acompañados de *hyperlinks*, este objeto fue una buena alternativa de solución para los diseños de *software* de mediana y gran magnitud, aparece como objeto en los lenguajes de programación visual, ya que antiguamente en los lenguajes estructurados se tenía que diseñar menús en base a opciones formadas por letras o números.

b) Lenguaje de comandos: Otra forma interactiva de comunicación (antigua, pero que en muchos casos son indispensables actualmente), el usuario ingresa vía teclado (mayormente), comandos definidos por el lenguaje de programación que se está usando en ese momento, los comandos en un lenguaje son una forma de comunicación directa entre el usuario y el computador, pero teniendo como enlace un dispositivo de entrada, los comandos son reconocidos como órdenes directas y depende mucho para un correcto funcionamiento que la sintaxis sea correcta, ya que basta con una letra inadecuada dentro del comando dará como resultado un mensaje de error mediante un dispositivo de salida (pantalla del computador).

c) Lenguaje natural: Es una aplicación de la IA (inteligencia artificial). La idea es que el usuario se pueda comunicar con el computador en su lenguaje natural, no usando los comandos de un programa. También la computadora puede generar lenguaje natural, para comunicarse con el usuario. Existen dos partes: reconocimiento de la voz y reconocimiento del lenguaje. Combinados permiten al computador comunicarse en su forma más natural.

Existen dos grandes métodos para procesar el lenguaje natural: análisis de palabras claves y análisis sintáctico semántico.

d) Manipulación de objetos: los objetos visuales de carácter gráfico representan un impacto visual en el usuario, por ejemplo colocar el gráfico de un disquete dentro de un botón de comandos representa la opción grabar sin necesidad de colocar el texto. Los objetos visuales acompañados de gráficos o imágenes tienen un impacto significativo en nuestro cerebro, tanto que nos brindan los pasos siguientes que debemos seguir para continuar interactuando con el computador. El caso específico de los *hyperlinks* en los diseños de páginas web, el solo hecho de ver un texto subrayado y al pasar sobre él, el cursor del mouse se transforma a un grafico de una "mano", nos incita al click derecho de manera inmediata. Hoy en día no podemos imaginar un menú sin su barra estándar que nos simplifique elegir las opciones más usadas dentro de una barra de menús, es imposible imaginar un diseño web sin imágenes y gráficos que nos brinden la intención principal de la pagina web, es por ello y por lo anteriormente expuesto es que los objetos de carácter visual han obtenido tanto auge en el diseño de las GUI. Esos objetos deben tener un significado de acción de lo que representa, ampliamente estudiado en la ingeniería semiótica [36].

2.2 Aspectos semióticos en GUI

La semiótica estudia los símbolos y su significado en la comunicación usuario-computadora, los aspectos culturales de los mismos y en nuestro trabajo trataremos como la semiótica nos ayuda al diseño y mejoramiento de las GUI.

Hacia la ultima década, numerosos investigadores en la materia han recalcado la importancia de incorporar una perspectiva semiótica a la IUC [24, 25, 26], más recientemente se han dedicado íntegramente a explorar el por qué y cómo la semiótica y la IUC deberían aproximarse mutuamente [27].

La mayoría de estudios semióticos comparten el objetivo de investigar los procesos de representación, interpretación y significación, que son el fundamento de lo que conocemos como comunicación. Este trabajo comparte este mismo objetivo y trataremos de utilizar la semiótica, para plantear una mejora en el diseño de las GUI, que es base en la comunicación usuario-computadora.

2.2.1 Semiótica visual

Es expuesto por los objetos o controles visuales y su real significado e impacto en el usuario. Estos controles visuales tienen que estar correctamente distribuidos, tener una simbología que realmente el usuario interprete lo que dicho objeto visual deba realizar para cumplir una tarea específica, de esa forma generamos una comunicación usuario-computadora mucho más eficiente. Podemos recalcar tres aspectos claves:

a) Organización

Se debe utilizar elementos simples, claros y consistentes; de esta manera, estamos fomentando una mejor usabilidad.

b) Economía

Maximizar la efectividad de un conjunto mínimo de elementos. Incluir elementos esenciales y no ambiguos. Elementos claramente diferenciables. Elementos u objetos distribuidos adecuadamente para lograr el impacto de comunicación deseado en el usuario, es aquí donde nuestro trabajo se va a centrar claramente.

c) Comunicación

Diseñar elementos para empatar con las características de los usuarios. Necesidades psicosociales, deseos educación, hábitos, etc.

2.2.2 Ingeniería semiótica y evaluación de la comunicabilidad

La ingeniería semiótica se concentra en la calidad y las posibilidades de comunicación en IUC. Su concepto fundamental es el de un artefacto de metacomunicación. El *software* interactivo es un tipo especial de artefacto comunicativo, uno que comunica sobre comunicación (por tanto, un artefacto de metacomunicación). Las interfaces de usuario son mensajes de un sentido (signos muy complejos) mandados de los diseñadores a los usuarios para decirles a éstos lo que han aprendido sobre los usuarios: cómo son, qué necesitan, qué les gusta, qué conocen, qué se ha hecho para los usuarios y por qué, y además como deben comunicarse los usuarios con el artefacto con el fin de conseguir un rango dado de objetivos.

El diseño de la GUI se ve desde la Ingeniería Semiótica como la actividad cuyo producto es el lenguaje artificial y sus frases, dichas en contextos interactivos se refieren a la visión del diseñador. Esta visión está motivada por un número de factores tales como oportunidades de mercado, estudios sobre usuarios, el talento y la creatividad de diseñadores y desarrolladores, entre otras cosas. En la Figura 2-2, se muestra una caracterización dramática de este proceso. El diseñador manda un mensaje único y unidireccional (súbito) a los usuarios, escrito por medio de varios códigos de computador. El mensaje existe y se desarrolla dentro del computador, al intercambiar los usuarios otros mensajes con el sistema, en un intento de lograr objetivos específicos. Y el significado completo de este mensaje súbito desde el diseñador a los usuarios es finalmente su visión de diseño completamente desarrollada.

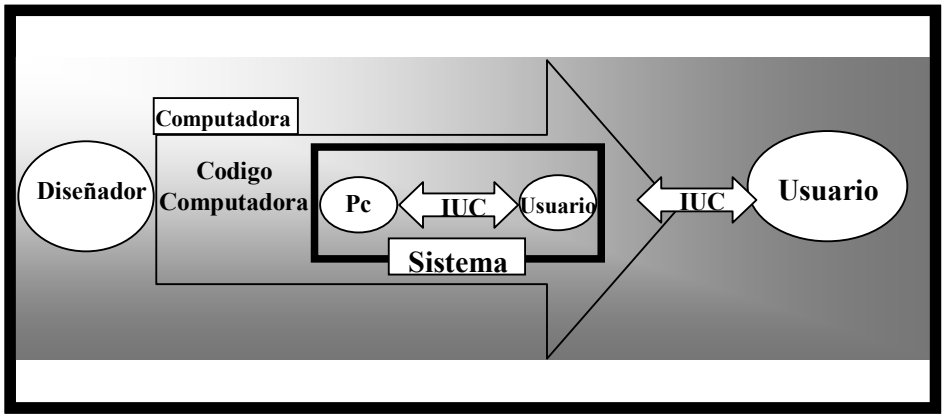


Figura 2-2: Diseño de la IUC desde la perspectiva de la Ingeniería Semiótica.

2.3 Principios fundamentales de una GUI

Una GUI para que cumpla los propósitos para el cual fue diseñada tiene que guardar principios fundamentales de tal manera que la GUI diseñada cumpla con el objetivo primordial que es la de establecer una comunicación hombre-máquina eficiente. Entre los principios fundamentales que debe guardar una GUI, encontramos los siguientes: autonomía, consistencia y eficiencia.

2.3.1 Autonomía

Una GUI debe permitir un correcto aprendizaje de la aplicación y que el grado de complejidad de la GUI sea la adecuada para una fácil comunicación usuario-computadora.

La computadora, la GUI y el entorno de trabajo deben estar a disposición del usuario. Se debe dar al usuario el ambiente flexible para que pueda aprender rápidamente a usar la aplicación. Sin embargo, está comprobado que el entorno de trabajo debe tener ciertas cotas; es decir, ser explorable pero no azaroso.

En la Figura 2-3, vemos un ejemplo de un ambiente complejo, que aunque puede ser explorado por el usuario, dificulta demasiado la comunicación entre la persona y el computador, debido a la complejidad del diseño de la GUI, este tipo de diseño termina por confundir al usuario, causando una considerable pérdida de tiempo y frustración en el usuario por no entablar una correcta comunicación con el computador.

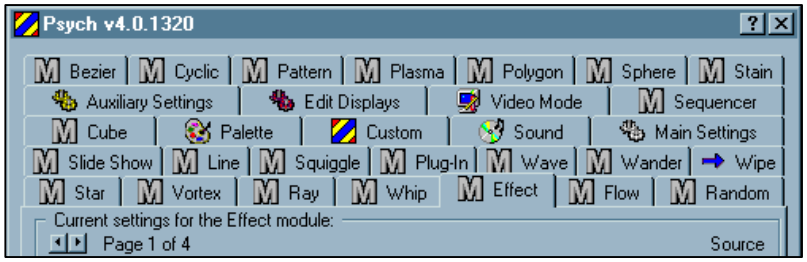


Figura 2-3: Ejemplo de ambiente complejo.

Es importante utilizar mecanismos indicadores de estado del sistema que mantengan a los usuarios alertas e informados. No puede existir autonomía en ausencia de control, y el control no puede ser ejercido sin información suficiente. Además, se debe mantener información del estado del sistema en ubicaciones fáciles de visualizar.

En la Figura 2-4, se ejemplifica una incorrecta disposición de componentes en la GUI. El reloj no debe ser incorporado en el menú del sistema ya que aporta confusión al usuario. Para mantenerlo informado sería más adecuado colocarlo en la barra de estado del sistema.

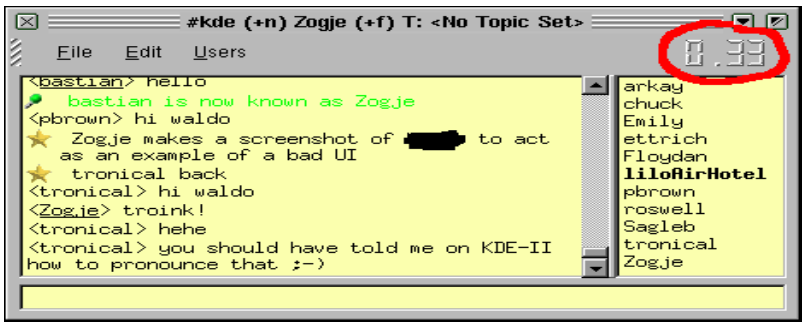


Figura 2-4: Ejemplo de información de estado inadecuada.

2.3.2 Consistencia

Una GUI debe permitir mantener estructuras visibles, fáciles de entender para el usuario, abreviación de comandos o barra de comandos para la ejecución de tareas, controles u objetos de ayuda que permita un rápido procesamiento de la información y respuestas al usuario en forma inmediata [4].

Para lograr una mayor consistencia en la GUI, se requiere profundizar en diferentes aspectos que están catalogados en niveles [1]. Se realiza un ordenamiento de mayor a menor consistencia:

Interpretación del comportamiento del usuario: la GUI debe comprender el significado que le atribuye un usuario a cada requerimiento. Ejemplo: mantener el significado de los comandos abreviados (*shortcut-keys*) definidos por el usuario.

Estructuras invisibles: se requiere una definición clara de las mismas, ya que sino el usuario nunca podría llegar a descubrir su uso. Ejemplo: la ampliación de ventanas mediante la extensión de sus bordes.

Pequeñas estructuras visibles: se puede establecer un conjunto de objetos visibles capaces de ser controlados por el usuario, que permitan ahorrar tiempo en la ejecución de tareas específicas. Ejemplo: ícono y/o botón para impresión.

Una sola aplicación o servicio: la GUI permite visualizar a la aplicación o servicio utilizado como un componente único. Ejemplo: la GUI despliega un único menú, pudiendo además acceder al mismo mediante comandos abreviados.

Un conjunto de aplicaciones o servicios: la GUI visualiza a la aplicación o servicio utilizado como un conjunto de componentes. Ejemplo: la GUI se presenta como un conjunto de barras de comandos desplegadas en diferentes lugares de la pantalla, pudiendo ser desactivadas en forma independiente.

Consistencia del ambiente: la GUI se mantiene en concordancia con el ambiente de trabajo. Ejemplo: la GUI utiliza objetos de control como menús, botones de comandos de manera análoga a otras GUI que se usen en el ambiente de trabajo.

Consistencia de la plataforma: la GUI es concordante con la plataforma. Ejemplo: la GUI tiene un esquema basado en ventanas, el cual es acorde al manejo del sistema operativo Windows.

En la Figura 2-5, puede observarse la mejora en la consistencia de las *pequeñas estructuras visibles,* para los sistemas gráficos basados en ventanas. La inclusión de la opción **X** para cerrar la ventana operación comúnmente utilizada en estas aplicaciones simplifica la operatividad del mismo. Teniendo encuenta que tipo de usuario va a operar la aplicación, el tipo de usuario principiante quizás no entienda que el botón con una **X,** sea el botón que cierre una aplicación. Pues semióticamente no representa la acción deseada.

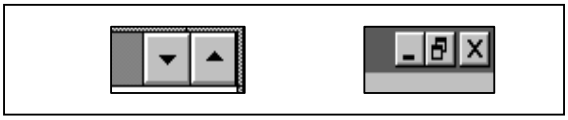


Figura 2-5: Ejemplo de consistencia.

La inconsistencia en el comportamiento de componentes de la GUI debe ser fácil de visualizar. Se debe evitar la uniformidad en los componentes de la GUI. Los objetos deben ser consistentes con su comportamiento. Si dos objetos actúan en forma diferente, deben lucir diferentes. La única forma de verificar si la GUI satisface las expectativas del usuario es mediante testeo.

2.3.3 Eficiencia de usuario

Cuando una GUI posee controles bien definidos, objetos visuales con etiquetas claras, menús de opciones fáciles de recorrer, botones con palabras claras del proceso, menús contextuales que facilitan el aprendizaje de la GUI, entonces esta GUI posee una buena eficiencia de usuario, ya que fácilmente hace prosperar una buena comunicación usuario-computadora y hace que el usuario tenga un desempeño más eficiente en el uso de la GUI.

Se debe considerar la productividad del usuario en el uso de una GUI, pues si el usuario demora un intervalo de tiempo considerable en realizar una tarea de comunicación con el sistema, al final de una sesión acumulará un tiempo mayor que se traducirá en pérdida de tiempo, generando pérdidas económicas para la organización.

En la Figura 2-6, se demuestra como una incorrecta definición de las palabras clave de las etiquetas de los botones de comando puede confundir al usuario. Los botones **OK** y **Apply**

aparentan realizar el mismo proceso. La similaridad o duplicidad de objetos de una interfaz también influye negativamente en la productividad del usuario, pues cada vez que el usuario quiera realizar una tarea específica no sabría que objeto usar.

Figura 2-6: Ejemplo de definición incorrecta de botones de acción.

2.4 Objetos de acceso a una interfaz

Existen una serie de objetos que afectan positiva o negativamente a una GUI. Estos son relativos a los:

- **mensajes**: deben tener todos el mismo aspecto y ser acompañados de sonidos y ventanas permanentes.
- **textos y gráficos**: los dibujos, fotografías y videos deben de ir acompañados de textos explicativos.
- **introducción de datos**: el texto escrito debe permitir recorrerse con el cursor para que los lectores de pantalla puedan leerlo.
- **redundancia del canal**: todos los textos y mensajes de sonido deben llevar asociado un sonido y varios colores para el tipo de letra, o bien, textos de alerta respectivamente. No es aconsejable que la entrada de datos sea redundante.
- **manejo del teclado**: se deben evitar que el usuario tenga que pulsar más dedos teclas a la vez y debe permitir explorar los menús de los programas.
- **servicios de ayuda al usuario**: es aconsejable incluir íconos descriptivos de las acciones que se van a realizar.

2.5 Eficiencia de interfaz

Cuando una GUI posee una cantidad de controles adecuados, no posee una complejidad innecesaria, resulta relativamente simple su aprendizaje y no distrae al usuario con controles u objetos visuales que no sean propios de la labor para la cual fue diseñada; entonces podemos decir que esta GUI posee una buena eficiencia de interfaz.

Para que una GUI posea una eficiencia de interfaz adecuada, debemos de respetar los siguientes principios de diseño:

- El interfaz debe compensar las limitaciones humanas, tanto físicas como cognitivas, siempre que sea posible. No obstante, tendría que ser "transparente", no ponerse en el camino de las acciones del usuario o impedir su progreso. Por otra parte, el interfaz no tendría que sobrecargar al usuario con complejidades innecesarias o distraerlo de su labor.
- Los componentes físicos del interfaz deben ser diseñados ergonómicamente[2], teniendo presente el confort y la salud del usuario tanto como sus necesidades.
- El interfaz debe ser consistente.
- El estilo de interacción como manipulación directa y menús es preferible al lenguaje de orden. Como mínimo, el usuario experimentado tendría que tener capacidad de moverse rápidamente a través de las capas de los menús.
- El interfaz puede tener acciones reversibles.
- El interfaz tendría que estar sujeto a pruebas al principio del diseño del proceso y durante su desarrollo.

El principio más básico del interfaz sería estar diseñado alrededor de las necesidades del usuario hasta después de que el sistema haya sido completado, atendiéndose de esta manera las restricciones impuestas por el sistema.

[2] Ergonómico: los periféricos de la computadora deben estar diseñados para que el trabajo del usuario sea confortable y saludable.

CAPÍTULO III: MÉTRICAS EN LA EVALUACIÓN DE LA GUI

Diremos que un producto es bueno o malo, y hasta sub-clasificarlo como aceptable, regular, óptimo, etc. dependiendo de algún indicador deducido a partir de las características del producto. En general, esos indicadores son numéricos, o indicadores de tendencias estadísticas, como también indicadores de eficiencia o complejidad en el uso del producto por los usuarios. En nuestro caso, el producto es una interfaz gráfica de usuario, que es un producto *software*.

Un elemento clave de cualquier proceso de ingeniería es la medición. Empleamos medidas para entender mejor los atributos de los modelos que creamos. Pero, fundamentalmente, empleamos las medidas para valorar la calidad de los productos de ingeniería o de los sistemas que construimos.

Cuando empleamos indicadores de medida, que pueden ser métricas caso de tratarse de operaciones de distancias, podemos saber que tan bueno es nuestro producto o que tanta cantidad tiene de una cualidad específica. Por ejemplo, si queremos saber que tan duro es el material con que construyen el chasis de los automóviles, tendríamos que tener una métrica que nos proporcione dicha información, diríamos para el ejemplo que el material tiene que tener un 60% de acero para que se considere aceptable, esto es una métrica, se le hace las pruebas correspondientes y obtenemos que el material solo tiene un 40% de acero por lo tanto este material solo es un 66.6% aceptable.

En el caso de la ingeniería de *software*, no tenemos medidas absolutas como la física que tiene voltaje, masa, velocidad, etc. En lugar de ello intentamos obtener un conjunto de medidas indirectas que dan lugar a métricas que proporcionan una indicación de la calidad de algún tipo de representación de *software*. Como las medidas y métricas del *software* no son absolutas, estas están abiertas a debate. Fenton [23] dice lo siguiente:

> *La medición es el proceso por el que se asignan números o símbolos a los atributos de las entidades en el mundo real, de tal manera que las definan de acuerdo con unas reglas claramente definidas.*

En el caso de métricas para una GUI, hay métricas que se basan principalmente en la aceptabilidad del usuario por ejemplo una métrica basada en el tiempo que requiere un usuario cualesquiera para aprender a usar la GUI, encuestas proporcionadas a los usuarios con preguntas tales como: ¿están claros los íconos?, ¿eran fáciles de recordar y de invocar las acciones?, ¿cuántas acciones diferentes han utilizado?, ¿resultaron fáciles de aprender las operaciones básicas del sistema? El resultado de la encuesta se evalúa en base a ciertos criterios y se obtiene un resultado cuantitativo.

En nuestro trabajo aplicaremos la métrica *diseño apropiado*, desarrollada por Sears [2], una valiosa métrica de diseño para interfaces usuario-computadora. Una GUI típica usa entidades de representación (íconos, gráficos, botones, etc.) para realizar una tarea. Esta métrica está basada en tareas de usuario, que explicaremos más adelante.

3.1 Distribución de los objetos en el diseño de una GUI

Consideramos el espacio del diálogo de la GUI como el plano cartesiano para localizar puntos en un espacio geométrico plano. El espacio del diálogo de la interfaz de usuario deberá ser transformada en una cuadrícula imaginariamente, en una grilla que permitirá posicionar en los diferentes celdas, los objetos de la interfaz.

La grilla, en esta ocasión, es definida de manera homogénea, pero con celdas no siempre cuadradas, de W x H como muestra en la Figura 3-1. Los números W como H son arbitrarios, pudiendo ser determinados en función de las dimensiones mayor o menor de los objetos, pues cada objeto distribuido en la grilla podrá ocupar una o más cuadrículas adyacentes, tanto horizontales como verticales, o combinación de horizontales y verticales, según sea la dimensión del objeto. Como ejemplo, en la Figura 3-1, se observa que el *Filename* ocupa ocho cuadrículas horizontales, mientras los botones OK y Cancel ocupan casi tres cuadrículas horizontales cada uno.

Para el propósito de este trabajo, consideramos como origen de la coordenada el vértice izquierdo inferior de la grilla, siendo numerable por números enteros, las líneas o columnas del lado izquierdo o inferior de la cuadrícula, respectivamente. Siendo así, los bordes izquierdo e inferior del diálogo son los ejes vertical y horizontal con numeración cero (0), los siguientes serán en sucesión progresiva: 1, 2, 3, etc.

Los objetos a ser distribuidos adecuadamente en la grilla, aparte de los aspectos semióticos, etiquetas textuales o símbolos representativos, deberán tener dimensiones geométricas, como lados, radio, ángulos, etc. de forma que sean simples de calcular sus centros o dimensiones geométricas apropiadas compatibles con la grilla. Sin perder la generalidad, para efecto de teste de nuestro método, sólo usaremos objetos de geometría rectangular.

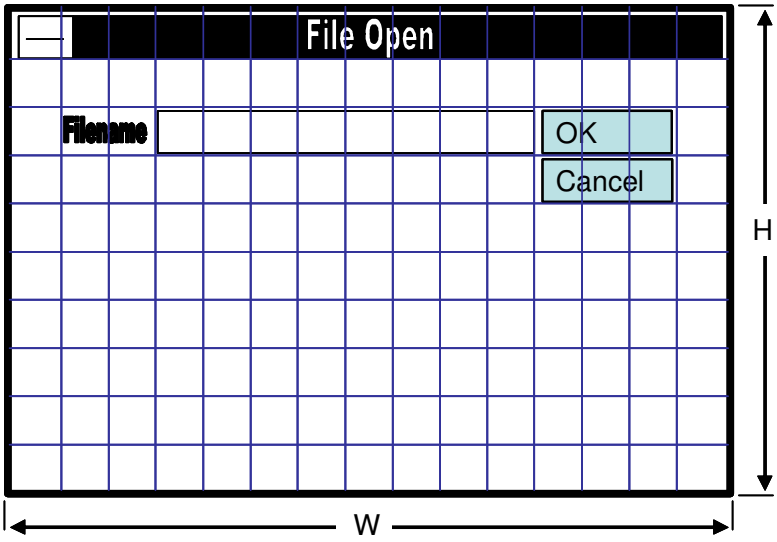


Figura 3-1: Diálogo de la GUI con grilla

3.1.1 Dimensiones de las celdas de una grilla

El valor de la celda de la grilla debe ser proporcional a las dimensiones de los objetos visuales, es por ello que debemos realizar un cálculo previo de las dimensiones de la celda para que los objetos visuales puedan posicionarse en dos o más celdas uniformes.

En la Figura 3-1, podemos observar tres objetos visuales posicionados sobre la GUI, en la cual mostramos al control *Filename* que esta posicionado sobre ocho celdas de ancho y una celda de alto, los otros dos botones (OK y Cancel) están posicionado sobre tres celdas de ancho por una celda de alto, como podemos observar estos objetos se encuentran posicionados en dos o más celdas pero de manera uniforme.

Para obtener las dimensiones de las celdas realizamos un MCD (máximo común divisor de las dimensiones de los objetos de la interfaz, tanto para su ancho como para su alto), de

esta manera, obtenemos una celda de dimensiones proporcionales a los objetos visuales sobre la superficie de la GUI.

Para simplificar los cálculos, asumimos los objetos de dimensiones proporcionales, de allí que obtenemos la celda de la grilla de dimensión 2 x 2, como muestra en la siguiente tabla (Tabla 3-1):

Objetos	Ancho	Alto
File	16	8
Directory	16	8
Filename	8	2
Botón guardar	4	4
Botón cancelar	4	4
MCD	2	2

Tabla 3-1: Dimensiones de los objetos.

3.2 Indicador de una GUI apropiada

Se dice que la interfaz gráfica es un diseño apropiado (DA), cuando los objetos están correctamente distribuidos sobre la GUI. La correcta distribución de los objetos es relativa, pues depende de la eficiencia en la comunicación usuario-computadora, indicado por una medida envolviendo rapidez de interacción y eficacia de comunicación de un tipo de usuario. Una forma de ver que una GUI es apropiada es minimizando el tiempo de desplazamiento de un objeto visual a otro a ser manipulado, o en todo caso se puede usar un costo basado en las preferencias de usuario.

La medida que nos indica, desde el punto de vista de eficiencia del usuario al interactuar con la GUI, está en función de varios elementos que el diseñador de la GUI debe considerar:

- objetos usados en la interfaz,
- secuencia de acciones que los usuarios deben realizar con los objetos,
- la frecuencia de las acciones realizadas,
- el costo que demanda cada acción realizada.

Los objetos usados en la interfaz deben ser explicitados. La Figura 3-2, por ejemplo, muestra una interfaz para abrir un archivo con cinco objetos: *Directory*, *Filename*, *Files*,

botón OK, botón Cancel. No se consideran objetos a los elementos de la interfaz que no producen acciones, como etiquetas, presentaciones, etc.

La secuencia de acciones que los usuarios deben realizar con los objetos, al interactuar con el computador en una sesión de comunicación con el sistema computacional, debe ser explicitada. Considere que una secuencia de acciones define un método de lograr una tarea, y puede ser descrita usando un diagrama de transición, como mostramos en la Figura 3-3 (para el ejemplo de la Figura 3-2) donde las etiquetas del acoplamiento entre los objetos visuales indican la frecuencia, o repeticiones, que se realiza la acción. Un ejemplo, con los objetos de la Figura 3-2, por primera vez, ir para *File*, para *Directory*, para *Filename* y para el boton Cancel. Las otras acciones son: de *File* ir para activar boton OK, de *Directory* ir para accionar *File*, de *Directory* ir para accionar *Filename*, y finalmente de *Filename* ir al botón OK. Una acción o una secuencia de acciones representa un método de realizar o lograr una tarea con el objeto (presionar, escribir, arrastrar, seleccionar, etc.).

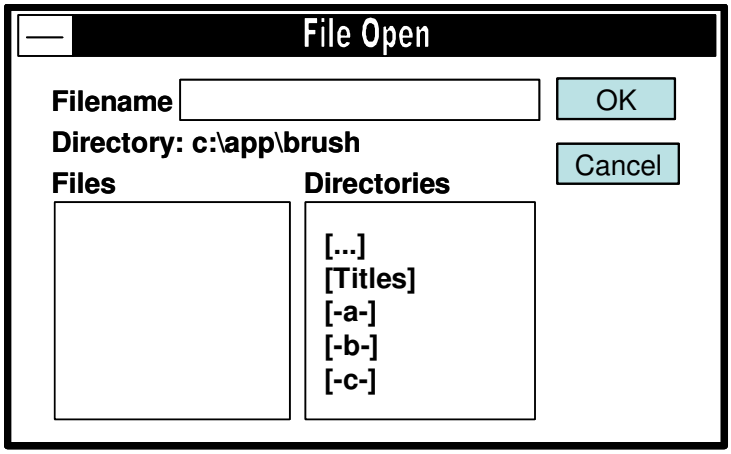


Figura 3-2: Interfaz de usuario para abrir un archivo.

La frecuencia de las acciones realizadas por los usuarios es el número de veces, número de repeticiones, que es realizada una determinada tarea en una sesión de comunicación con la máquina usando la interfaz. En la Tabla 3-2, se muestra la secuencia de siete tareas asociada al ejemplo ilustrado por la Figura 3-2, las frecuencias requeridas para cada tarea, la relación de pares de objetos usados parea realizar una tarea. Adicionalmente, consideramos la columna de valor de iteración. Por ejemplo, la tarea 1 involucra a los objetos *Directories* y *Filename*, con una frecuencia de 5, que significa que el usuario se

desplaza del objeto *Directory* para *Filename* o viceversa 5 veces durante la sesión. El objeto Inicio es un indicador del estado inicial en que el usuario puede encontrarse.

	Frecuencia	Valor transición	Objetos usados
1	5	5/108 = 0.05	Directory - Filename
2	25	25/108 = 0.23	Inicio - File
3	20	20/108 = 0.18	Inicio - Directory
4	2	2/108 = 0.01	Filename- Ok
5	5	5/108 = 0.05	Inicio - Filename
6	5	5/108 = 0.05	File - Ok
7	45	45/108 = 0.42	Directory – File
8	1	1/108 = 0.01	Ok - Cancel
T. sesión	108	1.00	

Tabla 3-2: Descripción de las tareas, la frecuencia y los objetos usados.

Al ejecutar todas las tareas, significa entablar una comunicación usuario-computadora y utilizar la GUI cumpliendo el propósito para la cual fue diseñada. Teniendo las tareas y la frecuencia con que estas se realizan, podemos formar un diagrama de transición, donde las etiquetas de acoplamiento estarán representando a los objetos involucrados, y en donde podemos observar los valores de transición calculados en la Tabla 3-2.

Los valores de transición más altos indicarán la mayor frecuencia entre los objetos involucrados. Hay que tener en consideración que justamente las tareas con mayor frecuencia son los puntos críticos en la redistribución de objetos sobre una GUI, quiere decir que debemos redistribuir de manera muy eficiente las tareas que contengan objetos con mayor frecuencia de uso para el usuario.

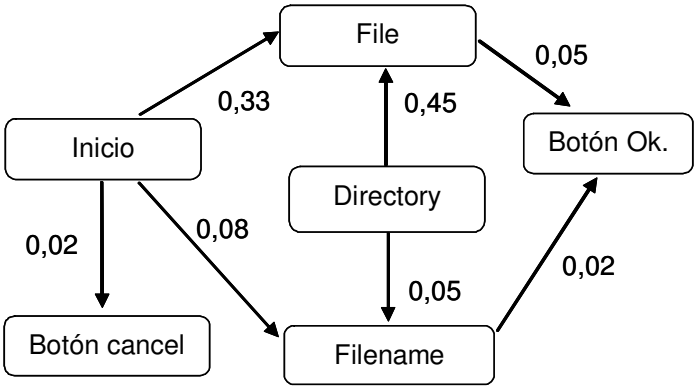


Figura 3-3: Diagrama de transición

En la Figura 3-3, podemos observar que la frecuencia más baja es 0,01, que pertenece a los objetos Inicio-*Directory*, la razón muy sencilla, porque *Directory* es el objeto que inicia la interacción usuario-computadora, sabiendo de antemano que el objeto Inicio es la posición donde se ubica el cursor del mouse, antes de iniciar la interacción del usuario con el computador por intermedio de la GUI.

Los valores de transición en la tarea i vt_i (3.1) son calculados como la frecuencia relativa.

$$vt_i = \frac{F_i}{\sum_{j=1}^{n} F_j}, \tag{3.1}$$

donde F_i es la frecuencia de la tarea i, y n es número de tareas de la sesión.

El costo de una tarea se fundamenta en su transición (recorrer de un objeto a otro para cumplir una determinada tarea), se puede relacionar con la distancia que el usuario debe recorrer con el mouse de un objeto a otro, o el tiempo que demora en realizar este recorrido, Paul Fitts, en 1954, demostró que el tiempo en alcanzar un objetivo está en función de la distancia que se recorre y del ancho del objeto, de ahí la famosa ley de Fitts [18]. Dicha ley la explicaremos detenidamente más adelante.

3.2.1 Costo de transición en una GUI

Una vez descrita la secuencia de acciones entre los objetos y las frecuencias de las tareas, se debe calcular el costo de esas tareas. Para ello, hallamos el costo de la transición y su respectiva frecuencia. La transición es la acción de recorrer un objeto a otro para cumplir una determinada tarea. Dicha tarea posee una acción o secuencia de acciones explicadas anteriormente. Entonces, dicho costo representa la dificultad para realizar la transición.

Si la frecuencia está dada por un número de transiciones, y cada transición tiene su costo constante, entonces el costo de una determinada tarea estará dado por el producto de la frecuencia por el costo de la transición: digamos que si se desea determinar el costo de la tarea k, entonces este será $F_k C_k$. El costo total de las tareas CosTar (3.2) será la sumatoria de los costos por sus frecuencias de todas las tareas que posee la GUI.

$$CosTar = \sum_{k=1}^{n} F_k C_k \tag{3.2}$$

El costo de transición puede estar dado por la distancia que debe moverse un ratón entre entidades o el tiempo de pasar de una entidad (objetos visuales) a otra con la ayuda del mouse. Para ser más específico nos basaremos en la ley de Fitts para encontrar el costo (distancia o tiempo de recorrido) entre objetos de dos dimensiones.

3.2.2 Costo de transición a través de la ley de Fitts

Hemos mencionado anteriormente que para obtener el costo (en términos de tiempo de trasladarse de un objeto a otro) de una secuencia de objetos, aplicaremos la ley de Fitts, que nos dice lo siguiente: "El tiempo en alcanzar un objetivo está en función de la distancia hacia el objetivo y del ancho del mismo", aplicando esta ley podemos decir que el tiempo MT (F3) requerido para seleccionar un objeto visual en una GUI está en función de la distancia que recorre el cursor del mouse de la posición inicial al objeto destino y del ancho del objeto destino.

Digamos que la posición inicial del cursor del ratón es P_0, y la distancia corta al centro del objeto O_1 sea A, cuyo ancho, en dirección del movimiento del cursor, es W (ver Figura (a)). Entonces, el tiempo demandado para activar el objeto O_1, según Fitts [14], estará dado por:

$$MT = a + b\log_2\left(\frac{2A}{W}\right) \tag{3.3}$$

donde los coeficientes a y b son determinados por métodos estadísticos [14], en particular por regresión y correlación de rectas y curvas, que aquí aplicamos en líneas abajo. Las unidades de MT está en segundos (o milisegundos), ya que a está en segundos (o milisegundos), b está en segundos por bits (o milisegundos por bits) y la variante del índice de dificultad $ID = \log_2\left(\frac{2A}{W}\right)$ esta dado en bits debido a la base binaria.

La formulación (3.3) proviene de las observaciones realizadas por Fitts basadas en diversas pruebas empíricas, y que el indicador de la dificultad en las tareas de usuario siempre es positivo.

Las unidades que maneja el índice de dificultad son bits (por eso se encuentra en logaritmo de base 2), ahora la pregunta sería, ¿porqué el índice de dificultad se encuentra en bits? La

respuesta es muy sencilla cuando trasladamos o movemos el cursor del mouse de un objeto visual a otro en una pantalla de la computadora, lo que está pasando en ese momento es que el CPU de la computadora está procesando la información referente a la distancia que está recorriendo el cursor del mouse sobre la superficie de la pantalla de la computadora, es por eso que el índice de dificultad está en unidades de procesamiento de información por parte del CPU de una computadora. Para que lo entendamos mejor, si quisiéramos aplicar la ley de Fitts para encontrar el tiempo que demora un misil al ser disparado desde una unidad militar hasta impactar en un blanco específico es muy probable que el índice de dificultad esté dado en kilómetros y no en bits.

Entre las variaciones a la ley de Fitts, está la propuesta por Welford [38] cuya formulación es la siguiente:

$$MT = a + b\log_2\left(\frac{A}{W} + 0{,}5\right)$$

y otra variación formula por Mackenzie [14], dada por la expresión:

$$MT = a + b\log_2\left(\frac{A}{W} + 1\right)$$

Estas variaciones en la formulación provienen por las siguientes razones:

- Observaciones realizadas a la ley de Fitts, realizando diversas pruebas.
- Obtener resultados positivos para el índice de dificultad en las tareas de usuario.

Podemos aplicar ejemplos simples y verificar qué formulación es la más adecuada para nuestro trabajo. Probemos con los siguientes valores: Se tiene un objeto de ancho (W) 2.46 cm y una distancia (A) de 1 cm. Para esa información, aplicamos las dos formulaciones y tendremos resultados tal como lo mostrado en la Tabla 3-3.

Método	**Índice de dificultad (ID)**	**Valor ID**
Welford	$\log_2(1/2.46 + 0{,}5)$	-0.14
Mackenzie	$\log_2(1/2.46 + 1)$	0.49

Tabla 3-3: Variaciones de Welford y Mackenzie

Observamos que Welford tendrá un índice de dificultad positivo si A/W >0.5, que indica que debe satisfacer la relación la distancia (A) y el ancho (W) tal como: $2A > W$. Caso

contrario será negativo. Mientras eso no ocurre con la formulación de Mackenzie que para cualquier valor de A como de W, el índice de dificultad siempre será positivo. Por lo tanto, la variación planteada por Mackenzie se ajusta mejor a nuestro trabajo en obtener índices de dificultad positivos para tareas de usuario usando objetos en dos dimensiones.

Bien, ahora hay una pregunta que deberíamos hacernos: ¿Cómo obtengo mis coeficientes de regresión a y b? Mackenzie realizó un test de viabilidad usando diferentes modelos para obtener los coeficientes de regresión. Estos modelos están netamente involucrados con las dimensiones del objeto destino, principalmente en el ancho del objeto destino (W) fundamentado en la ley de Fitts. Para realizar este test de viabilidad utilizó objetos en dos dimensiones con diferentes medidas para su ancho W y su alto H, a diferentes distancias y en algunos casos con ángulo θA, para la distancia A en que se encuentra un objeto del otro. La Figura 3-4 muestra algunos casos considerados, respecto a la línea de desplazamiento del curso de la posición inicial (origen del vector A) al centro del objeto de dimensión W x H. En el caso (a), el ancho W del objeto sigue siendo el mismo, mientras en el caso (b), el ancho a considerarse para los cálculos del índice de dificultad puede ser W' en función de W, H y el ángulo θA.

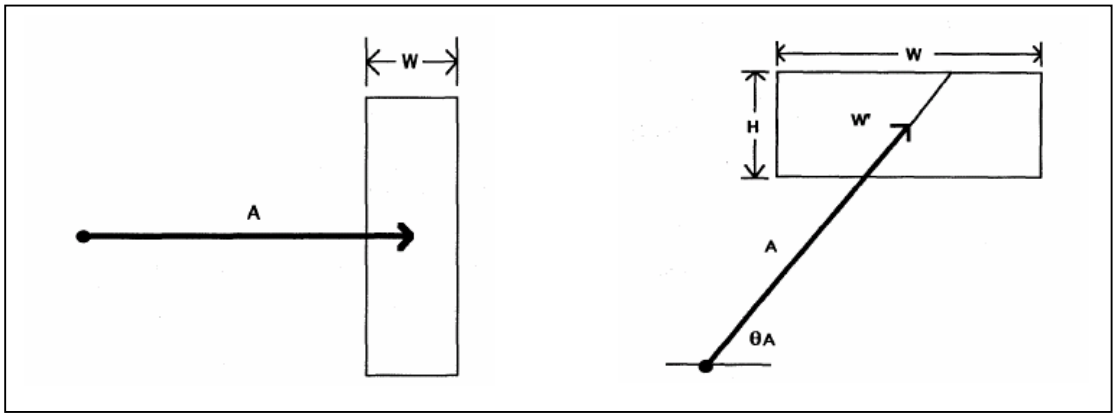


Figura 3-4: Algunos casos de objetos respecto a la línea de desplazamiento del cursor.

En algunos casos existe la posibilidad de sustituir el ancho del objeto destino por la dimensión más pequeña, quiere decir sustituir por el ancho o por el alto, dependiendo cual de estas dimensiones sea más pequeña, esta posibilidad es asequible dependiendo de la demanda de las tareas de usuario, que en muchos casos puede tener un número significativo.

Este modelo fue bautizado por Mackenzie como el "SMALLER-OF". Este es un modelo computacional simple, ya que sólo necesitamos conocer A (distancia hacia el objeto destino), W (ancho del objeto destino) y H (altura del objeto destino). El modelo W', requiere conocer A, W, H y θA (el ángulo de la distancia hacia el objeto destino). Este modelo se aplica en base a un cálculo geométrico y determinar la correcta sustitución para W.

Mackenzie realizó un test de viabilidad para determinar los coeficientes de regresión a aplicar con el índice de dificultad de la ley de Fitts. Este test de viabilidad lo realizó usando cinco modelos: dos modelos ya hemos mencionado, el SMALLER OF y el W'. El tercer modelo es el "STATUS QUO", donde W siempre va a ser la extensión horizontal del objeto destino. El cuarto modelo es sustituir el W del objeto destino por W+H. El quinto modelo fue bautizado con el nombre del modelo "AREA", ya que sustituimos el W del objeto destino por su área (WxH). Del test de viabilidad que realizó Mackenzie, se originó la Tabla 3-4, donde las columnas r^{∂} y SE^{b}(ms) son los indicadores estadísticos, índice de correlación y error estándar, respectivamente.

Modelo para el ancho del objeto	ID(rango en bits)		r^{∂}	SE^{b}(ms)	Coeficientes de regresión		
	Bajo	Alto			Intercepción a(ms)	Slope b (ms/bit)	IP (bits/seg.)
Smaller-Of	1,58	5,04	0,9501	64	230	166	6,0
W'	1,00	5,04	0,9313	74	337	160	6,3
W + H	0,74	3,54	0,8755	99	402	218	4,6
W x H	0,32	4,09	0,8446	110	481	173	5,8
Status Quo	1,00	5,04	0,8097	121	409	135	7,4

Tabla 3-4 Ceficientes de correlación y regresión para 5 modelos para el ancho del objeto destino.

Como podemos observar en la Tabla 3-4, el modelo SMALLER-OF posee una correlación obtenida del test de 0,9501 y un bajísimo error estándar 64 m, por lo tanto usando este modelo podemos obtener los coeficientes de regresión que estamos buscando a=230 ms. y b=166ms/bit, ms (milisegundos); por lo que la formulación de Mackenzie quedaría de la siguiente manera:

$$MT = 230 + 166\log_2\left(\frac{A}{W} + 1\right) \qquad \textbf{(3.4)}$$

Podemos formular un pequeño ejemplo utilizando la fórmula (3.4), si tenemos un índice de dificultad para una tarea de usuario de 5 bits. El tiempo que demoraría en completar la tarea seria: 230 + 166(5) ± 2(64)ms., quiere decir que el tiempo(MT) se estará posicionado dentro del intervalo <932 -1188>ms.

En la formulación, usamos 2(64)ms. porque según la Tabla 3-4 es el tiempo estimado de error estándar calculado para el modelo SMALLER-OFF; aparte de ello, este tiempo estimado de error estándar es calculado bajo dos coeficientes de regresión (a y b), es por ello que al usar la formulación de Mackenzie el tiempo estimado de error estándar es multiplicado por dos o en su defecto duplicado. Entonces diremos que en nuestro trabajo usaremos este modelo y la formulación planteada por Mackenzie para calcular el tiempo que demoramos en completar las tareas de usuario para una GUI.

3.2.3 Ejemplo de cálculo del costo de transición usando ley de Fitts

Sabemos que el costo de transición de una tarea está dado por la distancia que existe entre los objetos visuales que forman parte de esta transición o por el tiempo que demora el usuario en desplazarse de un objeto visual a otro, veamos un ejemplo de cómo se calcula el costo de transición usando la ley de Fitts. Tomaremos como modelo la GUI descrita en la Figura 3-1.

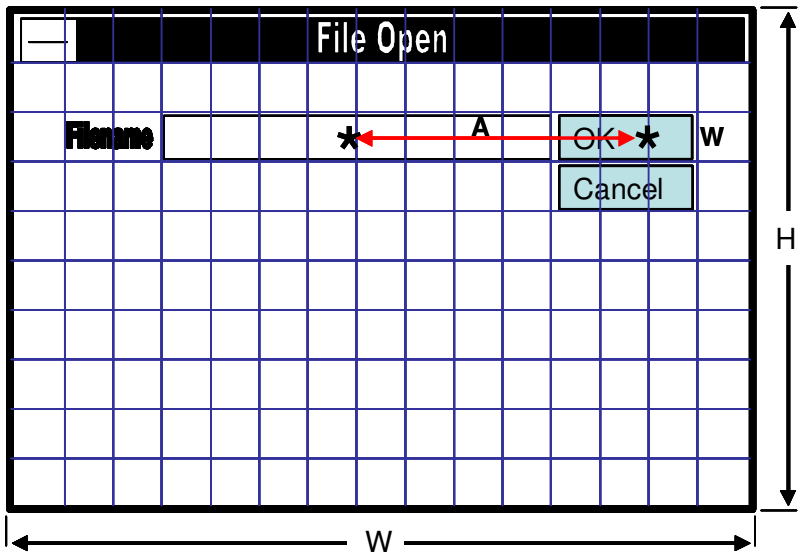


Figura 3-5: Los asteriscos nos muestran los centros de los objetos, la línea roja nos muestra la distancia (A), que existe entre ambos objetos y el ancho del objeto destino (W).

Tomando como referencia la Tabla 3-2, vamos a escoger la tarea 4, que involucra a los objetos *Filename* y botón OK. En esta tabla, podemos identificar que la tarea posee una transición cuya frecuencia es de 2.

Lo primero que debemos hacer es realizar el cálculo de la distancia(A), para lo cual se halla las coordenadas del centro de los objetos involucrados.

Si transformamos nuestra GUI en un plano cartesiano, que tiene como coordenada inicial (coordenada 0,0) a la esquina inferior izquierda de la GUI, entonces diremos que la coordenada superior izquierda del control *Filename* es (3, 8), por lo tanto la coordenada inferior derecha del control *Filename* es (11, 7), de esta manera con este par de coordenadas hallamos el punto medio del control *Filename* que sería (7, 7.5), asumiendo que cada celda de la cuadrícula tiene una dimensión de 1x1 cm.

Ahora tenemos que hallar el punto medio del botón OK, ubicamos sus coordenadas para hallar el punto medio de este control, las coordenadas serían (11, 8) y (14, 7), obtenemos el punto medio entre estas coordenadas y obtenemos como resultado (12.5, 7.5).

Ya hemos ubicado las coordenadas de los centros de ambos objetos *Filename* (7, 7.5) y el botón OK (12.5, 7.5), ahora hallamos la distancia(A), aplicando distancia euclidiana entre estos 2 puntos y obtendríamos como resultado 5.5 cm.

En la Tabla 3-2, nos dice que la transición es pasar del objeto *Filename* hasta el botón OK, por lo tanto el objeto destino es el botón Ok, seguidamente pasamos a aplicar la ley de Fitts, concretamente en la variación de Mackenzie, aplicando el modelo SMALLER-OFF, entonces el ancho (W) del objeto destino es de 1cm. Por lo tanto aplicando la fórmula quedaría de la siguiente manera: $MT=230 + 166\log_2(5.5/1 + 1)$, esto da como resultado 676,54 milisegundos, aquí le aplicamos el error estándar ±2(64) y obtenemos un tiempo estimado para realizar la transición, que se posiciona en el intervalo <550.2 -806.2>m; con esto hemos hallado un tiempo máximo de 806.2 m y un tiempo minimo de 550.2 m, que se tarda en realizar la transición de la tarea 4, pero hay que recordar que la tarea 4 tiene una frecuencia de 2, por lo tanto el costo de la tarea 4, expresado en tiempo de recorrer desde el control *Filename* hasta el botón OK, es de 1,612.4 ms como máximo y 1,100.4ms como mínimo, expresado en segundos sería aproximadamente 1.612 segundos el tiempo máximo que demoraríamos en completar la tarea 4. De esta misma forma calcularíamos los costos para todas las tareas restantes.

3.2.4 Métrica diseño apropiado

Cuando realizamos una redistribución de objetos sobre una GUI, es porque ya tenemos una GUI propuesta, una GUI que puede ser optimizada y que contribuya a una comunicaciónmás eficiente entre el usuario y el computador, pero podemos hacer una comparación entre la GUI propuesta y la GUI optimizada, de esta manera podríamos observar que tan eficiente era nuestra GUI propuesta.

Para estimar que tan eficiente era la GUI propuesta existe la métrica llamada "diseño apropiado".

Llamemos CRP al costo de la representación propuesta, que consiste en hallar el costo de la GUI propuesta en términos de tiempo aplicando la ley de Fitts y CRO que es el costo de la representación óptima calculado de la misma manera, pero con los objetos redistribuidos eficientemente, de esta forma la métrica del diseño apropiada se calcula con la fórmula F5.

$$DA = \frac{CRO}{CRP} \tag{3.5}$$

Esa relación permitirá saber si el diseño propuesto es apropiado o no, desde el punto de vista de la eficiencia del usuario. Si $DA \approx 1.0$ entonces diremos que el diseño propuesto es casi óptimo, lo ideal es que $DA = 1.0$ para que sea óptimo.

3.2.5 Otras métricas

Existen otras métricas muy interesantes que se utilizan para ver la eficiencia de una GUI, podemos considerar el tiempo de respuesta del sistema, una métrica también muy usada por los diseñadores de *software*.

3.2.5.1 Tiempo de respuesta del sistema

Para muchas aplicaciones interactivas, el tiempo de respuesta del sistema es el principal motivo de queja. En general, el tiempo de respuesta del sistema se mide desde el punto de vista que el usuario realiza una acción de control (por ejemplo, pulsar la tecla intro o pulsar el botón del ratón), hasta que el *software* responde con la salida o la acción deseada.

El tiempo de respuesta del sistema tiene dos características importantes: la duración y la variabilidad. Si la *duración* de la respuesta del sistema es demasiado larga, es inevitable

obtener como resultado la frustración y el estrés del usuario. Sin embargo, si la interfaz va marcando el ritmo del usuario una duración breve del tiempo de respuesta puede ser también perjudicial, ya que puede obligar a que el usuario se precipite y pueda cometer errores. La *variabilidad* se refiere a la desviación del tiempo de respuesta promedio, y en muchos aspectos es la característica más importante del tiempo de respuesta. Una variabilidad baja posibilita al usuario establecer un ritmo de interacción, aunque el tiempo de respuesta sea relativamente largo. Por ejemplo es preferible obtener una segunda respuesta de una orden a una respuesta que varié de 0,1 a 2,5 segundos. El usuario siempre estará desconcertado y preguntándose si ha ocurrido algo "diferente" detrás de la escena.

3.2.5.2 Encuestas de usuario

Es una métrica que se prepara en base a las preferencias de usuario, ya que toda GUI debe ser sometida a un continuo mejoramiento, quien mejor evaluador que el usuario que trabaja con ella. El usuario que produce la interacción usuario-computadora a través de la interfaz, es quien realiza una evaluación exhaustiva a medida que el tiempo de trabajo sea mayor.

Dichas encuestas o mejor dicho, las preguntas están basadas en temas específicos como los *servicios de ayuda al usuario*, en la cual se encuesta al usuario con preguntas sobre las ayudas que brinda el *software* para la correcta comunicación usuario-computadora, si el usuario al quedar entrampado en plena comunicación con el computador, el *software* le brinda "salidas o escapes", mediante un sistema de ayudas que ayudan de la mejor manera a que el usuario continúe con la interacción en forma normal. Otro tema que es considerado por los diseñadores es la *manipulación de información de errores* , muchas veces cuando el usuario realiza una acción que rompe la consistencia del *software*, el mensaje de error no brinda la suficiente información, entonces el usuario no sabe a que se refiere con ese mensaje; por lo tanto, ocurre un bloqueo en la comunicación hombre-máquina. Las preguntas que se obtienen de estos temas de acuerdo a la importancia de las mismas se les asigna un determinado peso, de acuerdo a la respuesta luego el puntaje obtenido es analizado y comparado con una tabla preestablecida y se obtiene un resultado que le sirve a los diseñadores que tan bien está su modelo.

3.3 Redistribución de objetos

Como mencionamos en una sección anterior, el área de la pantalla la dividimos en cuadrículas, esto se realiza para la redistribución de los objetos sobre la pantalla, cada

celda de la cuadrícula representa una posible posición de una entidad de la representación. Quiere decir que para una cuadrícula con N posibles posiciones y K diferentes objetos de representación, el número posible de distribuciones NPD (3.6) se representa como una combinación de K en N.

$$NPD = \frac{N!}{K!(N-K)!} \tag{3.6}$$

La Figura 3-6, claramente, ilustra una interpretación gráfica de la secuencia de acciones para abrir un archivo, en una GUI con una determinada distribución de los objetos. Observamos que la línea sólida representa la primera tarea de usuario, que el operario debe realizar para empezar la interacción con el computador. Las líneas punteadas representa la segunda tarea de usuario a seguir. Como podemos observar, redistribuyendo los objetos de la GUI, podemos obtener menor distancia de recorrido o menor tiempo de operación, cualquiera de estos representaría los costos de operar dicha GUI.

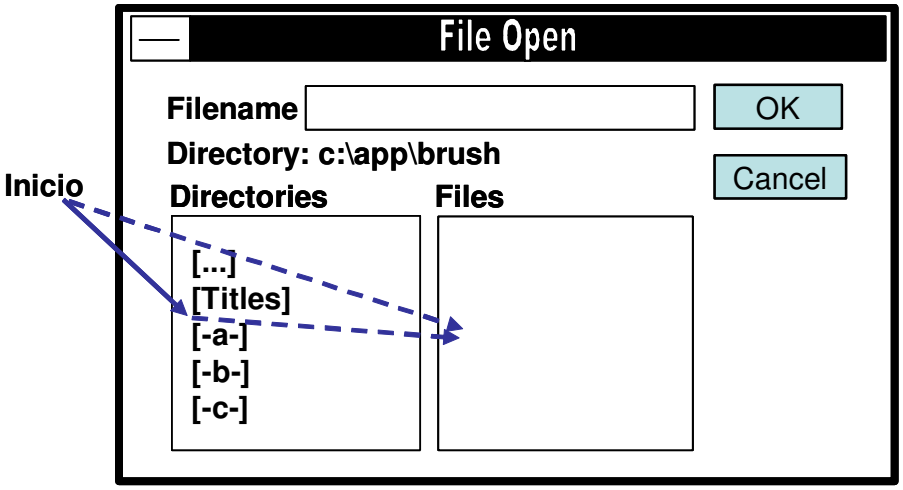


Figura 3-6: Visualiza secuencia de acciones.

En la Figura 3-7, podemos observar que se ha redistribuido 2 objetos visuales (*Directory* y *File*); con esto hemos logrado reducir los costos en términos de distancia recorrido o tiempo de recorrido que el usuario haría en una operación de abrir un archivo. Este es el objetivo principal de nuestro trabajo, redistribuir los objetos visuales reduciendo los costos en términos de distancia o tiempo de recorrido, de tal manera que hagamos la comunicación hombre-máquina más eficiente.

En ambas figuras tanto en la 3-6 como en la 3-7, podemos observar un objeto "Inicio", este objeto inicio representa la posición del cursor del mouse al momento de iniciar la interacción con la GUI, esto quiere decir que estamos asumiendo que al momento de entablar la comunicación con el computador por intermedio de la GUI, la posición del curso del mouse esta fuera de la GUI, asumimos este criterio para poder efectivizar una redistribución de los objetos visuales sobre la superficie de la GUI.

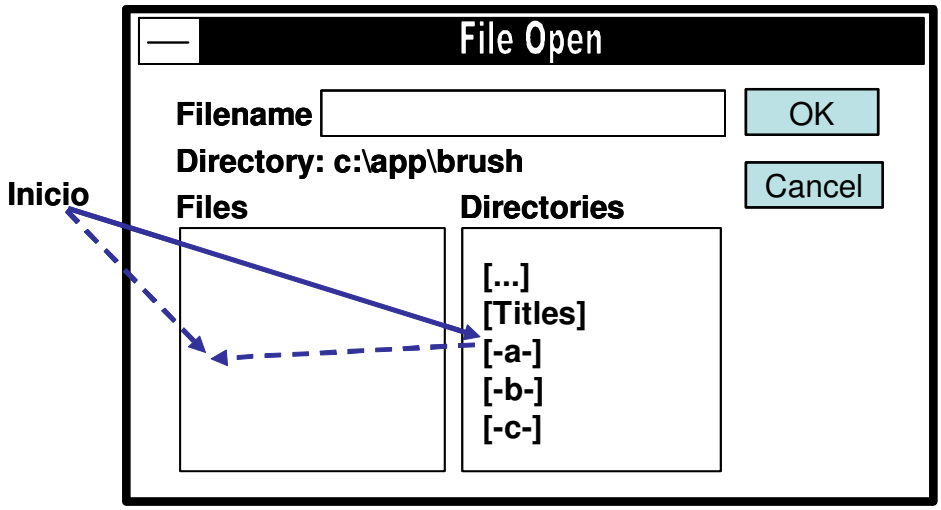


Figura 3-7: Una GUI con objetos redistribuidos.

En la evaluación de la GUI eficiente, se calculará los posibles posicionamientos de los objetos en el diálogo de la GUI en función de la grilla definida, como se explicó en una sección anterior, el número de distribuciones posibles se hace muy grande, como indica la fórmula F6, por ejemplo, cuando se tenga N = 12 y K = 5, el número de combinaciones seria 264. En casos reales, el número de cuadrículas de una GUI es relativamente grande y número de objetos no sea mucho, entonces la combinación es bastante elevada. Para optimizar la búsqueda y encontrar las mejores posiciones para los objetos visuales, se pretende usar un algoritmo goloso como primera instancia para luego utilizar un algoritmo GRASP, técnica eficiente usada en la inteligencia artificial. De manera diferente al algoritmo de búsqueda en árbol, utilizado por Sears [13], que genera recursividad y no aplica semiótica en la distribución de objetos.

En la Figura 3-8, se ilustra el comportamiento del algoritmo propuesto por Andrew Sears [13]. Cada nodo representa una distribución de objetos visuales sobre la grilla de la GUI con los posibles posicionamientos de un objeto por vez. Quiere decir que esta grilla esta

formada por 9 celdas que representan las posibles posiciones para los objetos visuales y 5 objetos visuales que ubicar. La celdas vacías están representadas por "0" y los objetos visuales por su respectiva numeración.

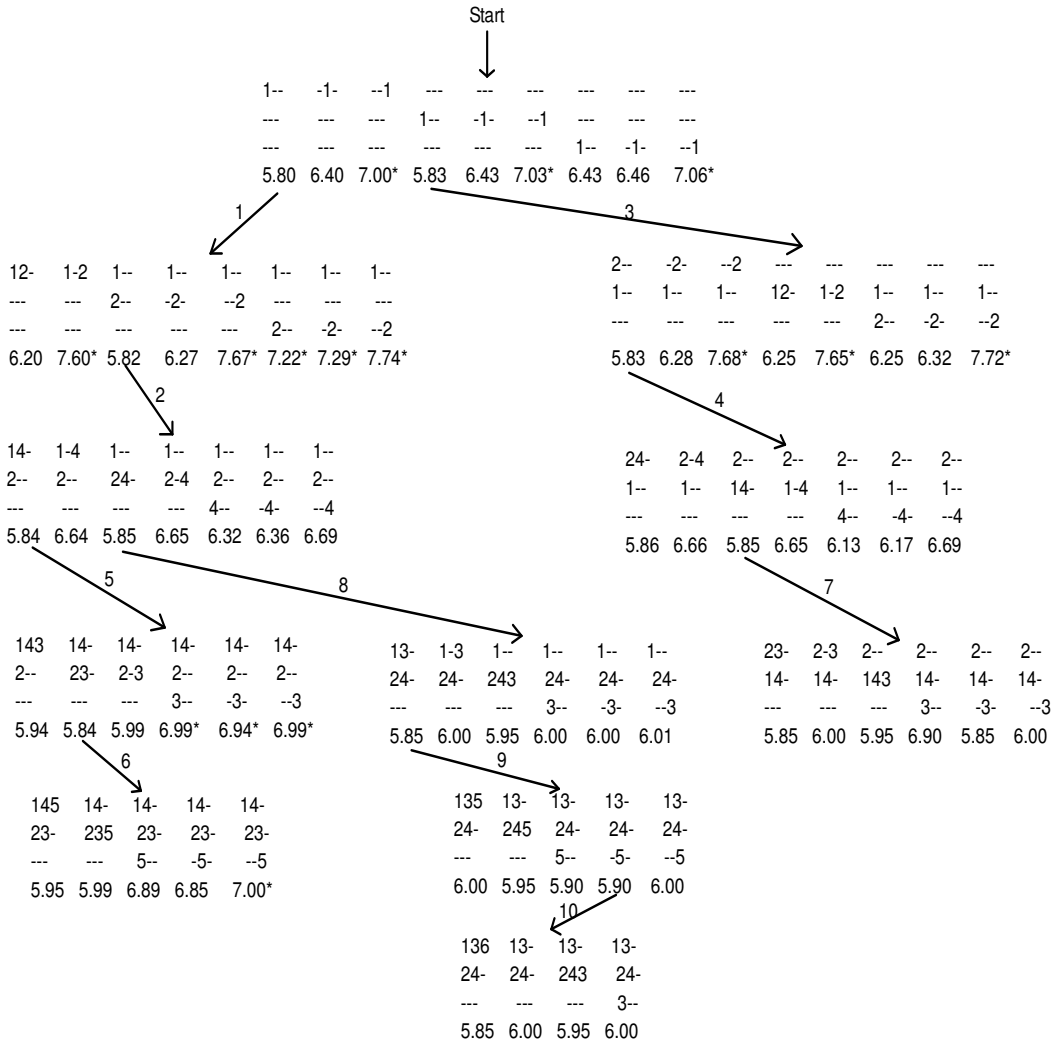


Figura 3-8: Muestra el árbol que se ha formado para encontrar la solución óptima del diseño una GUI, el costo mínimo final es de 5.85

Como podemos observar que Sears siempre toma el nodo con menor costo de distribución y comienza la búsqueda en forma de árbol de un menor costo de distribución de los objetos sobre la GUI. Lo que realiza es una "búsqueda en árbol", cada rama está formada por un conjunto de nodos, elige el nodo con menor costo y empieza la búsqueda de menor costo sobre ese nodo, redistribuyendo los objetos y recalculando los costos, lo que se puede observar también es que si en la búsqueda de hallar un menor costo sobre un nodo específico, advierte que en una rama anterior existe un nodo con menor costo, entonces

cambia la dirección de búsqueda y comienza a realizar la búsqueda en este nodo con menor costo, redistribuyendo nuevamente los objetos y recalculando nuevamente los costos de distribución.

Como podemos observar no utiliza una búsqueda en árbol con profundidad (no intenta llegar a la última hoja de la rama del árbol, entiéndase hoja a cada uno de los nodos), ya que siempre advierte ramas anteriores y si detecta un costo anterior menor en otra rama del árbol cambia el curso de su búsqueda, esto puede crear una deficiencia en el algoritmo, porque con mayores cantidades de objetos y una cantidad considerable de posibles posiciones para estos objetos dentro de la grilla, puede crear una mayor recursividad en la búsqueda de una distribución eficiente de los objetos visuales sobre una GUI, y por ende un mayor tiempo de procesamiento.

Podemos observar también que Sears no realiza bloqueo de celdas innecesarias utilizando semiótica visual, ya que siempre dentro de la grilla de la GUI van a existir posiciones "inutilizables" para algunos objetos visuales, utilizando semiótica visual podemos contribuir a mejorar la redistribución de los objetos desde el punto de vista de la aceptación del usuario. Por ejemplo no sería muy aconsejable que una barra de menús este en la parte inferior de una GUI.

CAPÍTULO IV: METAHEURÍSTICA EN LA EVALUACIÓN DE UNA GUI

En el capítulo anterior, observamos brevemente como funciona el algoritmo de Andrew Sears, para redistribuir los objetos visuales sobre una GUI de manera eficiente. También observamos algunas falencias que mostraba, ahora ¿por que deberíamos usar una metaheurística para obtener una eficiente distribución de objetos sobre una GUI, si existe ya un método exacto para lograrlo? Como sabemos en el método exacto de Sears, este trabaja con una cantidad bastante limitada de objetos visuales y sus objetos visuales solo ocupan una celda de la grilla de la GUI, esto quiere decir que con una cantidad limitada de objetos que ocupen una celda uniforme y pocas celdas donde colocar o redistribuir estos objetos, el método exacto funciona correctamente.

Pero no siempre vamos a encontrar GUI con pocos objetos y casi siempre vamos a encontrar objetos visuales que ocupenmás de una celda específica de la GUI, a parte de ello si queremos obtener mejores posiciones para los objetos dentro de la GUI, necesitamos contar con la mayor cantidad de celdas uniformes dentro de la GUI. Esto aumentará considerablemente las posibles posiciones para los objetos visuales. Es por ello que se hace necesario el uso de una meta-heurística para encontrar una distribución eficiente de los objetos, debido a la cantidad significativa de posibles distribuciones de los mismos. El uso de la meta-heurística nos ayudará a encontrar una distribución eficiente muy cercana al óptimo, pero que justifique su utilización debido a la complejidad del problema.

En nuestro trabajo de rediseño de una GUI mediante la redistribución de sus objetos visuales, la función objetivo (función a optimizar) estará dada en el tiempo que demora el operario en completar las tareas de usuario o la distancia de recorrido entre los objetos que forman la GUI, a lo que anteriormente en el Capítulo III, hemos denominado “costo de la GUI”. Ya que hablamos de optimización, entonces tenemos que tener dos criterios básicos: *maximización y minimización.* Lógicamente se pretende minimizar el costo de la GUI, para eso debemos minimizar las distancias entre los objetos visuales que forman la interfaz, de tal manera que resulte más óptima la comunicación hombre-máquina. Este proceso de optimización implica la modificación del diseño propuesto, tal como ocurre con las combinaciones de posicionamientos en la grilla mostradas en el Capítulo III, solo que esta

vez consideraremos un método metaheurístico para mejorar el modelo propuesto anteriormente, enfocado por Sears, usando árbol de reposicionamientos que, por tratarse del uso de heurísticas para tratar de resolver el modelo de optimización, el resultado obtenido será suficientemente bueno, tal vez próximo al óptimo.

La función objetivo debe minimizar el costo total de las transiciones entre los objetos en una determinada sesión de usuario-computadora. La formulación formal, en términos de modelo matemático, la presentamos en la Sección 4.3.

Se desea obtener una correcta distribución de los objetos visuales sobre un diálogo o *frame* a partir de la GUI propuesta, como vimos en el Capítulo III, tomando como referencia las celdas de la grilla definida imaginariamente sobre el *frame*. Habiendo mencionado, también, que existen las tareas de usuario, cada tarea esta formada por una secuencia que incluye trasladarse con la ayuda del mouse o algún otro dispositivo de entrada de un objeto visual a otro. Podemos deducir fácilmente que el "costo de estas tareas" estará dado por el tiempo que demora el usuario en recorrer la distancia entre los objetos que conforman una tarea específica.

Al tener un número significativo de celdas que forman la cuadrícula y una cantidad determinada de objetos, es de suponerse que podemos generar diversas posiciones para los objetos y por ende tener un número considerable de GUI con sus objetos redistribuidos. Por tal motivo, necesitamos encontrar la distribución suficientemente buena (lo ideal sería óptimo) de los objetos visuales, y para ello necesitamos un algoritmo exploratorio que nos permita obtener la distribución de objetos con el menor costo (en términos de distancia entre objetos o tiempo de recorrido de los mismos) posible.

La idea es ir generando diversas GUI, variando las posiciones de los objetos en la cuadrícula, cada redistribución de objetos obtendremos una nueva GUI con un determinado costo, justamente aplicando un algoritmo exploratorio podemos buscar la mejor disposición de los objetos visuales sobre la GUI, teniendo el criterio del menor costo de distribución de dichos objetos. Los objetos se irán posicionando uno a continuación del otro en las diversas celdas de la cuadrícula. Cada posición de objetos irá aumentando el costo de disposición de los mismos hasta completar el último objeto, al final obtenemos una GUI con sus objetos redistribuidos con un determinado costo.

4.1 Función heurística

Dada la dificultad práctica para resolver de forma exacta (simplex, "ramificación y acotación", teoría de grafos, etc.) toda una serie de importantes problemas combinatorios, es necesario ofrecer alguna solución para el cual comenzaron a aparecer algoritmos que proporcionan soluciones factibles (es decir que satisfacen las restricciones del problema), los cuales, aunque no optimicen la función objetivo, se supone que, al menos, se acercan al valor óptimo en un tiempo de cálculo razonable. Podríamos llamarlas en lugar de óptimas, "satisfactorias", pues al menos es de suponer que son lo suficientemente buenas como para servirnos.

Este tipo de algoritmos del cual vamos a hacer uso se denominan heurísticas, del griego *heuriskein*, que significa encontrar (palabra quizá no demasiada afortunada según apunta Reeves [28], dado que siendo más exacto, en principio lo que hacen es buscar). Aunque en un primer momento no fueron bien vistas en los círculos académicos acusadas de escaso rigor matemático [29], su interés práctico como herramienta útil que da soluciones a problemas reales, les fue abriendo poco a poco las puertas; sobre todo, a partir de la mitad de los años setenta con la proliferación de resultados en el campo de la complejidad computacional.

Una posible manera de definir estos métodos es como *"procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido"* [30].

La explicación clásica sobre el origen de las heurísticas reside en que ellas computan el costo de las soluciones exactas mediante una versión simplificada del problema original. Por ejemplo, en el juego de placas deslizantes (caso del juego de 4x4, con 15 placas y un espacio vacío), suponiendo violar o ignorar la restricción de solamente mover una placa hacia el espacio libre, se genera un nuevo problema más fácil, donde cualquier placa se puede desplazar a cualquier sitio vecino ocupado o no. Dado que se ha simplificado el problema removiendo la restricción de los movimientos, cualquier solución del problema original es, asimismo, una de las soluciones del problema simplificado. Se deduce que el costo de una mejor solución (puede ser la óptima) del problema simplificado pasa a ser el

límite inferior del costo de la mejor solución del problema original. De esa manera, la heurística derivada pasa a ser admisible[3].

Las soluciones heurísticas, las soluciones prácticas de un buen programador para el diseño de agentes que deben buscar en espacios de búsqueda y actuar como resultado de esa búsqueda, son una buena parte de la problemática de la inteligencia artificial. Son los algoritmos que resuelven problemas, partiendo de la epistemología de un dado dominio como punto de partida. Por ejemplo, la representación de las posiciones en el ajedrez.

4.2 Algoritmo goloso

Los algoritmos golosos miopes (del inglés myopic-greedy) son empleados para resolver problemas de optimización [21], cuyas estructuras se asemejen a la de un sistema de independencia. Estos poseen complejidad no polinomial (NP). Sus características habituales son:

- necesidad de brindar soluciones óptimas para un problema especifico,
- presencia de un conjunto o lista de candidatos (en nuestro ejemplo, los vértices y aristas de los árboles), y
- presencia de una función de mérito que determina si un conjunto de candidatos es solución al problema (siendo o no óptima) y de un criterio goloso que indique si es posible alcanzar una solución al problema añadiendo nuevos candidatos a la lista de candidatos, suponiendo que esta solución exista.

El nombre goloso–miope se debe a que, en cada paso, el algoritmo escoge el mejor candidato de la solución (pedazo) que es capaz de evaluar ("comer"), sin analizar las consecuencias de esa selección en el futuro o en el resto de la aplicación del método (“ver” más allá en el futuro). Nunca deshace una decisión o selección ya realizada: una vez incorporado un elemento de E (conjunto de candidatos) a S (solución), permanece ahí hasta el final; y cada vez que un candidato es rechazado, lo es para siempre.

[3] Se denomina admisible a toda heurística que estima un costo optimista (menor al real) de la trayectoria óptima del problema original.

Seguidamente damos una idea de un algoritmo goloso para un problema de minimización cualquiera, considerando que se tiene un conjunto de elementos E, un conjunto de condiciones F y una función objetivo f que se debe optimizar.

Goloso_miope_minimizacion (elementos E, condicion F, objetivo f)

1. Copiar elementos de E para un conjunto S
2. Mientras E tenga elementos, hacer
2.1. Seleccionar e = mejor elemento de E
2.2. Si S sin elemento e satisface la condición F, entonces
2.2.1 excluir elemento e del conjunto S
2.3. Excluir elemento e del conjunto E
3. Solución: S, con valores f(S).

Relacionado el problema de minimización de las distancias de transición entre los objetos en una GUI eficiente, que justamente sería la función objetiva, podemos fácilmente adaptar el algoritmo **Goloso_miope_minimización** para nuestro propósito.

4.3 Algoritmo goloso en la distribución de objetos de la GUI

La función objetivo será la distancia que existe entre los objetos visuales de una GUI, quiere decir que vamos a minimizar la distancia entre los objetos de la interfaz. Se va a redistribuir los objetos, colocando los objetos visuales en posiciones que mejoren la comunicación hombre-máquina y el usuario pueda utilizar la GUI de una manera mucho más eficiente.

Sea la secuencia de n tareas $J_1,\cdots,J_n$, con sus respectivas frecuencias $F_1,\cdots,F_n$, respectivamente, y un conjunto de m objetos $O=\{O_1,\cdots,O_m\}$ distribuidos sobre el *frame*. Con los elementos de O, se define un conjunto de n combinaciones $S=\{S_1,\cdots,S_n\}$, donde cada combinación $S_i=(O_{i1},\cdots,O_{ir_i})$ para $i=1,\cdots,n$, representa los r_i objetos que intervienen en una tarea que el usuario haría pasando de un objeto a otro, siendo $2\le r_i\le m$ porque la distancia entre objetos siempre se define entre cada par de objetos pudiendo envolver, en una tarea, todos los objetos de la interfaz. La Tabla 4-1 muestra lo descrito.

Tarea	Frecuencia	Objetos
J_1	F_1	S_1
…	…	…
J_n	F_n	S_n

Tabla 4-1: Descripción de las tareas, frecuencia y secuencia de objetos.

Con los elementos de la tabla anteriormente referenciada, se formula la función objetivo de la solución como:

$$f(S) = \sum_{i=1}^{n} F_i C(J_i)$$

donde $C(J_i)$ es el costo de la tarea, este costo es el tiempo que demora el usuario en recorrer la distancia entre dos o más objetos visuales que conforman una tarea.

Como podemos observar, el costo en términos de tiempo para las tareas de usuario está dado por el producto del tiempo que existe en recorrer dos o más objetos visuales por su respectiva frecuencia. La sumatoria de los costos de todas las tareas, nos dará como resultado el costo total de la GUI. El optimizar este costo con criterio de minimización al redistribuir los objetos sobre la GUI es lo que conocemos como nuestra función objetivo.

El tiempo entre los objetos de una tarea va a ser calculada mediante la ley de Fitts, formulado en el capítulo anterior, habiendo mencionado que cada secuencia estará formado por pares de objetos visuales, y que nuestra función objetivo es redistribuir justamente estos objetos sobre la GUI. Entonces, el costo total de una tarea $C(J_i)$ será la sumatoria de los tiempos entre los pares de objetos que el usuario haría durante la transición para completar la tarea J_i, siendo:

$$C(J_i) = \sum_{j=1}^{r_i - 1} C\left(O_{ij}, O_{i+1\,j}\right)$$

La redistribución de los objetos de la GUI, con criterio de minimización de los tiempos de recorrido de las distancias entre los objetos que conforman las tareas realizadas por el usuario, generará un impacto sobre el costo total del diseño de la GUI. Esto quiere decir que el impacto resultante de colocar o redistribuir todos los objetos visuales sobre la GUI resultará en el costo total (en términos de tiempo) de la interfaz.

Sabemos que nuestro trabajo se centra principalmente en el impacto que producen los objetos al ser colocados sobre una determinada posición sobre la superficie de la GUI.

Hemos mencionado también que nuestra función objetivo es la de reducir con el criterio de minimización el tiempo de recorrido entre los objetos de una GUI, por lo tanto el costo total (CT) de una GUI en términos de tiempo estará dado por la sumatoria del los mínimos costos (en términos de tiempo de recorrido) entre los objetos, para todas las tareas de usuario de una GUI, de esta manera obtendremos una GUI con menor costo y por ende optimizada para la comunicación con el usuario. Por lo tanto, podemos considerar lo siguiente:

$$CT = \sum_{i=1}^{n} \min(C(J_i))$$

Concluimos que el costo total de una GUI estará dado por los costos mínimos (en términos de tiempo de recorrido entre los objetos visuales que conforman una tarea) de colocar los objetos visuales en base a sus tareas de usuario sobre la superficie de una GUI. Habiendo expresado nuestra función objetivo y la forma como hemos establecido nuestro criterio para optimizar la distancia entre los objetos sobre una GUI, pasamos a establecer nuestro algoritmo goloso-miope para la distribución de objetos sobre la GUI, llamado **Goloso_miope_minimización_GUI** que recibe como argumentos el conjunto de objetos E, la matriz de grilla G, con las celdas G(x,y) que contiene objeto O_i y las tareas de usuario J_i. Cabe recalcar que para colocar un objeto visual sobre unas determinadas celdas de la grilla, estas tienen que estar desocupadas o disponibles.

Goloso_miope_minimizacion_GUI (objetos O, grilla G, tarea de usuario J, índice i)

1. Iniciamos conjunto de objetos $E = \{O_1, O_2, ...O_n\}$
2. Iniciamos conjunto solución en vacio $S = \phi$
3. Mientras E tenga objetos hacer

Obtener un conjunto de costos de ubicación del objeto O_i (de acuerdo a tareas de usuario J) en celdas disponible de la grilla $G(x, y)$

Seleccionamos objeto O_i y sus celdas ocupadas con menor costo de ubicación en la grilla $G(x, y)$.

Añadimos objeto O_i seleccionado y sus celdas ocupadas en la grilla $G(x, y)$ con menor costo de ubicación al conjunto solución $S = S \cup \{O_i\}$.

Excluimos el objeto O_i de E, $E = E - \{O_i\}$

4. Escribir solución S, con valores de $f(S)$.

En el algoritmo goloso-miope propuesto para la optimización de una GUI podemos observar el conjunto inicial formado por todos los objetos que forman la GUI, iniciamos el conjunto solución en vacío. Iniciando la iteración con el primer objeto visual, obtenemos un conjunto de costos de ubicación del objeto visual, de acuerdo a las tareas de usuario en las posiciones disponibles de la grilla.

Calculamos el tiempo de recorrido de un objeto visual a otro, de acuerdo a las tareas de usuario, usando la ley de Fitts, en las diferentes posiciones disponibles que ofrece la grilla o cuadrícula. Una vez calculado los diferentes costos (en términos de tiempo de recorrido), tomamos el criterio goloso-miope, seleccionamos el objeto y la posición del objeto (conjunto de celdas ocupadas por el objeto sobre la grilla) con menor costo de ubicación, lógicamente este objeto visual formará parte del conjunto de objetos asignados, seguimos iterando sucesivamente hasta completar la colocación de todos los objetos visuales sobre la GUI.

Para poder colocar un objeto visual sobre un conjunto de celdas específicas de la grilla, se tiene que realizar una validación previa. Esta validación consiste en considerar que este conjunto de celdas no estén ocupadas por ningún otro objeto visual y que este conjunto de celdas disponibles pueda contener, sin ningún problema, al objeto visual que queremos colocar sobre ellas.

4.4 Adaptación del Grasp en evaluación de una GUI

Tener una GUI bien diseñada nos proporciona una mejor comunicación hombre-máquina, al utilizar Grasp en el diseño de una GUI tendremos un conjunto de soluciones (diseños de GUI aleatorizados). Cada una de estas soluciones es un diseño propuesto de la GUI. Al utilizar Grasp, podemos encontrar la mejor solución, por ende el mejor diseño propuesto para una GUI con sus objetos mejor distribuidos.

4.4.1 Metaheurística Grasp

La palabra Grasp proviene de las siglas de *Greedy Randomized Adaptive Search Procedures* que en castellano sería algo así como: Procedimientos de Búsqueda basados en funciones "Greedy" Aleatorizadas Adaptativas.

Un Grasp es un método de multiarranque, en la cual cada iteración Grasp consiste en la construcción de una solución miope aleatorizada, seguida de una búsqueda local usando la solución construida, como el punto inicial de la búsqueda local. Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones Grasp se devuelve como la solución aproximada.

Los métodos Grasp fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimientos de conjuntos [31]. El término Grasp fue introducido por Feo y Resende [22] como una nueva técnica metaheurística de propósito general.

Grasp es un procedimiento iterativo que cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción, se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La mejor de todas las soluciones examinadas se guarda como resultado final.

En la fase de construcción se construye iterativamente una solución posible, considerando un elemento en cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial viene determinada por una función *greedy*. Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor. Note que esta medida es miope en el sentido que no tiene en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección, sino únicamente en esta iteración.

Se dice que el heurístico *greedy* se adapta porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial. Es decir, la evaluación que se tenga de añadir un determinado elemento a la solución en la iteración j, no coincidirá necesariamente con la que se tenga en la iteración j+1.

La heurística es aleatorizada porque no selecciona el mejor candidato según la función *greedy* adaptada; sino que, con el objeto de diversificar y no repetir soluciones en dos construcciones diferentes, se construye un lista con los mejores candidatos de entre los que se toma uno al azar.

Al igual que ocurre en muchos métodos deterministas, las soluciones generadas por la fase de construcción de Grasp no suelen ser óptimos locales. Dado que la fase inicial no garantiza la optimicidad local respecto a la estructura de entorno en la que se esté

trabajando (notar que hay selecciones aleatorias), se aplica un procedimiento de búsqueda local como post-procesamiento para mejorar la solución obtenida.

En la fase de mejora, se suele emplear un procedimiento de intercambio simple con el objeto de no emplear mucho tiempo en esta mejora. Notar que Grasp se basa en realizar múltiples iteraciones y quedarse con la mejor, por lo que no es especialmente beneficioso para el método el detenerse demasiado en mejorar una solución dada.

Las implementaciones Grasp generalmente son robustas en el sentido de que es difícil encontrar ejemplos patológicos en donde el método funcione arbitrariamente mal. Tal y como señalan Feo y Resende [22], una de las características más relevantes de Grasp es su sencillez y facilidad de implementación. Basta con fijar el tamaño de la lista de candidatos y el número de iteraciones para determinar completamente el procedimiento. De esta forma se pueden concentrar los esfuerzos en diseñar estructuras de datos para optimizar la eficiencia del código y proporcionar una gran rapidez al algoritmo; dado que este es uno de los objetivos principales del método.

A seguir presentamos el algoritmo Grasp genérico, tal como lo explicamos anteriormente, para un conjunto de elementos de problemas E, conjunto de elementos que verifican la condición del problema F, función objetivo (función a optimizar) f y un parámetro de relajación alfa que está entre 0 y 1.

Algoritmo GRASP (Elementos E, condición F, objetivo f, relajacion alfa)

1. Mientras condición de parada permite, hacer
1.1. GRASP_Construcción // solución
1.2. GRASP_Mejoría // mejora la solución
1.3. Registrar la mejor solución
2. Retornar la mejor solución.

Como podemos observar el algoritmo Grasp consta de dos parte importantes para convergir en la mejor solución: la construcción (Grasp_Construcción) y la mejoría (Grasp_Mejoría). La mejoría es la mejora de la solución que la construcción escoge como solución, lo que indica que este tipo de metaheurística está diseñada a encontrar de manera persistente la mejor solución. Teniendo esta explicación previa a seguir, formulemos una Grasp_Construcción con criterio de minimización.

4.4.1.1 Descripción del Grasp_Construcción

El algoritmo Grasp_Construcción inicia un conjunto solución (S) y un conjunto mejor solución (MS) en nulo, aparte de un conjunto de elementos E, luego inicia la iteración con el conjunto de elementos E, forma el $\beta_{\min}$ con el mejor elemento de E y el $\beta_{\max}$ con el peor elemento de E; a partir de ahí forma el conjunto de candidatos RCL (conjunto de elementos de E, que se encuentran dentro del intervalo), luego elegimos un elemento al azar del conjunto RCL, si ese elemento al azar unido al conjunto solución S satisface las condiciones del problema, será añadido al conjunto solución; al final, acabamos la iteración formando un conjunto solución S, ahora si esta solución S es mejor que la solución almacenada en MS (mejor solución), entonces S pasará a ser MS. A continuación mostramos el algoritmo Grasp_Construcción.

GRASP_Construcción (Elementos E, Condicion F, objetivo f, Relajación alfa)

1. Iniciar con NULO el conjunto solución S
2. Iniciar con NULO el conjunto mejor solución MS
3. Mientras condición de parada permita
3.1. Mientras existe elementos en E, hacer
3.1.1. Sea $\beta_{\min}$ mejor candidato $\{f(e) : e \in E\}$
3.1.2. Sea $\beta_{\max}$ peor candidato $\{f(e) : e \in E\}$
3.1.3. $RCL = \{e \in E : \beta_{\min} \le f(e) \le \beta_{\min} + \alpha(\beta_{\max} - \beta_{\min})\}$
3.1.4. $e_k = Random(RCL)$ // escogemos un candidato aleatoriamente
3.1.5. Si $S \cup \{e_k\} \in F$, entonces
3.1.6. $S := S \cup \{e_k\}$ // adicionamos candidato al conj. S
3.1.7. $E := E - \{e_k\}$ // excluimos candidato de E
3.2. Si S es mejor solución que MS, entonces
3.2.1. $MS := S$ // almacenamos mejor solución
4. Escribimos MS

En el algoritmo Grasp_Construcción, definimos un conjunto de posibles soluciones RCL, en función de $\beta_{\min}$, $\beta_{\max}$ y α.

Como podemos observar, hemos construido un algoritmo Grasp construcción (**Grasp_Construcción**) con criterio de minimización para un propósito general y el Grasp mejoría (**Grasp_Mejoría**) para mejorar la solución, que muchas veces se origina a partir del

análisis puesto en la solución encontrada por el Grasp_Construcción. En el capítulo siguiente formularemos nuestro algoritmo Grasp_Construcción para la redistribución de objetos sobre una GUI, así como nuestro algoritmo Grasp_Mejoría.

4.5 Optimización de distribución de objetos utilizando Grasp

Ya tenemos una idea concreta de lo que es la metaheurística Grasp. Ahora la idea principal es adaptarla a nuestro trabajo de redistribución de objetos sobre una GUI. Como ya hemos visto anteriormente en el algoritmo goloso-miope expuesto, nosotros cuando iniciamos la iteración, elegimos el primer objeto de la GUI, lo ubicamos en las diferentes posiciones libres de la GUI y calculamos el costo de colocación expresado en términos de tiempo de recorrido. Una vez que hemos calculado el costo de cada posición, elegimos por criterio goloso de minimización la posición con el menor costo.

En Grasp, ya no elegimos la posición con menor costo, sino que armaríamos un grupo de candidatos posibles, utilizando un parámetro de relajación. Este grupo de candidatos posibles lógicamente será un grupo de posiciones de ubicación del objeto visual sobre la GUI, de este grupo elegimos uno al azar, diferente al criterio goloso que era la elección mediante el de menor costo. Seguimos iterando hasta terminar el total de objetos sobre la interfaz. Como hemos explicado anteriormente, el costo total de la GUI sería la sumatoria de todos los costos de las tareas de usuario. Una vez que hayamos logrado colocar aleatoriamente todos los objetos de la interfaz, tenemos una primera solución con un determinado costo. Esta primera solución es almacenada temporalmente y nuevamente volvemos a iterar hasta colocar nuevamente y aleatoriamente todos los objetos de la GUI y encontramos una segunda solución. Si esta segunda solución es mejor que la primera, entonces esta será seleccionada como la mejor solución. Seguimos iterando nuevamente hasta hallar otra solución mejor que será seleccionada como mejor solución. La pregunta es ¿cuándo terminamos de iterar?, lógicamente para responder a esta pregunta, tenemos que colocar en nuestro algoritmo Grasp una condición de parada, que muchas veces puede ser un número determinado de iteraciones o cuando ya el algoritmo no encuentra una mejor solución.

El hecho de usar Grasp es sencillamente encontrar una mejor solución en la redistribución de objetos sobre una GUI, de esta manera, estaríamos obteniendo interfaces gráficas de usuario con mejor diseño y por ende mejorando la interacción usuario-computadora.

La metaheurística Grasp nos permite mejorar la redistribución de objetos sobre una GUI por las siguientes razones:

- La Grasp permite generar un conjunto de candidatos posibles. Para nuestro trabajo las posibles posiciones en los cuales se puede colocar un objeto sobre la superficie de la GUI utilizando un parámetro de relación (α), desechando los peores candidatos (peores posiciones posibles), mejorando el tiempo de procesamiento.
- Podemos ir generando muchas posibles soluciones, una mejor que la otra, utilizando condiciones de parada. En nuestro trabajo, podemos decir que podemos ir generando diversas GUI, con diversas posiciones de los objetos sobre su superficie, todas con un determinado costo, lógicamente siempre guardando o reservando la mejor.
- Al terminar la primera etapa del Grasp, o sea el Grasp_Construcción, hemos generado una muy buena solución, esto quiere decir que hemos generado una redistribución de objetos sobre una GUI de manera muy eficiente.

Aquí no termina todo, podemos buscar la mejora de la solución, utilizando la segunda parte del algoritmo el Grasp_Mejoría. Esto es como ya hemos mencionado la mejora de la solución. Quiere decir que aún podemos optimizar más nuestra distribución de objetos sobre una GUI; para ello, podemos usar un análisis sobre la distribución obtenida del Grasp_Construcción y ver la manera cómo mejorar esa distribución de objetos o utilizar otra heurística si fuera el caso, por ejemplo algoritmos genéticos. Por lo anteriormente expuesto, podemos considerar el algoritmo Grasp_Construcción, seguidamente propuesto para la redistribución de objetos visuales sobre una GUI:

Grasp_Construcción_GUI (Objetos O, grilla G, tarea J, i, relajacion alfa)

1. Iniciamos conjunto solución en vacío $S = \phi$
2. Iniciamos conjunto mejor solución en vacío $MS = \phi$
3. Mientras condición de parada permita
3.1. Iniciamos conjunto de objetos $E = \{O_1, O_2, ...O_n\}$
3.2. Mientras E tenga objetos hacer
3.2.1. Obtener un conjunto de costos de ubicación del objeto O_i (de acuerdo a tareas de usuario J) en celdas disponible de la grilla $G(x, y)$
3.2.2. Seleccionamos el objeto O_i y sus celdas ocupadas con menor costo de ubicación en la grilla $G(x, y)$, a la que llamaremos $\beta_{\min}$

3.2.3. Seleccionamos el objeto O_i y sus celdas ocupadas con mayor costo de ubicación en la grilla $G(x, y)$, a la que llamaremos $\beta_{\max}$

3.2.4. Iniciamos nuestro conjunto de candidatos RCL, tal que $RCL = \{e \in E : \beta_{\min} \leq f(e) \leq \beta_{\min} + \alpha(\beta_{\max} - \beta_{\min})\}$

3.2.5. Seleccionamos del conjunto RCL, el objeto O_i con sus celdas ocupadas dentro de la grilla $G(x, y)$ y su respectivo costo de ubicación, al azar

3.2.6. Añadimos objeto O_i seleccionado y sus celdas ocupadas en la grilla $G(x, y)$, su respectivo costo de ubicación al conjunto solución $S = S \cup \{O_i\}$

3.2.7. Excluimos el objeto O_i de E, $E = E - \{O_i\}$

3.3. Si S es mejor solución que MS entonces

3.3.1. $MS = S$

4. Escribir mejor solución MS

4.6 Prueba y resultados del Grasp_Construcción en una GUI

Para iniciar las pruebas y resultados con el Grasp_Construcción, debemos considerar una GUI propuesta por los diseñadores. Esta GUI propuesta es la que se va a someter al rediseño, redistribuyendo los objetos visuales que sobre ella se han colocado, hay que indicar claramente que el diseñador de la GUI propone las tareas de usuario. Las tareas de usuario, la frecuencia de éstas y los objetos visuales que involucran estas tareas fueron explicados anteriormente, lo que el algoritmo Grasp_Construcción va a realizar es redistribuir estos objetos visuales de tal manera que obtengamos un diseño de interfaz de GUI más eficiente para generar pruebas y resultados, utilizando Grasp_Construcción podemos considerar la siguiente GUI de la Figura 4-1.

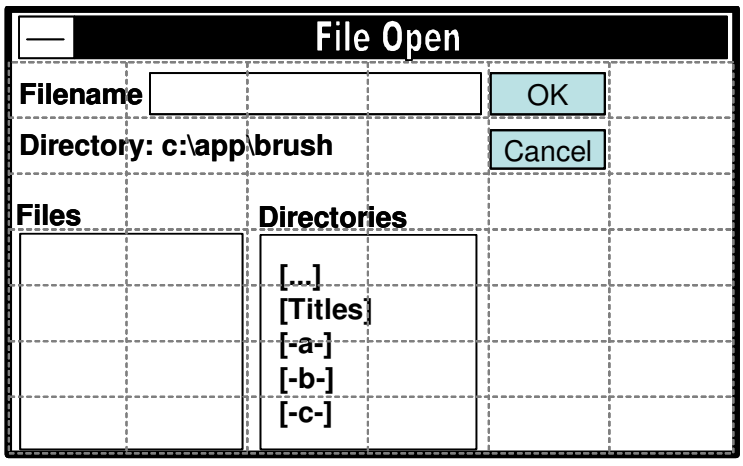


Figura 4-1: Interfaz de GUI propuesta por el diseñador.

Como podemos observar en la Figura 4-1, la GUI propuesta por el diseñador ha sido dividida en celdas uniformes. Estas celdas forman una grilla. Las celdas de esta grilla, en este caso tienen una dimensión de 2 cm de ancho y 1 cm de alto. La grilla esta formada por seis celdas a lo ancho y siete celdas a lo alto. También observamos que los objetos visuales ocupan una determinada cantidad uniforme de celdas, esto nos facilitara que al ser sometida la GUI al algoritmo Grasp_Construcción la redistribución de sus objetos sea más eficiente.

Recordemos la tabla de tareas de usuario en que los objetos involucrados son representados con sus respectivas frecuencias, ya que de ello va a depender el cálculo de los costos de dichas tareas en términos de tiempo de recorrido de los objetos visuales que involucran estas tareas.

Tarea	**Frecuencia**	**Objetos Usados**
1	5	Directory - Filename
2	33	Inicio –File
3	2	Inicio – Cancel
4	2	Filename, OK
5	8	Inicio - Filename
6	5	File – OK
7	45	Directory – File
T. sesión	100	

Tabla 4-2: Tareas de usuario, objetos y sus frecuencias.

Con la Tabla 4-2, podemos calcular los costos para las diferentes tareas de usuario. Cabe indicar que podemos realizar pruebas con el Grasp_Construcción, utilizando el valor calibrado de alfa (α)=0.2. La calibración de este valor se realiza, ejecutando el algoritmo con un determinado número de corridas (100, 500, 1000, 2000, etc.) y con distintos valores para alfa (0.1, 0.15, 0.2, 0.3, etc.); de esta manera, seleccionamos el mejor valor para alfa, para nuestro caso fue 0.2, por tal motivo podemos generar la Tabla 4-3.

Objetos	Tareas de usuario	Celdas en grilla		Corridas	Costo	
		Ancho	Alto		Miliseg.	Seg.
5	8	1cm.	1cm	100≤C≤1,000	51,870	51.87
5	8	1cm.	1cm.	1,000<C≤10,000	51,753	51.75
5	8	1cm.	1cm.	10,000<C≤20,000	51,672	51.67

Tabla 4-3: Pruebas con el Grasp_Construcción.

Verificando los resultados de la Tabla 4-3, podemos concluir que el mejor tiempo es de 51.67 seg, que es el tiempo que demora el usuario en utilizar la GUI y cumplir con el objetivo para la cual fue diseñada (en este caso abrir el archivo deseado); por lo tanto, la GUI propuesta quedará rediseñada de la forma que ilustra la Figura 4-2.

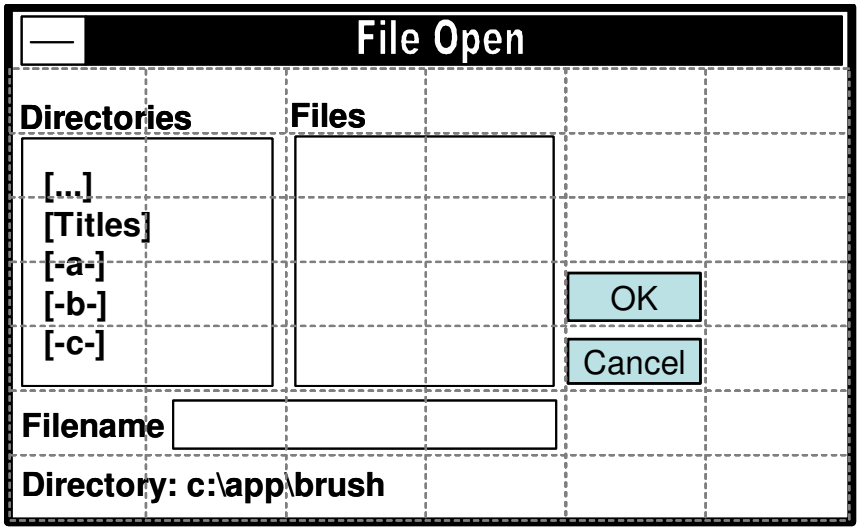


Figura 4-2: GUI rediseñada por el Grasp_Construcción.

Como se puede apreciar en la Figura 4-2, la GUI ha sido rediseñada por el algoritmo Grasp_Construcción, pero todavía no se tiene un diseño realmente eficiente, aunque el tiempo para utilizar la GUI se disminuyó, mejorando la distribución de los objetos sobre la GUI. Ahora bien la distribución de los objetos sobre la GUI mejoró porque ahora podemos realizar en menor tiempo todas las tareas de usuario y de esta manera cumplimos más rápidamente el objetivo para el cual esta GUI fue creada.

Para entenderlo mejor, pongamos un ejemplo sencillo, qué pasaría si tenemos que abrir 10 archivos para ver su contenido, con esta GUI rediseñada por el Grasp_Construcción lo realizaremos en menor tiempo que en la GUI propuesta por los diseñadores, ya que la mejor distribución de los objetos sobre la GUI nos permite ejecutar las tareas de usuario en menor tiempo.

4.7 Grasp mejoría en una distribución eficiente de objetos sobre una GUI

En cada ejecucion de Grasp_Construcción se genera una GUI configuración que en su proceso iterativo de construcción sólo considera los objetos ya ubicados. Esto es, consideremos por un instante que en la interación *r* del proceso de construcción, el objeto O_r se posiciona en una celda apropriada que sólo contempla los *r*-1 objetos ya ubicados en las *r*- 1 iteraciones anteriores, no contemplando los *m-r* objetos restantes.

En este sentido proponemos un procedimiento de mejoría que en cada iteración reubique un objeto que presente menor costo respecto a la solución que se desea mejorar. Este procedimiento se puede repetir mientras exista una mejor solución.

Grasp_Mejoria_Gui (Solución *S*, grilla *G*, tarea *J*)
1. $C := Costo(S)$ // Costo de la solución
2. Hacer
2.1. $Cs = C$ // Actualiza costo anterior
2.2. Para $O_i \in S, i = 1,...,m$ // Para cada objeto que define S
2.2.1. Retirar temporalmente O_i de S
2.2.2. Buscar la ubicación de menor costo para O_i
2.2.3. Si $Costo(O_i) < C$, entonces
2.2.3.1. $C := Costo(O_i)$
2.2.3.2. $k := i,\ p :=$ ubicación de menor costo para O_i
2.2.4. Fin-Si
2.3. Fin-Para
2.3. Si $C < C_s,$ entonces
2.3.1. $S := S - \{(O_k, p_k)\} \cup \{(O_k, p)\}$ //Actualizar *S* con nueva ubic. para O_k
3. Hasta (*Cs* > *C*)
4. Solución mejorada **S**.

En cada iteración del Grasp_Mejoría_Gui (segmentos de pasos 2.-3.), el algoritmo determina una mejor solución respecto a la configuración de la iteración anterior. En el segmento 2.2-2.3 se busca el objeto de menor costo en función de los *m*-1 objetos restantes.

4.8 Prueba y resultados del Grasp_Mejoría en una GUI

Para realizar una prueba a una GUI utilizando Grasp_Mejoría y observando realmente cuanto puede mejorar, tomemos como ejemplo la GUI rediseñada por el

Grasp_Construccción y realizamos un *testing* para ver realmente la mejoría en el rediseño de interfaces de usuario, aplicamos, realizamos la misma cantidad de corridas y nuestro parámetro se seguirá moviendo entre (0.1, 0.15, 0.2, 0.3, etc.), obteniendo como mejor parámetro el (α)=0.2, generamos la Tabla 4-4.

Objetos	Tareas de usuario	Celdas en grilla		Corridas	Costo	
		Ancho	Alto		Miliseg	Seg.
5	8	1cm.	1cm	100≤C≤500	50,942	50.94
5	8	1cm.	1cm.	500<C≤1000	50,756	50.76
5	8	1cm.	1cm.	1000<C≤2000	50,027	50.03

Tabla 4-4: Pruebas con el Grasp_Mejoria.

Verificando la tabla 4-4, podemos concluir que el mejor tiempo es de 50.03 segundos, tiempo que demora el usuario en completar las tareas y utilizar la GUI para lo que realmente fue diseñada, de esta manera la GUI rediseñada por el Grasp_Mejoría quedará rediseñada de la siguiente manera:

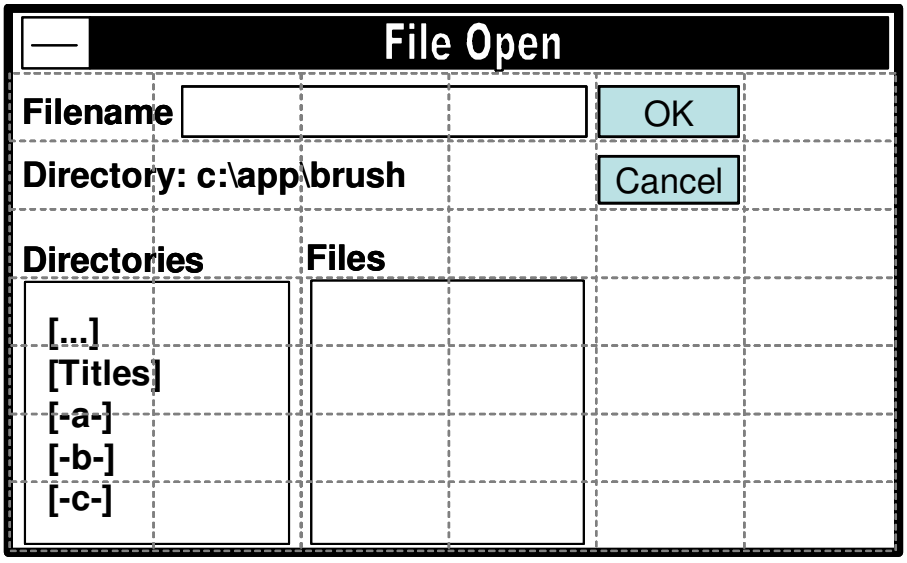


Figura 4-3: GUI rediseñada por el Grasp_Mejoria.

CAPÍTULO V: EXPERIMENTOS NUMÉRICOS

Habiendo desarrollado el Grasp_Construcción y el Grasp_Mejoría, pasamos a realizar los experimentos numéricos, teniendo en cuenta lo siguiente:

- El *hardware* en el cual realizamos los test numéricos.
- Las herramientas en la cual se desarrollo el *software* para realizar los experimentos numéricos.
- Las plataformas en la cual se ejecutó el *software*.
- Las instancias de pruebas propuestas.
- Análisis de los resultados obtenidos, resultados generados por el Grasp_Construcción y el Grasp_Mejoria.
- Teniendo en cuenta que este trabajo se enfoca en el rediseño de interfaces gráficas de usuario, analizaremos la calidad de los rediseños propuestos por los diversos algoritmos y de qué manera influyen en la interacción usuario-computadora.

5.1 *Hardware* y herramientas de desarrollo de *software* empleados

Para la generación de las pruebas numéricas empleamos el siguiente equipo de cómputo:

- Microprocesador Pentium IV Intel mobile.
- Velocidad 3.06 Ghz.
- Memoria ram 512mb.

Los sistemas operativos empleados fueron el Microsoft Windows 2000 service pack 4 y el Microsoft Windows XP con service pack 1, la herramienta de desarrollo es el Jbuilder versión X de Borland Corporation. Esta herramienta de desarrollo muy utilizada a nivel mundial está diseñada para utilizar el lenguaje orientado a objetos Java en su versión J2EE (Java 2 Enterprise Edition).

5.2 El *testing* en una GUI

Actualmente, no existen instancias de prueba múltiples para realizar pruebas numéricas en los diversos algoritmos propuestos (goloso, Grasp_Construcción, Grasp_Mejoria), pero para el trabajo propuesto podemos decir que cualquier GUI diseñada para la interacción usuario-computadora es una instancia de prueba.

Para realizar el test numérico consideremos una GUI con una determinada cantidad de objetos (4,5,6……..10), sobretodo que el problema del diseño o rediseño en una GUI no sólo se base en la cantidad de objetos que están sobre la superficie de la misma, sino también en la cantidad de celdas que formarán la grilla de la GUI, sabiendo que cada una de las celdas o un conjunto de celdas representan una posible posición para el objeto visual que está sobre la superficie de la GUI.

Para nuestro *testing* utilizaremos mayormente grillas de 6x7 y de 8x8 (no necesariamente), es decir dividiremos la GUI en una grilla de seis celdas de ancho por siete celdas a lo alto, en el testing de una GUI, no sólo se relacionan con los objetos y las celdas de una grilla, sino también con las tareas de usuario, ya que justamente una mejor redistribución de los objetos sobre la superficie de la GUI generará una GUI más eficiente y las tareas de usuario se podrán realizar en menor tiempo, la GUI tendrá un mejor diseño y cumplirá mejor su objetivo para la cual fue creada dentro de la interacción usuario-computadora.

Si queremos realizar *testing* de una GUI, teniendo una GUI de N objetos visuales sobre su superficie y una grilla de 6x7 que generaría 42 posibles posiciones, podemos generalizar el número de posibles diseños (NPD) de la fórmula F6 definida en la sección 3.3,

$$NPD = \frac{K!}{N!(K-N)}$$

Podemos realizar la operación con K=42 y N=5, de

$$NPD = \frac{42!}{5!(42-5)!} = 850{,}668 \qquad \text{Posibles diseños}$$

Como podemos observar, el número de posibles diseños (NPD) es bastante amplio. Para evitar una cantidad de diseños que nosotros podemos considerar innecesarios, debemos considerar una serie de "restricciones" que nos permitirán descartar diseños de GUI que serían inutilizables para la eficiente comunicación usuario-computadora.

Estas “restricciones” podemos considerarla observando las tareas de usuario, por ejemplo el botón Cancel siempre debe estar debajo del botón OK, aquí podemos tener una restricción para eliminar muchos diseños que consideren al botón Cancel arriba, derecha o izquierda del botón Ok; a parte de ello, la utilización de la metaheurística Grasp nos permitirá realizar búsquedas aleatorias de un conjunto determinado de diseños de GUI, lo que será bastante beneficioso en la obtención de una GUI con sus objetos visuales correctamente distribuidos.

5.3 El parámetro de relajación α

El algoritmo Grasp_Construcción maneja un parámetro de relajación α, que viene a ser una constante. En la notación del algoritmo Grasp_Construcción (capítulo IV), podemos ver claramente este parámetro.

El valor que tome este parámetro permitirá mejorar la solución generada por el algoritmo goloso, es por ello que se tiene que realizar pruebas con diversos valores para el parámetro de relajación, el parámetro puede tomar valores dentro del intervalo [0, 1].

5.4 Diseños propuestos de las GUI a evaluar

Toda GUI es construida por un diseñador, el cual se encarga de plasmar como debe ser la interacción entre el usuario y el computador a través de una interfaz gráfica de usuario. Para ello debe conocer las tareas que el usuario debe realizar con la GUI para cumplir con el objetivo para la cual esta fue diseñada. Es por ello que el diseñador muestra un primer diseño de lo que debe ser la GUI para realizar la interacción usuario-computadora u hombre-máquina. Las GUI propuestas por el diseñador se muestran en el Anexo A. Así, la propuesta de la GUI mostrada en la Figura A-1 y sus tareas de usuario mostrada en la tabla A-1 definen su caso de evaluación. También definen otro caso la GUI de la Figura A-2 con sus tareas de usuario mostrada en la tabla A-2.

5.5 Resultados numéricos

Los resultados numéricos que mostraremos es el resultado de la ejecución de nuestro algoritmo goloso, Grasp_Construcción y Grasp_Mejoría para los cuales realizaremos el *testing* a interfaces gráficas de usuario de 5, 6 y 7 objetos visuales sobre su superficie.

Realizamos el testing utilizando el parámetro de relajación α, con diversos valores todos en el intervalo [0,1], aparte de ello el testing lo hacemos sobre un determinado número de corridas del *software* construido para el testing de las GUI. Para verificar que tan eficiente es el Grasp_Mejoría con respecto al algoritmo goloso, usaremos la siguiente fórmula:

$$\%Eficiencia = 100 - \left(\frac{\text{Re}\,sultado\_Grasp\_Mejoria}{\text{Re}\,sultado\_A\text{lg}\_goloso} \right) * 100$$

Como podemos observar en la fórmula Fe el porcentaje de eficiencia nos indica que tan eficiente es el Grasp_Mejoría frente al algoritmo goloso, de esta manera podemos tener un indicativo de que tan eficientemente se están distribuyendo los objetos sobre la superficie de una GUI. Veamos un ejemplo para una GUI de 5 objetos visuales: hemos tenido como resultado para el algoritmo goloso 52.88 segundos y para la metaheurística Grasp 50.03, de esta manera aplicando la fórmula Fe lo siguiente:

$$\%Eficiencia = 100 - \left(\frac{50.03}{52.88} \right) * 100 = 5.43\%$$

Un porcentaje de eficiencia de 0.2% de la metaheurística Grasp sobre el algoritmo goloso. Como podemos observar el Grasp_Mejoría obtuvo un tiempo menor para completar las tareas de usuario que el tiempo obtenido por el algoritmo goloso.

De lo expuesto anteriormente, mostraremos dos tablas con resultados numéricos para 2 diferentes GUI, con determinadas cantidades de objetos visuales sobre ella. Para ver las GUI propuestas véase el Anexo A, donde también mostramos el costo inicial de las GUI propuestas.

Alg. Goloso	**(α)**	**Grasp_Construcción (Corridas)**				**MV**	**% E**	**Grasp_Mejoria (Corridas)**				**MV**	**% E**
		100	500	1000	2000			100	500	1000	2000		
52.88	0,1	51.87	51.81	51.77	51.70	51.70	2.23	51.70	51.60	51.30	51.23	51.23	3.12
	0,15	51.82	51.81	51.77	51.68	51.68	2.27	51.68	51.60	51.40	51.10	51.10	3.37
	0,2	51.80	51.78	51.76	51.67	51.67	2.29	51.01	50.98	50.94	50.03	50.03	4.88
	0,3	51.75	51.73	51.75	51.70	51.73	2.17	51.10	51.04	50.01	50.98	50.01	5.43

Tabla 5-1 Testing para una GUI con 5 objetos visuales en una grilla 6x7

Como podemos observar en la tabla 5-1, para el Grasp_Construcción el mejor valor (MV) es de 51.67 obteniendo un porcentaje de eficiencia de 2.29% con respecto al algoritmo

goloso. Tambien observamos que el Grasp_Mejoría tiene un mejor valor de 50.01 obteniendo un porcentaje de eficiencia del 5.43% con respecto al algoritmo goloso.

Goloso(Seg.)	Nro. Corridas	Alfa(α)	Grasp_Mejoria	% Eficiencia
25.51	100≤C≤1,000	0.2	25.30	0.82
25.51	1,000<C≤10,000	0.2	25.10	1.61
25.51	10,000<C≤20,000	0.2	24.98	2.08

Tabla 5-2 testing para una GUI con 6 objetos visuales en una grilla 8x8 y 9 tareas de usuario

En las tablas anteriores 5-1 y 5-2, podemos observar que el Grasp_Mejoria reduce el tiempo de ejecución de las tareas de usuario, aumentando la eficiencia en la comunicación usuario-computadora.

Para mostrar las GUI rediseñadas por el Grasp_Mejoría vease el Anexo C, donde la Figura C-1 es la GUI con sus objetos redistribuidos usando el Grasp_Mejoría a partir de la GUI de la Figura A-1, de forma similar es la Figura C-2 que es la GUI con sus objetos redistribuidos por el Grasp_Mejoría de la Figura A-2.

5.5.1. Eficiencia del Grasp_Construcción frente al diseño propuesto

Como podemos observar en la Figura 5-1, a medida que aumentamos el número de corridas del algoritmo Grasp_Construcción este mejora su eficiencia, reduciendo el tiempo de ejecución de las tareas de usuario, como sabemos el tiempo de reducción es en segundos que se hace evidente cuando se distribuyen los objetos más eficientemente sobre la superficie de la GUI, aumentando de sobremanera la comunicación usuario-computadora, aunque la reducción del tiempo en la ejecución de las tareas de usuario sea solo en algunos pocos segundos.

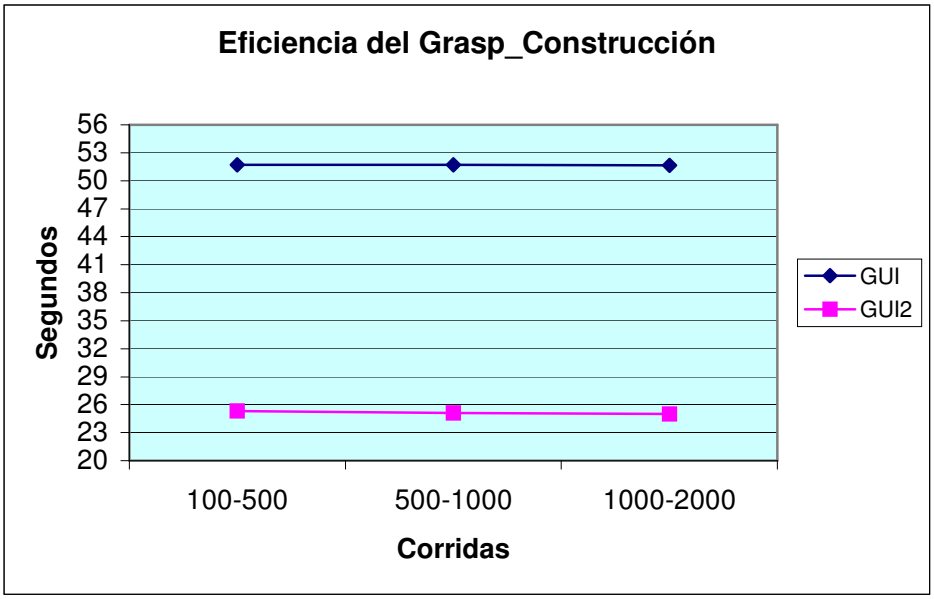


Figura 5-1: Muestra como el Grasp_Construcción reduce el tiempo de ejecución de las tareas de usuario al aumentar el número de corridas.

5.5.2. Eficiencia del Grasp_Mejoría frente al diseño propuesto

Como podemos observar en la Figura 5-2 al igual que el Grasp_Construcción el Grasp_Mejoria mejora a medida que el número de ejecuciones aumenta es por ello que la eficiencia de la metaheuristica Grasp se hace más efectiva.

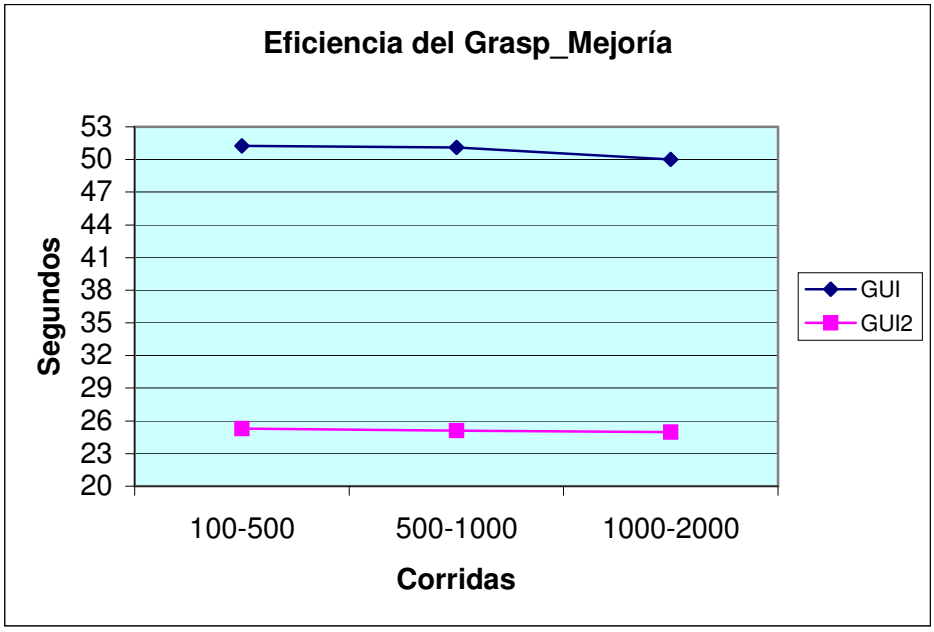


Figura 5-2: Muestra como el Grasp_Mejoría reduce el tiempo de ejecución de las tareas teniendo mayor eficiencia que el Grasp_Construcción

CAPÍTULO VI: CONCLUSIONES Y FUTUROS TRABAJOS

En la actualidad, el hombre continua constantemente rediseñando interfaces gráficas que mejoren la comunicación usuario-computadora, entiende cabalmente que para que un *software* tenga éxito debe poseer una GUI eficiente, que haga que el usuario se sienta cómodo trabajando en ella, imagínense por un momento una GUI de una entidad bancaria a las 12 del día, con una cantidad de clientes esperando depositar o retirar un dinero, si la GUI no es eficiente se tendría muchos problemas de congestionamiento de clientes, el usuario tendría gruesos problemas de tiempo para la atención y sencillamente se generaría un caos.

Cabe indicar que un principio básico del diseño es la *retroalimentación,* es la información o respuesta que da la computadora a alguna acción o mandato que se hizo. Esta es la parte esencial entre el usuario y la computadora. Muchas veces cuando se tiene una conversación con otra persona y esta no sonríe, no da un movimiento de cabeza o de alguna otra parte del cuerpo y solamente responde cuando es forzada a hacerlo, puede ser una pequeña muestra de que no esta entendiendo lo que le estamos diciendo. Bajo este principio quizás en el futuro tengamos que diseñar o rediseñar la GUI en base a respuestas que los objetos sobre la superficie de la GUI envíen al usuario.

6.1 Rediseño de GUI en pantallas táctiles

Las GUI han evolucionado. En la actualidad, en muchas instituciones, bancos, universidades y centros comerciales utilizan pantallas táctiles, en las cuales ya no usan un periférico de señalización como el mouse, sino que utilizan sus dedos para señalar los objetos de la GUI e interactuar con la computadora.

Esto quiere decir que tenemos que realizar *testing* o rediseñar interfaces gráficas, pero teniendo a los dedos de la mano como un señalador en lugar del Mouse. Es un trabajo futuro de rediseño muy interesante que nos va a permitir diseñar interfaces pensando que en los dedos de las manos como un periférico más del computador.

Las pantallas táctiles son unos dispositivos que aúnan las interfaces de comunicación de la máquina con el usuario y viceversa. De aspecto y construcción casi idénticas a las pantallas de ordenador o televisión normales (Figura 6-1), son capaces además de detectar el lugar en el que el usuario toca la pantalla, y en función de ello, realizar una u otra acción. Un ejemplo claro del uso de estos dispositivos son los puestos automáticos disponibles en muchas entidades bancarias. El usuario, ante la pantalla, ha de tocar esta en el lugar donde aparece escrita la operación que desea efectuar, a continuación cambia la presentación y se le ofrecen las opciones disponibles de esa operación, que el usuario irá eligiendo hasta que en la pantalla acaba por ofrecerse únicamente la información sobre el progreso de la operación.

Figura 6-1: Clasica pantalla táctil

6.2 Aspecto semiótico en la mejoría de una GUI

Como sabemos hoy en día, el diseño de una GUI es un problema bastante serio de tomar encuenta, ya que el *software* que tiene un código interno muy aceptable, un grado de almacenamiento mejor, no son aceptados por la sencilla razón de que la comunicación usuario-computadora es muy deficiente. Es por ello que las interfaces gráficas de usuario hoy en día juegan un papel primordial, ya que de ellas dependen en gran medida que las interacciones con la computadora se realicen cada vez más amigables, de tal manera que si existe una comunicación aceptable entre el usuario y la computadora por medio de la GUI, esta cumplirá mejor los propósitos para la cual fue diseñada.

6.3 Funcionalidad de objetos para una distribución eficiente

Podemos obtener una funcionalidad entre objetos visuales, en base a la relación existente entre ellos, como sabemos una GUI involucra tareas de usuario, estas tareas de usuario contiene sus respectivos objetos visuales y ellos poseen una frecuencia de recorrido, que

debe hacer el usuario con la ayuda del mouse o algún otro apuntador. En la Figura 6-2 podemos observar un diagrama de transición que nos va a ayudar a entender la relación que existe entre los objetos visuales.

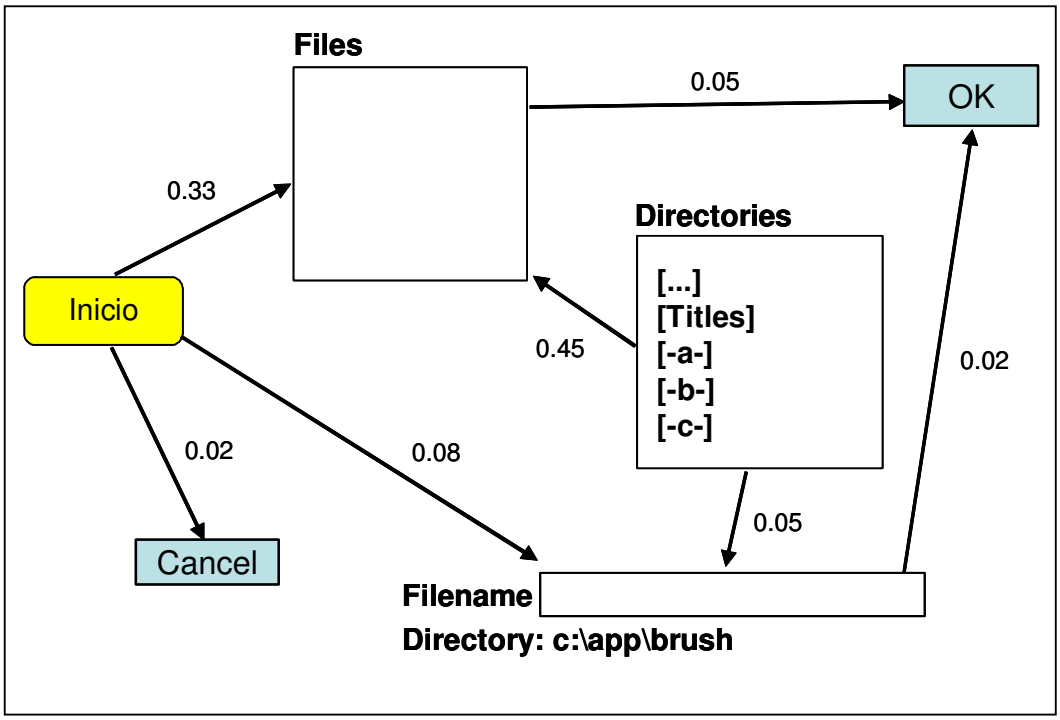


Figura 6-2: valores de transición entre objetos que nos indica la funcionalidad entre ellos

En la Figura 6-2, podemos observar un valor de transición (este valor fue calculado en el Capítulo III, en base a sus respectivas frecuencias), este valor de transición nos va a otorgar la mayor o menor funcionalidad que existe entre ellos. Esto quiere decir que cuando hagamos una redistribución de objetos visuales mediante este criterio, seleccionamos el objeto visual que queremos redistribuir y buscamos el objeto visual con el que tiene mayor funcionalidad, en este caso mayor valor de transición. Colocamos el objeto en la celda disponible de la GUI y realizamos el cálculo nuevamente aplicando la ley de Fitts para calcular el costo en base al tiempo de recorrido para completar la tarea que involucra a estos objetos. Repetimos el cálculo del objeto visual seleccionado que queremos redistribuir en todas las celdas disponibles de la GUI y elegimos la ubicación del objeto visual en la GUI con menor costo.

De acuerdo a la Figura 6-3, el botón OK posee una relación de funcionalidad con los objetos File y *Filename*, pero posee una mayor funcionalidad con el objeto File, por lo tanto la distancia entre ellos debe ser mínima.

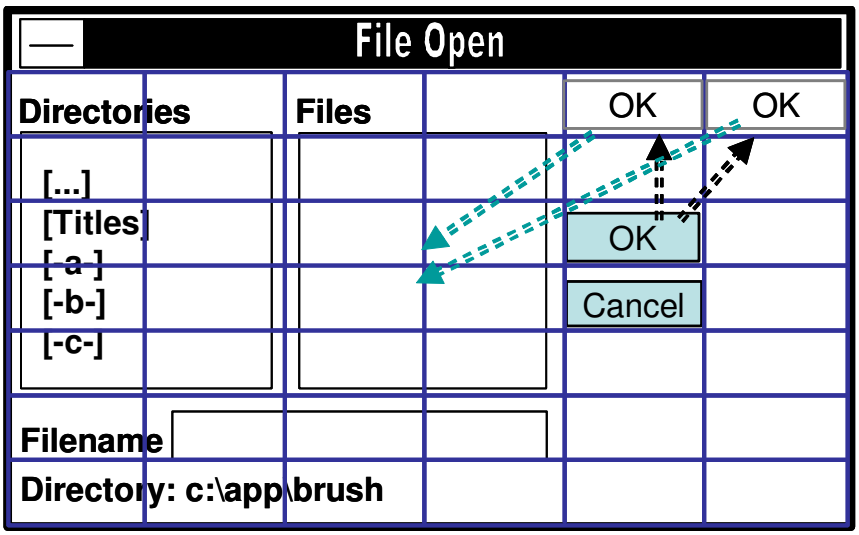


Figura 6-3: Redistribución del objeto visual OK.

En la Figura 6-3 las líneas negras nos indican la dirección hacia las nuevas posibles posiciones que puede tener el botón OK, ya que dichas celdas se encuentran disponibles para redistribuir este objeto visual. Las líneas mas claras nos indican el costo en términos de tiempo de recorrido que debemos calcular con el objeto que posee mayor funcionalidad en este caso el objeto visual *File*.

6.4 Semiótica visual en la distribución de objetos

Como hemos explicado anteriormente (Capítulo II), semiótica visual es el impacto que causa la simbología, forma y distribución de los objetos visuales en el usuario. Esto quiere decir que la ubicación de los objetos visuales sobre la superficie de la GUI originan un impacto en la comunicación usuario-computadora, para explicarlo detenidamente seleccionamos un objeto visual, para nuestro ejemplo elegiremos nuevamente el botón OK.

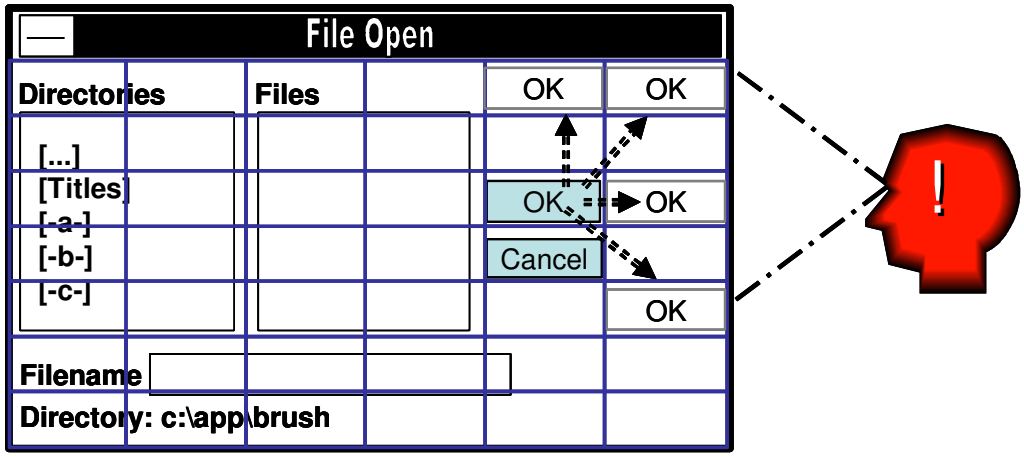


Figura 6-4: Impacto de comunicación al redistribuir el botón OK.

En la Figura 6-4, podemos observar las diferentes posiciones que puede ser ubicado el botón OK, dentro de la superficie de la GUI, el usuario u operario tendrá un impacto de comunicación determinado en cada ubicación disponible para el botón OK sobre la superficie de la GUI. Esto quiere decir que si ubicamos el botón OK en la celda con coordenadas (5,5) tendrá un impacto de comunicación diferente a colocar el botón OK en la celda con coordenadas (7,5).

Para mejorar el diseño de la GUI utilizando semiótica visual en la redistribución de objetos visuales sobre la superficie de la GUI, podemos asignar puntajes o valores específicos a las celdas de la GUI con respecto al objeto visual. Estos valores o puntajes que asignamos a las celdas varían para cada uno de los objetos visuales que conforman la GUI, ya que dependiendo del objeto visual y de la función que realizan hay celdas con mayor o menor impacto de comunicación usuario-computadora.

Un ejemplo sencillo seria por ejemplo un menú de opciones, si tenemos una GUI en la cual debemos colocar un menú de opciones, las celdas superiores tendrán un mayor valor visual que las celdas inferiores, porque semioticamente el impacto mayor de un menú de opciones esta en ubicarlo en la parte superior de la superficie de la GUI.

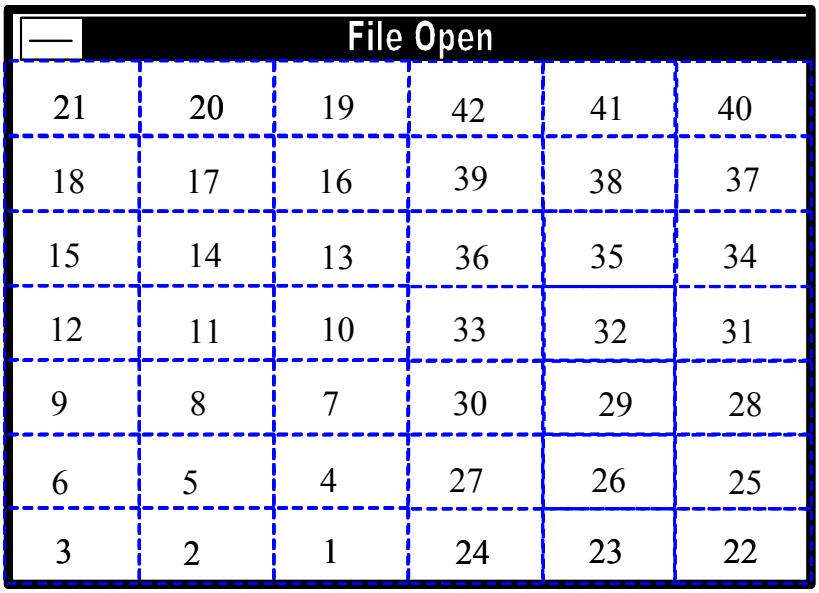


Figura 6-5: Celdas con valores o puntajes semióticos para el botón OK.

Como podemos observar en la Figura 6-5 hemos asignado puntajes o valores semióticos para el botón OK. De la misma manera tenemos que asignar valores semióticos para cada objeto visual sobre la superficie de la GUI.

Las celdas con mayores valores semióticos son las que causan mayor impacto de comunicación entre el usuario y el computador al ubicar en esa posición o celda al objeto visual correspondiente, en este caso el botón OK.

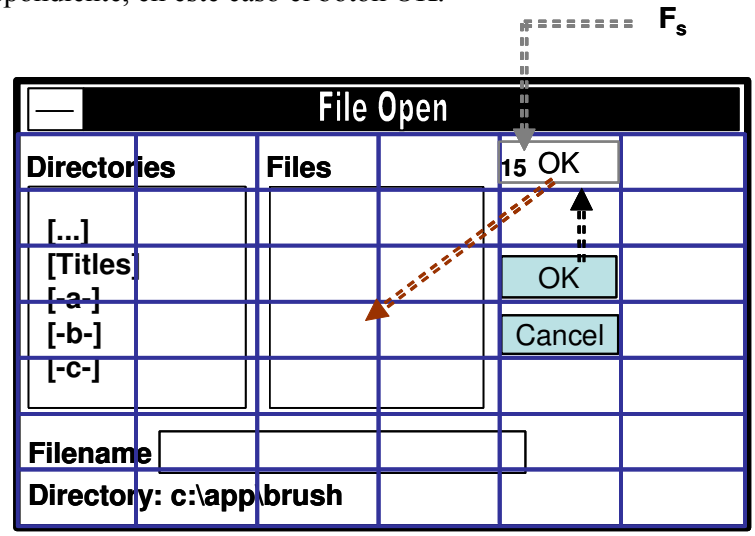


Figura 6-6: Funciones para una GUI mejorada

Como podemos observar en la Figura 6-6, hemos creado una función semiótica que contenga los puntajes asignados para un determinado objeto sobre la GUI, en este caso tenemos puntajes asignados al botón OK. Con este puntaje asignado, podríamos crear una función semiótica que pueda ponderizarse con la distancia calculada por la Ley de Fitts. De de esta manera, podríamos introducir semiótica en el rediseño de las interfaces de usuario.

6.5 Utilización de Semiótica en el rediseño visual

En nuestro trabajo, hemos hablado de semiótica visual como un trabajo futuro para rediseñar las GUI, desde el punto de vista de un rediseño visual. De esta manera, un punto importante en el rediseño visual es la claridad visual.

Se puede decir que se tiene claridad visual si el significado de una imagen es realmente clara para la persona. Un punto importante para llegar a tener claridad visual es organizar la información de una manera lógica. Existen algunas reglas para organizar la información visualmente, estas reglas fueron hechas por la *Gestalt psychologist,* Wertheimer [48]. Ellos describen como una persona organiza los estímulos visuales individualmente en una forma global (de ahí el término Gestalt, literalmente "figura" o "forma" la cual denota un énfasis en la totalidad, en lugar de las partes que la constituyen).

Bajo este punto, sería un trabajo muy interesante rediseñar las GUI, en base a la posición donde más claro se ubica para el usuario. De esta manera aparte de diseñar o rediseñar GUI en base a tareas de usuario, utilizando Semiótica estaríamos diseñando interfaces en base a las preferencias de usuario. Un punto importante a tener en cuenta en el diseño de las GUI.

6.6 Conclusiones finales

El diseñar GUI con objetos correctamente ordenados u organizados, elevan considerablemente la comunicación usuario-computadora, de esta manera el *software* creado se hace mucho más eficiente.

Tenemos que entender que las interfaces gráficas de usuario es el inicio de la interacción entre el usuario y la computadora y que si se posee una GUI deficiente esta comunicación resultará totalmente errónea.

El mundo se basa en figuras y colores, el aspecto semiótico es muy importante y las metáforas que apliquemos para nuestros objetos dentro de una GUI, impactaran de manera significativa en el cerebro del ser humano y este de inmediato tendrá una idea para que sirve ese objeto dentro de la interfaz o acaso no nos damos cuenta cuando vemos un botón que contiene el gráfico de un disquete, inmediatamente sabemos que ese botón es para grabar información.

Las GUI en el futuro formaran parte esencial de nuestras vidas, las computadoras realizaran casi todas nuestras tareas cotidianas y tendremos GUI por todos lados, así que preparémonos para un mundo formado por interfaces graficas de usuario.

REFERENCIAS BIBLIOGRÁFICAS

[1] Molich, R. y Nielsen, J. **"Heuristic evaluation of user interfaces"**, Proceedings of the ACM CHI´90. Conference, 1990. pp. 249-256.

[2] Sears, A. **"Layout Appropriateness:A Metric for Evaluating User Interface Widget Layout"**, IEEE Trans. Software Engineering, vol. 19, n.7, Julio 1993, pp 707-719.

[3] Pressman, Roger S. **"Ingeniería de Software**, **un enfoque práctico"**. McGraw-Hill, 2001. pp 299 – 336.

[4] Gómez, L.S.M. **"Diseños de interfaz de usuario"**. Vol 1. 2001, pp 15 – 40.

[5] Kantowitz, B. and Sorkin, R. *Human Factors***: "Understanding poeple-system relationships"**. New York: John Wiley and Sons, 1983. pp. 226-227.

[6] Gupta, R. and Sharit, J. **"Human-Computer Interaction in Facilities Layout".** In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. Amsterdam: Elsevier Science Publishers, 1988, pp. 729-736.

[7] Card, S.; Moran, T. and Newell, A. **"The Psychology of Human-Computer "Interaction, Hillsdale**, NJ: Erlbaum, 1983.

[8] Kieras, D. **"Towards a Practical GOMS Model Methodology for User Interface Design"** In M Helander (Ed.), *Handbook of Human-Computer Interaction* (pp. 135-157). Amsterdam: Elsevier Science Publishers, 1988.

[9] Tullis, T. **"A system for evaluating screen formats: Research and application"** In: R. Hartson& D. Hix (Eds.). *Advances in Human-Computer Interaction.* Vol. 2, 1988, pp. 214-286.

[10] Perlman, G. **"An Axiomatic Model of Information Presentation"**. *Proceedings of the 31st Annual Meeting of the Human Factors Society.* 1987, pp. 1129-1233.

[11] Lohse, J. **"A Cognitive Model for the Perception and Understanding of Graphics"** *CHI'91 Proceedings*. 1991. pp. 137-144.

[12] Shneiderman, B. **"Designing the User Interface: Strategies for Effective Human-Computer Interaction (4ta. Edition)"**, Reading, MA: Addison-Wesley, 2005, pp. 75-80.

[13] Sears, A. **"Layout Appropriateness: Guiding interface design and evaluation with simple task descriptions"**, Ph D Dissertation, Computer Science Department, University of Maryland. College Park, MD, 1999.

[14] MacKenzie, I. S. and Buxton, W. **"Extending Fitts' Law to Two-Dimensional Tasks",** *CHI'92 Proceedings*, May 1992, pp. 219-226.

[15] De Baar, D.; Foley, J. and Mullet, K. **"Coupling Application Design and User Interface Design",** *CHI '92 Proceedings*, May 1992, pp. 259-266.

[16] Kim, W. and Foley, J. **"DON: User Interface Presentation Design Assistant",** *UIST '90Proceedings*. 1990, pp. 10-20.

[17] Mackinlay, J. **"Applying a Theory of Graphical Presentation to the Graphic Design of UserInterfaces",** *UIST '88 Proceedings*. 1988, pp. 179-189.

[18] Szekely, P. **"Template-Based Mapping of Application Data to Interactive Displays",** *UIST'90 Proceedings*, 1990, pp. 1-9.

[19] Phillips, M. D.; Bashinski, H. S.; Ammerman, H. L. and Fligg, C. M. Jr., **"A Task Analytic Approach to Dialogue Design",** In M Helander (Ed.), *Handbook of Human-Computer Interaction*(pp. 835-857). Amsterdam: Elsevier Science Publishers, 1988.

[20] Casner, S. and Larkin, J. H. **"Cognitive Efficiency Considerations for Good Graphic Design",** *Cognitive Science Society Proceedings*, August 1989.

[21] Gómez, A. **Algoritmos voraces, webepcc.unex.es/agomez/EDA0001Tema11.pdf** Universidad de Extremadura España, Departamento de Informática - Área de Lenguajes de Programación y Sistemas de Información (2000 - 2001).

[22] Feo, T. y Resende, M**. "Greedy Randomized Adaptive Search Procedure Journal of Global Optimization",** número 6, 1995, pp. 109-133.

[23] Fenton, N. **"Software Metrics"**. Chapman & Hall, 1,991

[24] Andersen, P. B. **"A Theory of Computer Semiotics".** Cambridge, MA. Cambridge University Press. 1990.

[25] Nadin, M. Interface design and evaluation, in R. Hartson, D. Hix (Eds.) "**Advances in Human-Computer Interaction"**, Norwood, NJ: Ablex Publishing Corp., vol. 2, 1988.

[26] Nake, F. "**Languages of Design 2",** Human-computer interaction: signs and signals interfacing, 2002, pp. 193-205.

[27] De Souza, C. S. "**Knowledge Based Systems",** Editorial. Amsterdam: Vol.14, N.8, 2001. p.415 - 418.

[28] Reeves, C. R. (ed.), "**Modern Heuristic Techniques for Combinatorial Problems"**, Blackwell Scientific Pub., Oxford, 1993.

[29] Eilon, S. **"More Against Optimizacion"**, Omega, Vol. 5, 1977.

[30] Zanakis, S. H.; Evans, J. R. **"Heuristic optimizacion**: Why, When, and How to Use it", interfaces. Vol. 11, N.5, 1981.

[31] Feo, T. A.; Resende, M. G. **"A probabilistic heuristic for a computationally difficult set covering problem"**, Operations Research Letters, 8:67-71, 1989.

[32] Booth, P. **"An introduction to Human-Computer Interaction"**. Hilldale, NJ:Erlbaum, 1989.

[33] Foley, J.; Wallage, V.; Chan, P. **"The human factors of computer graphics interaction techniques"**. IEE Computer Graphic and Applications, 4:11**,** noviembre 1984, pp. 13-48.

[34] Edwards, A. Speech synthesis: **"technology for disabled people"**. London: Chapman, 1991.

[35] Martin, Suzanne **"Effective visual Communication for grafical user interfaces",** 1998, http://web.cs.wpi.edu/~matt/courses/cs563/talks/smartin/int_design.html. accesado en enero 2007.

[36] De Souza, Clarisse "**Semiotic engineering principles for evaluating end-user programming environments",** Interacting with Computers 13, 2001, pp. 467–495.

[37] Marcos, M. C. "**Presente, pasado y futuro de la investigación en interacción hombre-ordenador aplicada a la documentación**", 2002.

[38] Welford, A. T. **Fundamentals of skill**, London:Methuen, 1968.

[39] Millan, José "**Adaptative Brain Interfaces"**, Communications of the ACM, v. 46, n. 3, 2003, pp. 74-80.

[40] Shumin, Zhai **"What´s in the eyes for attentive inputs"**, Communications of the ACM, v. 46, n. 3, 2003, pp. 34-39.

[41] Maglio, Paul y Campbell, Christopher, "**Attentive agents"**, Communications of the ACM, v. 46, n. 3, 2003, pp. 47-51.

[42] Savage Carmona, Jesus **"A Hybrid Systems with Symbolic AI and Statical Methods for Speech Recognition"**, University of Washington, 1995.

[43] Oropeza Rodríguez, José Luis **"Reconocimiento de Comandos Verbales usando HMM"**, Centro de Investigación en Computación, Instituto Politécnico Nacional Mexico D.F. 2000.

[44] Rabiner, Lawrence and Biing-Hwang Juang, **"Fundamentals of Speech Recognition"**, Prentice Hall, 1993.

[45] Wilken, Peter and Honekamp, Dirk **"La sílaba como elemento de reconocimiento en un ASR"**, Abacus 1991.

[46] Rabiner, L. R. **"A tutorial on hidden Markov models and selected applications in speech recognition"**, Proc. IEEE, V. 77, N. 2,1989, pp. 257-286.

[47] Sears, Andrew **"AIDE: A tool to assist in the design and evaluation of user interfaces"**, Interactive Systems Research Center, Baltimore, 1993.

[48] LUCHINS, A. S. and LUCHINS, E. H. **"Introduction to the Origins of Wertheimer's Gestalt Psychology"**, Gestalt Theory, 4(3-4), 1982, pp. 145-171.

[49] ESSi PIE 24306: Tilo Linz, Mattias Daigl imbus GMBH, D-91096 Möhrendorf Germany, http://www.imbus.de/engl/forschung/pie24306/index.shtml, accesado em enero 2007.

ANEXO A

GUI propuestas por los diseñadores

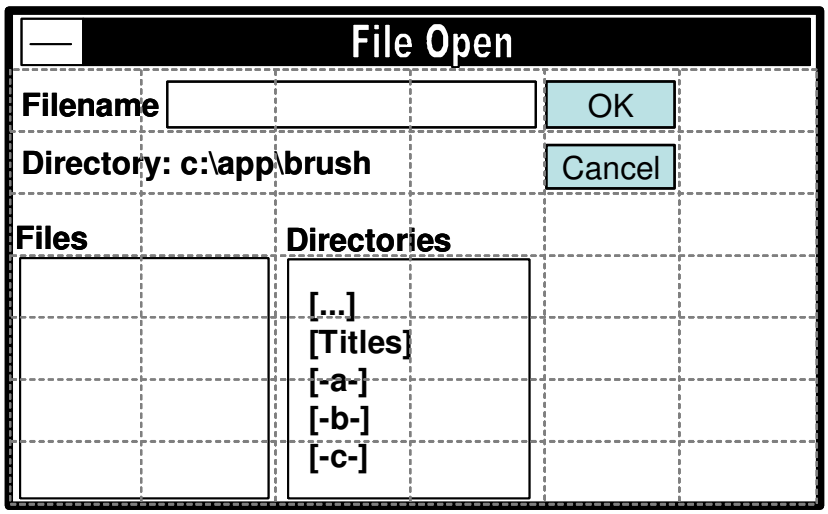


Figura A-1: Primera GUI propuesta por los diseñadores, teniendo un costo inicial de 68.20 segundos en una grilla de 6x7 para completar sus tareas de usuario.

Tarea	Frecuencia	Objetos Usados
1	5	Directory - Filename
2	25	File
3	20	Directory
4	2	Filename- Ok
5	5	Filename
6	5	File - Ok
7	45	Directory – File
8	1	Ok - Cancel
T. sesión	108	

Tabla A-1: Tareas de usuario para la GUI mostrada en la Figura A-1.

La Figura A-1 muestra una GUI propuesta por los diseñadores con 5 objetos visuales, teniendo un costo inicial de 51.14 segundos que el usuario utiliza al completar todas las tareas descritas en la Tabla A-1 y cumplir el objetivo para la cual la GUI fue construida.

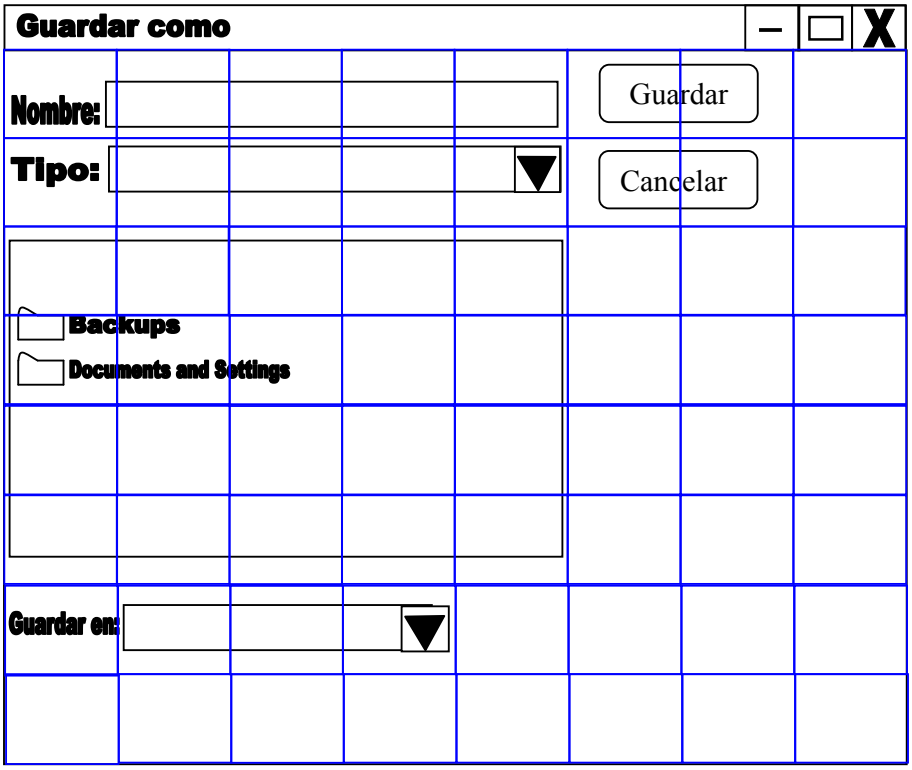


Figura A-2: Segunda GUI propuesta por los diseñadores(aplicación universal para el guardado de un archivo), teniendo un costo inicial de 25.96 segundos para completar las tareas de usuario.

Tarea	Frecuencia	Objetos Usados
1	5	Comboguardar
2	10	Comboguardar - File
3	20	File
4	4	Filename- Combotipo
5	2	File - Comboguardar
6	10	File - Filename
7	3	Filename
8	1	Boton guardar – Boton cancelar
9	1	File - Combotipo
T. sesión	56	

Tabla A-2: Tareas de usuario para la GUI mostrada en la Figura A-2.

La Figura A-2 nos muestra una GUI con un grado más de complejidad (6 objetos visuales y una grilla de 8x8) propuesta por los diseñadores. En esta GUI, se requiere 25.96 segundos para que el usuario complete las tareas descritas en la Tabla A-2, de igual manera que en la GUI anterior cumplir el objetivo para la cual fue construida.

Anexo B

GUI con los objetos redistribuidos por el Grasp_Construcción

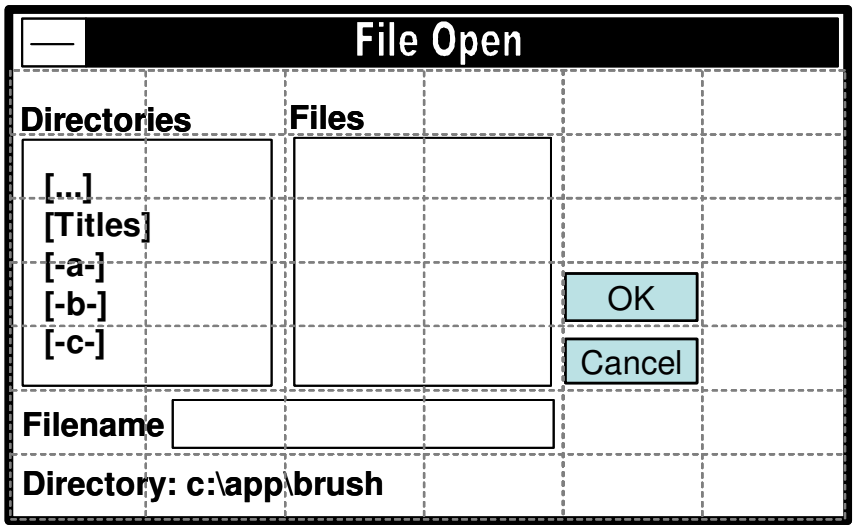


Figura B-1: GUI de 5 objetos rediseñada por el Grasp_Construcción con un costo calculado mínimo de 51.67 segundos para ejecutar sus tareas.

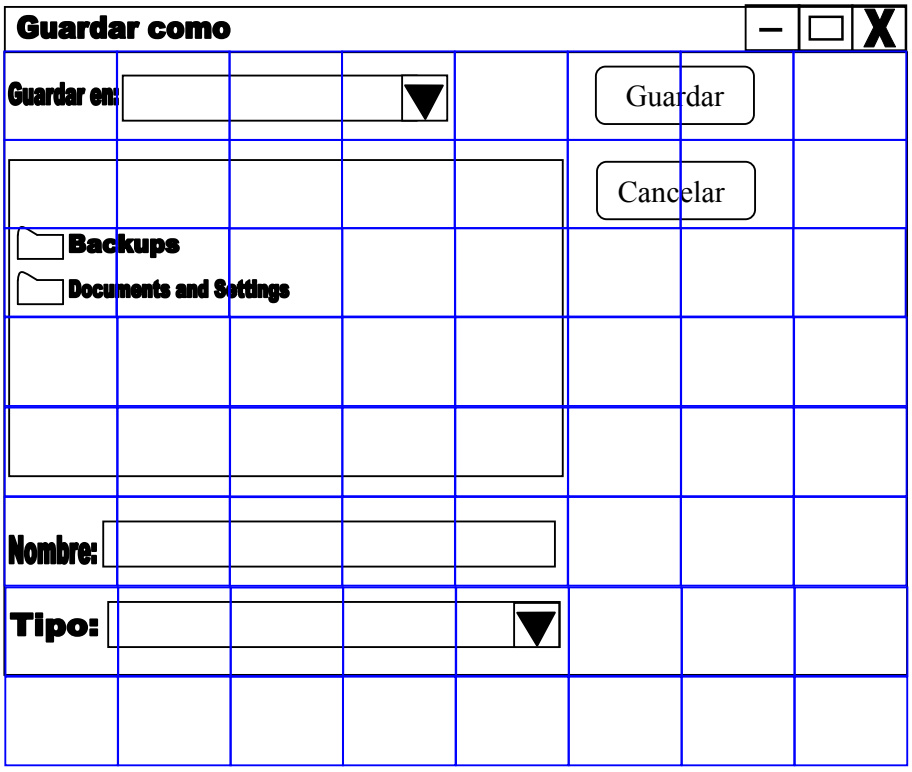


Figura B-2: GUI de 6 objetos rediseñados por el Grasp_Construcción, con un costo calculado mínimo de 24.98 segundos para ejecutar sus tareas.

Anexo C

GUI con los objetos redistribuidos por el Grasp_mejoria

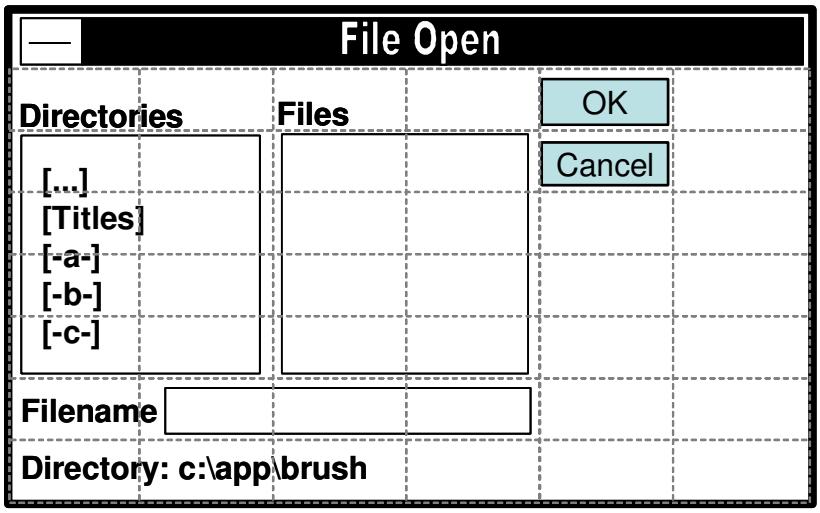


Figura C-1: GUI rediseñada por el Grasp_Mejoria con un valor de redistribución de 50.03

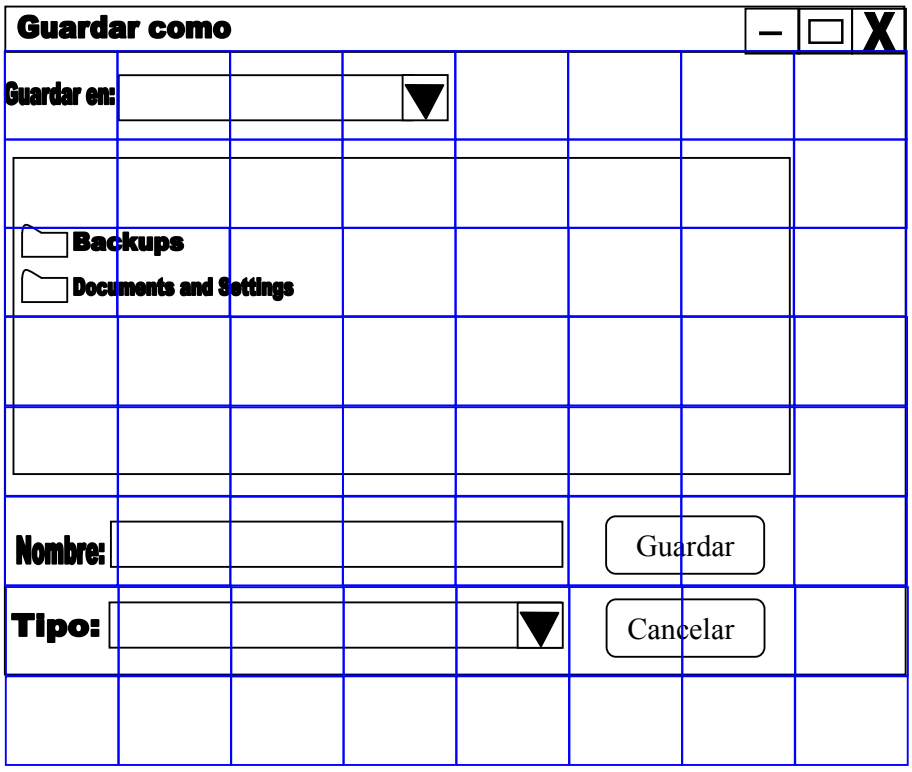


Figura B-2: GUI de 6 objetos rediseñados por el Grasp_Construcción, con un costo calculado mínimo de 24.98 segundos para ejecutar sus tareas.