Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Interfaz Gráfica de Usuario aplicada a un Sistema Experto.

Autor: Jorge Lázaro González Martínez.

Tutores: Ing. Ailet Abreu López.

Dr. José Rafael Abreu García.

Consultante: MSc. Luis Alberto Quintero Domínguez

Santa Clara

2016

"Año 58 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Interfaz Gráfica de Usuario aplicada a un Sistema Experto.

Autor: Jorge Lázaro González Matinés

jgmartinez@uclv.cu

Tutores: Ing. Ailet Abreu López.

aileta@uclv.cu

Dr. José Rafael Abreu García.

abreu@uclv.edu.cu

Consultante: MSc. Luis Alberto Quintero Domínguez

lquitero@uclv.cu

Santa Clara

2016

"Año 58 de la Revolución"

PENSAMIENTO

¿Qué poder es éste? No lo sé. Sólo sé que existe.

Alexander Graham Bell

DEDICATORIA

A las personas más importantes de mi vida: mis padres Lázara y Jorge Luis, a mi abuela Nené y mi tía Livita por su dedicación, apoyo y todos los sacrificios que han hecho por mí todos estos años sin los cuales no hubiera llegado a donde estoy.

AGRADECIMIENTOS

A mis padres por quererme incondicionalmente, por apoyarme en todas mis decisiones y seguirme en todos los pasos que he dado hasta convertirme en lo que soy hoy.

A mi abuela y a mi tía por su amor, su cariño y todo el sacrificio que han hecho por mí en todos estos años.

En general a toda mi familia que me ha apoyado incondicionalmente durante todo este tiempo.

A Betty por estar ahí cuando más lo he necesitado, por todo el apoyo y cariño que me ha brindado desde que nos "conocimos".

A mi tutora Ailet, por dedicarme tanto tiempo y confiar en mí.

A todos mis amigos de la universidad, los cuales hemos estado unidos en momentos de diversión y en momentos de tensión.

En general, a todas las personas que han contribuido a la realización de esta tesis.

RESUMEN

Las interfaces gráficas de usuario (GUI) proporcionan un método de interacción entre los usuarios y los ordenadores. Su objetivo es facilitar la forma en que los usuarios proporcionan instrucciones al ordenador, es decir actúa como un mediador entre el ordenador y el usuario. Durante los últimos años se han realizado diversos avances en el desarrollo de las GUIs, llegándose a convertir en herramientas sumamente importantes para facilitar el uso de las más variadas aplicaciones dentro del ámbito ingenieril. El presente trabajo se basa en la necesidad de crear una GUI que facilite el proceso de diagnóstico de fallas en bombas de alimentación de agua. A partir de dicha necesidad, se pretende diseñar e implementar una GUI para el Sistema Experto *"CTExperto"*, desarrollado en investigaciones anteriores. En tal sentido, se realizó una exhaustiva búsqueda bibliográfica de donde se extrajeron los principios básicos y la herramienta adecuada para el diseño e implementación de la GUI del sistema.

La GUI diseñada posibilita, de forma sencilla y ergonómica, realizar el proceso de diagnóstico de las bombas; minimizando en gran medida el tiempo de aprendizaje de la aplicación. De aquí deriva, en gran medida, la relevancia del trabajo realizado.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Las interfaces de usuario constituyen una de las partes más importantes y determinantes del desarrollo de cualquier producto software. La interfaz de usuario es la cara de la aplicación ante los usuarios, que interactúan con ella; dirige al usuario en la recogida de datos para que éstos sean procesados y, al mismo tiempo, permite la visualización del procesamiento de estos datos, convirtiéndolos en información útil para la toma de decisiones (Cruz Ocampo, 2012).

Las interfaces gráficas de usuario conocida también como GUI (de inglés graphical user interface) representan, en la actualidad, el punto de interconexión que permite el constante flujo de información entre el usuario común y el medio de cómputo. La necesidad de confeccionar una herramienta que posibilitara el empleo de la computadora a cualquier usuario, hizo que estas surgieran como resultado de la evolución de las complejas interfaces de líneas de comando empleadas para el manejo de los primeros sistemas operativos. Un conjunto de imágenes y objetos gráficos, tales como: botones, menús, sliders, campos de texto, gráficos y otros, hacen posible la representación de la información y las acciones disponibles en la interfaz, proporcionando un entorno gráfico ergonómico que facilita la interactividad con la máquina.

En la actualidad existe un gran número de aplicaciones y herramientas de software que hacen uso de GUIs. Ejemplo de ello, son los denominados Sistemas Expertos (SE) que se crean en diferentes campos del conocimiento como en plantas de energía, grandes industrias, cohetes, control del tráfico aéreo, búsqueda de yacimientos petrolíferos y hasta hospitales. Unas de las ventajas de estos sistemas es que no es necesario la presencia de una persona especialista en un área particular del conocimiento humano, debido a que el programa permite resolver problemas específicos de esa área de manera inteligente y

satisfactoria. Donde su tarea principal puede ser tratar de aconsejar al usuario. Con el objetivo de hacer más fácil y ergonómica dicha tarea es indispensable contar con una GUI adecuada.

El diseño de interfaces de usuario de calidad se preocupa por el logro de una interacción adecuada entre el usuario y el ordenador, asegurando de esta manera que el usuario pueda realizar las tareas de una manera efectiva, eficiente y satisfactoria (Cruz Ocampo, 2012).

Varios son los trabajos que, en los últimos años, se han realizado con el fin de incrementar la productividad en el desarrollo de interfaces de usuario que faciliten el trabajo con las más variadas aplicaciones dentro del ámbito ingenieril.

Tal es el caso del artículo "***Diseño de interfaces de usuario para aplicaciones colaborativas a partir de modelos independientes de la computación***", de María Luisa Rodríguez, José Luís Garrido, Manuel Noguera, María Visitación Hurtado, José Ramón Polo. Donde se propone una aproximación al diseño de interfaces de usuario para aplicaciones colaborativas a partir de modelos independientes de la computación (Luisa et al., 2010).

El desarrollo de interfaces de usuario basado en modelos es la construcción de interfaces mediante descripciones de alto nivel de los distintos aspectos de la interfaz. En este sentido se han creado un gran número de lenguajes de modelado. En el trabajo de Nuria Hurtado Rodríguez, J. Mariano González Romano y Jesús Torres Valderrama, "***Revisión de Lenguajes Declarativos para la Descripción de Interfaces de Usuario Independientes del Dispositivo***", se presenta una revisión de los principales lenguajes basados en XML para la descripción de interfaces de usuario independientes del dispositivo, realizando una comparativa entre ellos y destacando su relación con el proceso de modelado de interfaces de usuario basado en modelos declarativos y sus fases o niveles de abstracción (Rodríguez et al., 2004).

Con el avance en el campo de las tecnologías el desarrollo de un producto software concreto es una labor compleja debido a que deben dar respuesta a nuevas necesidades, como puede ser la movilidad, la tolerancia a fallos, la sincronización, la detección de bloqueos, la adaptabilidad, el cifrado o el control de accesos, entre otros. Francisco Montero, Víctor López Jaquero, María Lozano y Pascual González en el artículo "***De***

Platón al Desarrollo de Interfaces de Usuario" plantean la necesidad de contemplar, a la hora de desarrollar GUIs, una arquitectura metanivel en el que actividades ortogonales y comunes asignadas a la interfaz pueden establecerse a nivel meta y mediante un proceso de reflexión, y a partir de la contemplación de casos, cristalizar en componentes concretos de interfaz para un dispositivo concreto. Para la implementación de dicha propuesta se abusa del concepto de patrón en todas sus versiones; arquitectónico, de diseño, de leguaje y, por supuesto, de interacción con el fin último de garantizar la "*usabilidad*" de las interfaces finalmente generadas (Montero et al., 2005).

Las técnicas de representación de la información, como el análisis de coautoría y el escalamiento multidimensional (MDS), permiten obtener una imagen clara del funcionamiento de una realidad particular mediante el análisis de forma gráfica de estas relaciones. El artículo "***La coautoría en la revista ACIMED en el período 2005–2006: un análisis mediante interfaces gráficas"*** de Ramón Alberto Manso Rodríguez publicado en Cuba expone los resultados de la aplicación de estas técnicas a la publicación seriada "*ACIMED*" en el período comprendido entre los años 2005 y 2006, con el objetivo de conocer el comportamiento de las relaciones entre los autores y sus instituciones (Manso Rodríguez, 2007).

En la Universidad Central "Marta Abreu" de las Villas se ha tratado el tema en cuestión siendo ejemplo de ello el trabajo de diploma titulado ***"Sistema inteligente para el diagnóstico de defectos en calderas de vapor acuotubulares",*** en esta se diseña e implementa un sistema experto para el diagnóstico de fallas en calderas de vapor acuotubulares (Espinosa, 2013).

Para que el diagnóstico de fallas mediante sistemas expertos tenga efectividad es necesario que el diagnóstico realizado se desarrolle a través de una interfaz capaz de facilitar la comunicación de manera rápida y eficaz entre el sistema y el usuario.

En la búsqueda realizada no se ha encontrado referencias sobre alguna GUI adecuada destinada a informar al usuario acerca de las fallas diagnosticadas en las bombas de alimentación de agua a un generador de vapor.

Por lo que se plantea la siguiente **situación del problema**: No existe una GUI para el SE encargado del diagnóstico de fallos mecánicos en bombas de alimentar agua a calderas generadoras de vapor, capaz de mejorar la comunicación de los usuarios con el sistema.

A partir de la situación planteada, surgen entonces las siguientes **interrogantes científicas**:

¿Cómo desarrollar una GUI para el SE encargado del diagnóstico de posibles causas de síntomas de fallos mecánicos en las bombas de alimentar agua?

¿Cómo determinar la herramienta apropiada para desarrollar la GUI atendiendo a las características del software UCShell en el cual esta implementado el SE?

¿Qué particularidades presenta este software para implementar la GUI y a su vez comunicarla con el sistema experto?

¿Cómo programar y comunicar, la interfaz gráfica de usuario con el SE creado en UCShell?

Con el fin de dar solución a las interrogantes científicas anteriores se proponen los siguientes objetivos de este trabajo:

Objetivo General:

- Programar una Interfaz Gráfica de Usuario (GUI) para el Sistema Experto, encargado del diagnóstico de fallas mecánicas en las bombas de alimentar agua a un generador de vapor.

Objetivos específicos:

- Analizar la bibliografía especializada sobre las herramientas que permiten diseñar e implementar GUIs.
- Estudiar las características del software elegido y la factibilidad de su utilización para la programación de la GUI del Sistema Experto.
- Diseñar, la interfaz gráfica de usuario para el Sistema Experto en cuestión.
- Implementar la comunicación de la GUI con el Sistema Experto.
- Validar la GUI diseñada.

En vistas de cumplir los objetivos propuestos el informe quedará estructurado como sigue:

Organización del informe

Introducción

CAPÍTULO I***: Interfaces Gráficas de Usuarios.*** Este capítulo se dedicara al análisis de los resultados obtenidos, en la búsqueda bibliográfica, acerca de las herramientas para el desarrollo de GUIs.

CAPÍTULO II: ***Interfaz Gráfica de Usuario para "CTExperto".*** Aquí se expondrá los materiales y métodos utilizados para el diseño de la GUI que permitirá mejorar la comunicación del sistema "*CTExperto*" con los usuarios.

CAPÍTULO III: ***Pruebas y análisis de los resultados.*** Se mostrarán los principales resultados obtenidos de la GUI.

Conclusiones

Recomendaciones

Referencias bibliográficas

Anexos

CAPÍTULO 1. INTERFAZ GRÁFICA DE USUARIO

En los albores de la ciencia informática, las interfaces de usuario eran espartanas, muy limitadas por la tecnología existente. Conforme las capacidades de los equipos han ido creciendo en prestaciones, hemos pasado de la época de los terminales de texto, a los menús textuales hasta llegar a las actuales GUIs que añaden capacidades multimedia a la presentación tradicional de datos, lo que constituye un salto cualitativo en la calidad de interacción Hombre-Máquina (Moreno, 2003b).

En la presente investigación se pretende diseñar una GUI para el SE "*CTExperto*" desarrollado en UCSshell 3.0, por lo que se expondrá lo que es un SE y algunas de sus características y se abordarán aspectos importantes en el diseño de GUI.

1.1 Sistemas Expertos

Los SEs son programas que aplican los conceptos de la inteligencia artificial; según (Carlos Soto, 2002) son la pieza comercial y la que más aplicación se le ha dado en Inteligencia artificial.

La idea básica de estos programas es capturar en un ordenador la experiencia de una persona experta en un área determinada del conocimiento, de tal modo que una persona no experta pueda aprovechar esta información (Carlos Soto, 2002).

1.1.1 Características de los SEs

Las características más sobresalientes de este tipo de sistemas según (Sierra et al., 2007) son:

- Amplia difusión del conocimiento

- Fácil modificación
- Respuestas coherentes
- Disponibilidad casi completa
- Conservación del conocimiento
- Capacidad de resolver problemas disponiendo de información incompleta
- Capacidad de explicar los resultados y la forma de obtenerlos

1.1.2 Arquitectura de un SE

No existe una estructura de SE común, en la figura 1.1 se muestra la arquitectura completa de un SE. Sin embargo, la mayoría de los SEs tienen unos componentes básicos: base de conocimientos (BC), motor de inferencia (MI), base de hechos e interfaz de usuario. Muchos tienen, además, un módulo de explicación y un módulo de adquisición del conocimiento (Rolston et al., 1990).

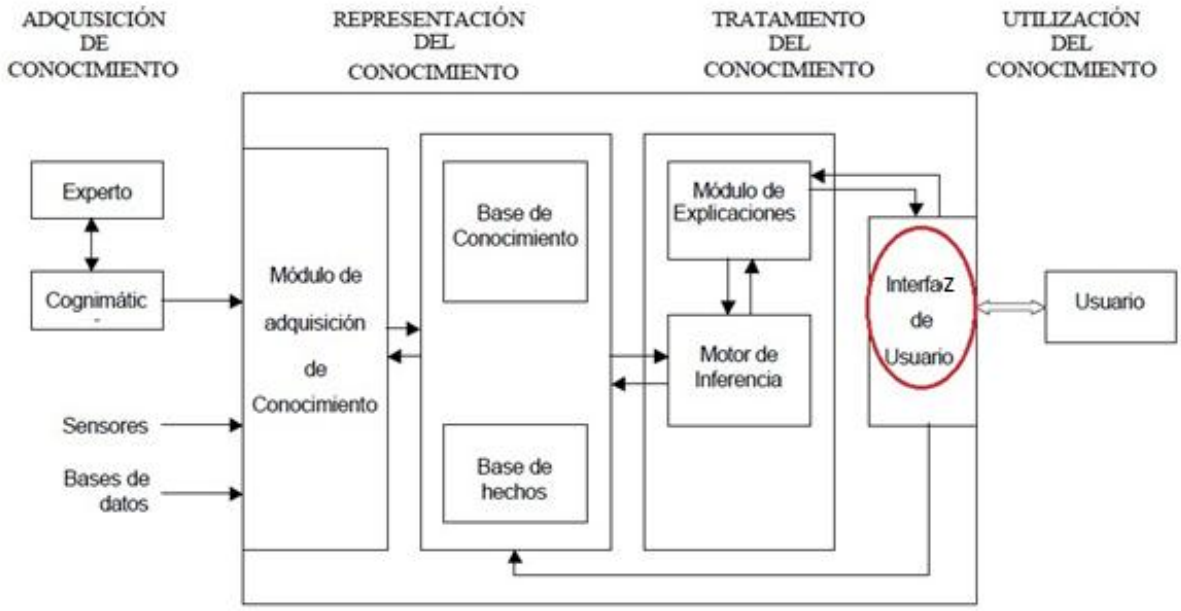

Figura 1.1: Arquitectura completa de un SE.

La estructura de un SE está organizada alrededor de tres elementos principales.

Base de conocimientos:

Es una estructura de datos que contiene una gran cantidad de información sobre un tema específico, generalmente introducida por un experto en dicho tema (se puede asociar a una memoria permanente), sobre el cual se desarrolla la aplicación.

Base de hechos:

Es una memoria auxiliar que contiene a la vez los datos sobre la situación concreta en la cual se va a realizar la aplicación (hechos iniciales que describen el enunciado del problema a resolver) y los resultados intermedios obtenidos a lo largo del procedimiento de deducción.

Motor de inferencia:

Es el núcleo del SE, ya que ponen en acción los elementos de la base de conocimientos para construir los razonamientos. Ejecuta las inferencias (deducciones) en el curso del proceso de resolución, bien sea por modificación, o bien por adjunción de los elementos de la base de hechos.

Aunque el motor de inferencia sea un programa procedimental en el sentido habitual del término, la forma en que utiliza el conocimiento nunca está prevista por el programador.

1.2 Interfaces Gráficas de Usuario

La GUI surge como evolución de las interfaces de línea de comando utilizadas para manejar los primeros sistemas operativos, es el conjunto de métodos que permite lograr la fácil interactividad entre un usuario y una computadora (López, 2009). El usuario común, habitualmente, realiza las acciones deseadas con la manipulación directa de la interfaz, sin necesidad de conocer la programación implícita en el hecho.

1.2.1 Definición

Según (Solano et al., 2011) el concepto de "*Interfaz Gráfica*" se refiere a los modos de interacción entre un programa y el usuario, es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

En el contexto del proceso de interacción usuario-ordenador, la GUI es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción ergonómica con un sistema informático (Solano et al., 2011).

1.2.2 Características de las Interfaces Gráficas

Las características básicas de una buena GUI (Bevan, 1995) podrían sintetizarse en:

- Facilidad de comprensión, aprendizaje y uso.
- Representación fija y permanente de un determinado contexto de acción (fondo).
- El objeto de interés ha de ser de fácil identificación.
- Diseño ergonómico mediante el establecimiento de menús, barras de acciones e íconos de fácil acceso.
- Las interacciones se basarán en acciones físicas sobre elementos de código visual o auditivo (íconos, botones, imágenes, mensajes de texto o sonoros, barras de desplazamiento y navegación...) y en selecciones de tipo menú con sintaxis y órdenes.
- Las operaciones serán rápidas, incrementales y reversibles, con efectos inmediatos.
- Existencia de herramientas de Ayuda y Consulta.

1.2.3 Principios para el diseño de interfaces gráficas

Cuando se decide construir interfaces de usuario, bien sea de modo manual o de manera automatizada por medio de un generador de código, se debe tener muy presentes una serie de principios para intentar alcanzar los niveles de calidad exigibles a las aplicaciones WIMP (Windows, Icon, Menu, Pointer) actuales.

Usabilidad

Según la Organización Internacional de Estándares (ISO) *"La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso."* (Granollers and Lorés, 2004). La usabilidad,

no es más, que una medida de las cualidades percibidas en el software por usuarios muy concretos en un contexto de uso muy localizado.

Ergonomía

Ergonomía: "*Conjunto de estudios, métodos y disposiciones para hacer el trabajo más humano en función de las capacidades fisiológicas y psicológicas del individuo."* (Castillo and López, 1998) Es imprescindible en el desarrollo de las interfaces de usuario. Algo tan simple como el empleo de fuentes gráficas o colores uniformes tiene un peso muy importante en la productividad que el usuario final es capaz de alcanzar con las aplicaciones.

De igual forma que en mecánica se establece un convenio universal para el sentido de giro de tornillos y tuercas, o en electrónica donde se siguen estándares de color para los cables. En informática existen numerosas guías de estilo que tratan de estandarizar aspectos de interfaz de usuario, desde como cerrar o abrir una ventana hasta como debe comportarse un botón o donde debe situarse el menú. Ejemplos remarcables de estas guías son Apple Desktop Interface (Apple Computer, 1987), Common User Access (CUA) (Design, 1992, Berry, 1988), MS Windows 95, OSF/Motif , KDE, Sun Open Look (Microsystems, 1990), Java Look & Feel Design Guidelines (Microsystems, 1990, Look, 1999) y GNOME. El *Workshop Tool for Working with Guidelines* celebrado en 2000 es un claro exponente de la investigación en este campo (Vanderdonckt and Farenc, 2000).

Las guías aportan un modo de proceder preciso acerca de cómo diseñar interfaces de usuario en entornos o plataformas específicas. Estas guías favorecen que la curva de aprendizaje del usuario con una nueva aplicación sea mucho menor.

Dado que CUA sigue siendo la referencia para muchas guías de estilo, merece la pena especificar los principios en los que CUA se basa.

El estándar CUA (Berry, 1988) fue desarrollado por IBM para servir de guía de ergonomía para aplicaciones que deben ejecutarse sobre entornos gráficos. Una aplicación que cumple con los estándares de CUA es más sencilla de usar y es consistente con el entorno gráfico.

Imagen mental del usuario

El principio primordial de CUA se pude reducir a: "*Reproducir la imagen mental del usuario*." Figura 1.2 (Moreno, 2003b)

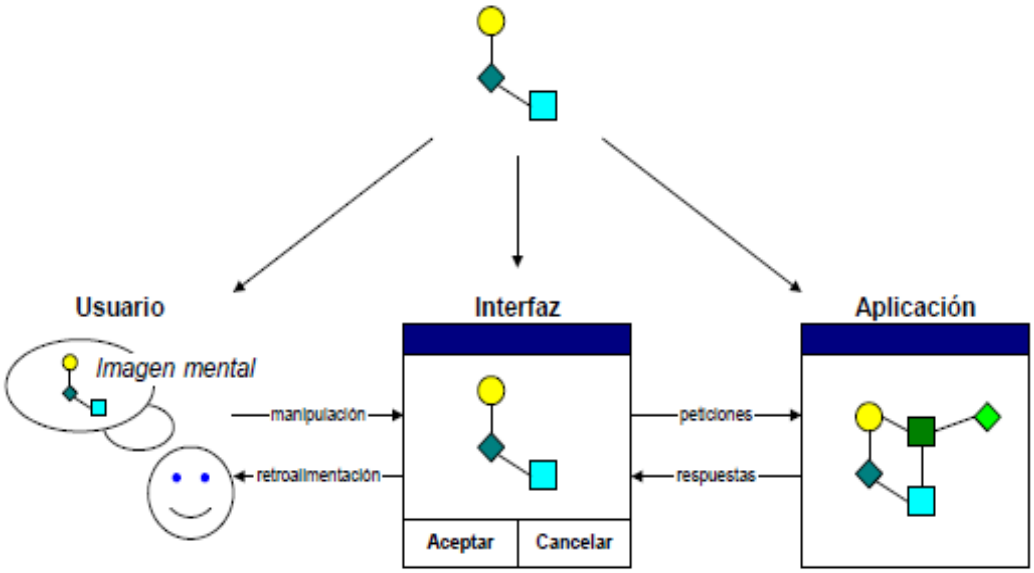

Figura 1.2: Imagen mental del usuario ante la interfaz de usuario.

Principios comunes a toda buena guía de estilo

Según (Moreno, 2003b) el principio de alto nivel: "*Reproducir la imagen mental del usuario*", puede ser refinado y/o complementado en base a otros principios más específicos que a continuación se describen.

Consistencia

La consistencia implica un conjunto de convenciones que han de ser seguidas dentro de la aplicación (intra-aplicación) y entre las aplicaciones (inter-aplicaciones). Los objetos con apariencia o comportamiento similar son interpretados de manera consistente. La consistencia temporal (o estabilidad) también es recomendada por la mayoría de las guías de estilo. Los beneficios aportados por interfaces consistentes pueden resumirse en los siguientes: sistemas más predecibles, mejora de la curva de aprendizaje, uso de distintas aplicaciones sin instrucciones adicionales.

Familiaridad

El principio de la familiaridad puede ser visto como la consistencia con el mundo real. Éste tipo de imitación, ayuda a los usuarios a usar su conocimiento del mundo real para interactuar con el sistema, simplificando el proceso de aprendizaje ya que se apoya en conceptos del mundo real que el usuario ya ha experimentado antes.

Simplicidad

Cuanto más simple mejor. El usuario no debe sentirse desbordado en ningún momento por los detalles. Cuanto más simple es la interfaz, más fácil es que el modelo mental del usuario coincida con el de la aplicación.

Diseño visual

El diseño visual engloba la posición y la apariencia de los objetos. Las guías de estilo favorecen la simplicidad y la claridad en detrimento de diseños recargados o barrocos con vistosos efectos visuales. De este modo, la atención recae sobre la tarea que el usuario realiza evitando despistes innecesarios sobre los detalles y adornos de la interfaz.

Retroalimentación

Todas las acciones del usuario han de tener como respuesta una rápida retroalimentación que sea consistente y entendible para el usuario. De este modo, los usuarios permanecen centrados en su tarea y sienten tener el sistema bajo su control.

Robustez

Los usuarios son propensos a cometer errores, unas veces por la propia naturaleza humana y otras por que el usuario prefiere aprender mediante el método de prueba y error antes que recurrir a leer el manual.

Eficiencia

La eficiencia mide cuán rápido y preciso un usuario puede realizar las tareas en el sistema. Un buen diseño de un sistema debe realizar un análisis de tareas para favorecer el rápido desarrollo de aquellas tareas más frecuentes e importantes. La eficiencia se consigue con la ayuda de otros principios como la consistencia, simplicidad, familiaridad y el diseño visual.

1.3 Problemas de las interfaces inadecuadas y ventajas de las correctas

El abuso o uso incorrecto de las características gráficas de un entorno puede llevar al fracaso de la interfaz diseñada. Algunos problemas más frecuentes según (Moreno, 2003a) son:

Confusión: El usuario está perdido en la aplicación, no sabe dónde ésta ni que es lo que puede hacer. Este es el principal problema de las aplicaciones cuya profundidad o nivel de detalle es excesivo o que no proporcionan mensajes de información durante la ejecución de la operación en curso.

Frustración: El acceso a las acciones no es intuitivo y el usuario no sabe cómo hacer lo que desea. La percepción que el usuario tiene en mente de la aplicación no se corresponde con su interfaz real.

Aburrimiento: Surge cuando el usuario tiene que repetir una y otra vez los mismos pasos para llevar a cabo una acción debido a que la aplicación es inflexible y no dispone de atajos o no está ajustada al trabajo que realiza el usuario.

Desaprovechamiento: La aplicación no se usa en su totalidad debido a que es demasiado compleja.

Sobreexplotación: La aplicación es tan ambigua que es necesario recorrer las ventanas y cajas de diálogo atrás y adelante para asegurarse que la acción a realizar es la correcta. El usuario tiene que abrir y cerrar un gran número de cajas de diálogo o ventanas antes de poder acceder a la funcionalidad que buscaba.

Ventajas

- El uso apropiado de las capacidades del entorno conduce a aplicaciones con interfaces agradables que satisfacen a los usuarios (Moreno, 2003b). Algunos ejemplos de las ventajas producidas son:
- **Mejor aceptación:** El usuario acoge con agrado la aplicación en lugar de rechazarla y la considera herramienta esencial para su trabajo.
- **Mejor uso:** Cuando la interfaz representa fielmente la imagen mental de los usuarios, la aplicación se usa correctamente, minimizando los errores, y por tanto de modo eficiente.

- **Mínimo entrenamiento:** La necesidad de entrenar a los usuarios se minimiza desarrollando aplicaciones consistentes ya que los usuarios pueden reaprovechar los conceptos ya adquiridos en otras aplicaciones y herramientas.
- **Documentación mínima:** Cuando el acceso es intuitivo y natural y la aplicación se corresponde con la imagen mental del usuario, la aplicación es autosuficiente.
- **Reducción en el tiempo de desarrollo y facilidad de mantenimiento:** Una aplicación bien diseñada con una interfaz de usuario adecuada sigue principios de homogeneidad y reutilización, minimizando el número de escenarios (ventanas) con operatoria diferente y reduce el tiempo de desarrollo.

1.4 Herramientas para la construcción de Interfaces de Usuario

En el ámbito de las aplicaciones comerciales se pueden diferenciar dos tipos de aproximaciones para el desarrollo de interfaces de usuario:

Herramientas basadas en modelos. Intentan aumentar el nivel de abstracción. Se apoyan en generadores de códigos parciales o totales.

RAD (Rapid Application Development) y herramientas de autor. Son herramientas de diseño para construir directamente la interfaz de usuario.

1.4.1 Herramientas basadas en modelos

Con el propósito de automatizar la generación de interfaces de usuario, ha mediado de los años noventa, surgió la propuesta de usar modelos para el desarrollo de interfaces de usuario. Dicha propuesta consiste en la utilización de modelos en los cuales se consideran las diferentes etapas del desarrollo de la interfaz de usuario. Su principal objetivo es la construcción de interfaces mediante descripciones de alto nivel de los distintos aspectos de la interfaz: estructura y comportamiento, de modo que a partir de dichos modelos declarativos se pueda generar automáticamente la interfaz de usuario final (Rodríguez et al., 2004). Por modelo se entiende una especificación de la estructura y comportamiento de cualquier elemento software y un modelo es declarativo porque no contienen código sino descripciones a alto nivel de abstracción (Luisa et al., 2010).

La ventaja principal de este método es la expresividad que pueden aportar estos modelos.

Algunos de estos modelos declarativos son el modelo de información, modelo de usuario, modelo de tareas, modelo de diálogo, modelo de presentación, modelo de aplicación, modelo espacio de trabajo, etc.

Con el fin de dar un recorrido breve sobre las herramientas que permiten la generación automática de interfaces de usuario se comentaran algunas de ellas, como Rational Rose, TogetherSoft, Argo UML, Cool:Plex y Genova.

Rational Rose

Rational Rose 2000 es la herramienta CASE (o editor de modelo) de referencia para la especificación gráfica con la notación UML (*Lenguaje Unificado de Modelado*). Las herramientas de Rational Corp. intentan abarcar todo el ciclo de vida para dar soporte a su metodología RUP (Ration Unified Process). Sin embargo, Rational no dispone de herramientas para el soporte directo al desarrollo de interfaces de usuario. Ni en la notación UML, ni en las herramientas de soporte a RUP, se da una respuesta al modelado o diseño de interfaces de usuario.

Together

Together es otra herramienta de modelado de código (modelado y documentación de diseño orientado a lenguajes de 3ra generación) de la compañía TogetherSoft. Similar a Rational Rose, da soporte al modelado de código, generación de clases y permite mantener sincronizado código y modelo de diseño. En la última versión, incluye características tipo RAD (UI Builder) con el cual es posible diseñar interfaces de usuario para el lenguaje Java.

Argo UML

Argo UML (Robbins et al., 1997) es la respuesta del mundo del software de código abierto a herramientas como Rational Rose. Implementado en Java y con licencia de código abierto, permite modelar usando notación UML. La apertura de código facilita la creación de módulos adicionales para la generación de código. Al igual que Rational Rose, no dispone de aspectos específicos para el desarrollo de interfaces de usuario.

Cool:Plex

Cool:Plex es un producto de *Computer Associates*. Es una herramienta comercial destinada al diseño de aplicaciones cliente-servidor independiente de plataforma. Usa componentes y

patrones de diseño que son traducidos en la fase de generación a un lenguaje destino en particular. La interfaz de usuario se diseña al estilo WYSIWYG de modo similar a las herramientas RAD. De la interfaz, lo único que se permite diseñar es la estática de cada ventana o panel.

Genova

Genova 7.0 es un módulo de extensión para Rational Rose desarrollado por Genera A.S. A partir de modelos UML construidos con Rational Rose, Genova permite construir modelos de diálogo y de presentación de modo parcial que representan la interfaz de usuario como un árbol de composición de objeto abstracto de interfaz o AIO. (Abstract Interaction Object).

1.4.2 RAD: Entornos de desarrollo rápido

Esta categoría tiene un menor nivel de abstracción. Aquí se incluyen las llamadas herramientas RAD (Rapid Application Development). Visual Basic, Power Builder (Sybase), Delphi o C++ Builder son brillantes ejemplos de este tipo de herramientas donde la construcción de interfaces de usuario es muy rápida en comparación con las herramientas y lenguajes de 3ra generación previos. En ellas, el programador y/o diseñador de la interfaz de usuario ayudados por un IDE (Entorno integrado de desarrollo) (como pueden ser Eclipse o Visual Studio .NET) van construyendo la interfaz de usuario mediante el paradigma WYSIWYG (*What You See Is What You Get*): eligiendo los componentes de la interfaz, ajustando sus propiedades y programando el enlace con la lógica de la aplicación propiamente dicha.

A continuación expondremos algunas características relevantes de tres de estas herramientas:

QT Designer

QT Designer es una herramienta para el desarrollo de formularios y presentaciones gráficas para las aplicaciones. Permite acelerar el desarrollo de interfaces de alto rendimiento, a la vez que proporciona una forma fácil de diseñar GUIs avanzadas, generando el código fuente para las mismas, lo que permite al desarrollador ajustarlo a sus necesidades.

El QT Designer utiliza como base la librería gráfica de QT, que ha sido transportada a diversas plataformas, lo que permite que el código generado por el QT Designer pueda ser utilizado en diversas plataformas. Además, el QT funciona solo o asociándose a algunos entornos de desarrollo integrado como Visual Studio .Net o Eclipse. Esta herramienta provee características muy poderosas como la previsualización de la interfaz, soporte para widgets y un editor de propiedades bastante poderoso.

Los conceptos básicos de la librería son los widgets (objetos), los slots (o señales) y los eventos. Los widgets son contenedores que pueden contener widgets en cualquier cantidad de niveles. El widget "padre" de dicha jerarquía puede ser cualquiera.

La interfaz para la creación de las GUI se basa en el uso de una paleta que clasifica los diversos objetos que incluye muchos de los widgets de la librería QT y además permite la adición de nuevos widgets (aún creados por el usuario).

El diseño de la interfaz es muy sencillo pues el creador de interfaz utiliza el estilo "Selecciona y dibuja". Es decir, el usuario escoge de la paleta un tipo de widget y luego lo pone sobre el formulario, dibujándolo en la posición deseada y estableciendo el tamaño deseado (Santo Orcero, 2001).

Windows Form Designer

El *Windows Form Designer* es la herramienta de interfaz de usuario de Visual Estudio que permite agregar elementos a un *Form* (Formulario). El elemento y los formularios son parte de *Windows Forms,* que es el nombre que recibe el ***API*** para la GUI incluida como parte de la infraestructura *.NET* de Microsoft. Ello permite que dichos controles puedan ser usados en cualquiera de los lenguajes de programación que sigan la *CLS* (Common Languaje Specification), porque debemos recordar que en la arquitectura *.NET* todos ellos se traducen a un lenguaje intermedio común. No obstante, es importante señalar que los componentes de las *Windows Forms* son dependientes de la plataforma, pues dependen fuertemente del *Win32 API,* por lo que para portarlas a una nueva plataforma (por ejemplo OSX o Linux) sería preciso redefinir las mismas aunque ya en dicha plataforma se pudiera utilizar el *CLR* (Common Languaje Runtime) y el CLS de *.NET.*

Señalada esta limitación que hereda de la infraestructura *.NET,* se puede señalar que el *Windows Form Designer* es una herramienta de diseño de interfaces gráficas robusta y

sencilla. En realidad, provee todas las características señaladas para las herramientas anteriores y las mejora.

Una diferencia importante con la herramienta mencionada anteriormente es que el diseñador no es código abierto y es propiedad de Microsoft. Aunque Microsoft permite la descarga gratuita del *.NET Framework* y una versión simplificada del Visual Studio llamada *Visual Studio Express Edition*. Esto implica que al utilizar este diseñador de pantalla se está asumiendo el riesgo de aceptar que el soporte y desarrollo de la herramienta seguirá estando en manos de Microsoft, a diferencia de las herramientas de código abierto, donde se puede delegar el soporte (y el desarrollo) en manos de cualquier persona o compañía (Alvarez and Valerio, 2009).

NetBeans

La Plataforma NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones (Egas Clavijo, 2015).

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas)
- Administración de las configuraciones del usuario
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato)
- Administración de ventanas
- Framework basado en asistentes (diálogos paso a paso)

Otra característica son los paquetes que componen a NetBeans

NetBeans Enterprise Pack

Provee soporte para la creación de aplicaciones orientadas a servicios (SOA), incluyendo herramientas de esquemas XML, un editor WSDL, y un editor BPEL para servicios web.

PHP

NetBeans permite crear aplicaciones Web con PHP 5, un potente debugger integrado y además viene con soporte para *Symfony* un gran framework MVC escrito en php. Al tener también soporte para AJAX, cada vez más desarrolladores de aplicaciones LAMP o WAMP, están utilizando NetBeans como IDE.

Python

NetBeans permite crear aplicaciones con Python ya que posee un motor para escribir (resaltando la sintaxis), identificar errores y el debugger. Sin duda alguna, NetBeans se ha convertido en un IDE apto para la mayoría de los lenguajes de programación opensource modernos. También se estima que dará para soporte GUI para varias librerías gráficas como son PyQt y GTK. Se espera que salga una versión con motor para soporte para JPython, con acceso a todas las librerías de java e incluyendo soporte para Swing y también para las librerías gráficas de python ya mencionadas.

Ventajas

- La plataforma NetBeans puede ser usada para desarrollar cualquier tipo de aplicación.
- Reutilización de Módulos.
- Permite el uso de la herramienta Update Center Module.
- Instalación y actualización simple.
- Incluye Templates y Wizards.
- Posee soporte para Php.

Desventajas

- Poca existencia de pluguins para esta plataforma.
- Hace falta documentación del Rich Client Plataform (RCP).
- No posee un editor de código HTML.

NetBeans posee, también, un editor de interfaces gráficas el cual está orientado hacia la librería gráfica Swing de Java. Es decir, que únicamente produce código fuente para Java.

Además de dicho código fuente, el generador de interfaz produce unos archivos "*.form*" escritos en formato XML que contienen las especificaciones de los diversos componentes gráficos del usuario. Si bien estos archivos no necesitan ser distribuidos junto con el código para compilar las interfaces, sin ellos es imposible para el editor gráfico reconstruir el diseño de la pantalla para una edición posterior. Por ello tampoco el editor puede generar un archivo "*.form*" para clases GUI no creadas por NetBeans (aunque hay herramientas en desarrollo para ello).

Es importante señalar que el generador de GUI del NetBeans (y en general todo el IDE) ha mostrado un sorprendente desarrollo desde su inicio y cada vez se le agregan nuevas funcionalidades. Por ello, si bien tiene algunas desventajas, posee considerables características interesantes y permite generar interfaces agradables en poco tiempo, además de que se encuentra respaldado por el principal desarrollador del lenguaje Java, como lo es Sun Microsystems.

El hecho de generar código que usa librerías estándar de Java dota a las interfaces realizadas en dicho editor de una gran portabilidad, pues únicamente dependen de que exista una JVM (Java Virtual Machine) apropiada.

1.5 Patrones Arquitectónicos

Los patrones, en el desarrollo de software, son recursos útiles para identificar la solución de un problema concreto, pues en ellos se encuentra documentada la experiencia acumulada de los diseñadores de software en solucionar ciertos tipos de problemas. Existe una amplia gama de patrones en la Ingeniería de Software (IS), cada uno centrado en algún esbozo general de un tipo de problema, que puede ser de diseño, de análisis, de arquitectura, entre otros. Los problemas referentes a la arquitectura de software abordan la selección de los elementos estructurales del sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones y su composición (Bass et al., 2003.). Un patrón de arquitectura especifica una serie de subsistemas y sus responsabilidades respectivas, e incluyen reglas para organizar las relaciones entre ellos. En esencia expresan un esquema organizativo estructural fundamental para sistemas de software (Mitchell, 2003).

Dentro de los patrones arquitectónicos se destaca la arquitectura Modelo-Vista-Controlador (MVC), la cual separa la lógica de negocio de la interfaz gráfica de la aplicación y se auxilia de un módulo responsable de la gestión de los eventos sobre esa interfaz.

1.5.1 Descripción de la Arquitectura MVC

En la Figura 1.3 se muestra una perspectiva de las relaciones entre las componentes del patrón MVC, sin embargo, existen otras implementaciones de esta arquitectura que proponen dependencias y asociaciones diferentes.

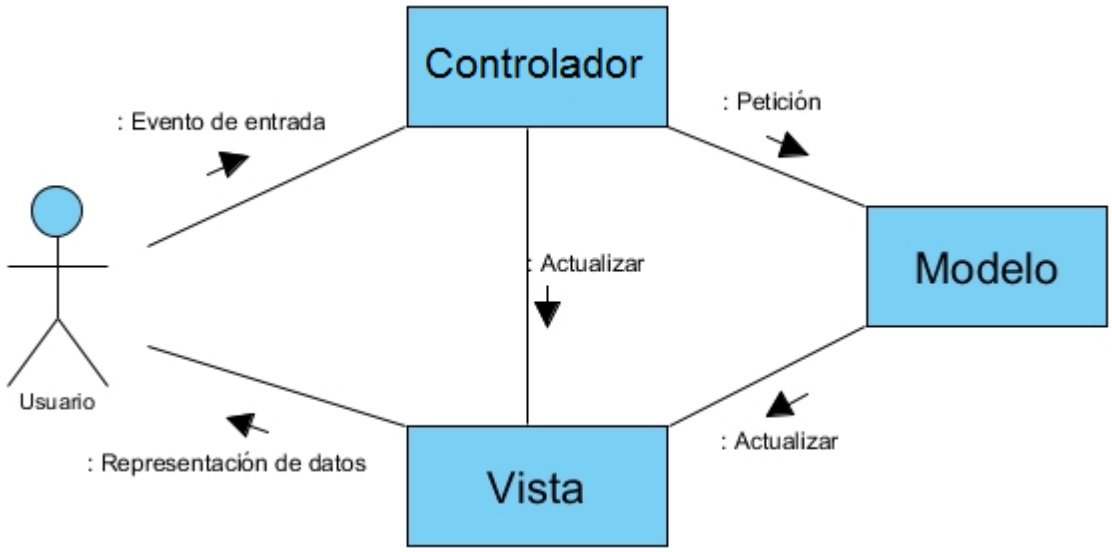

Figura 1.3: Relaciones entre las componentes del patrón MVC.

Seguidamente se describen el **Modelo**, la **Vista** y el **Controlador** con énfasis en sus responsabilidades dentro del flujo de una aplicación basada en dicha arquitectura.

Al **Controlador** lo integran las clases responsables de procesar los eventos generados por el usuario desde la Vista, para luego ser traducidos en solicitudes al Modelo. Estas solicitudes pueden implicar cambios en los datos internos de la aplicación o solo comprender la respuesta a una consulta, como también pueden significar cambios en la propia Vista desde donde fueron generadas (Freeman, 2004).

El **Modelo** está compuesto por el conjunto de clases que gestionan la lógica de negocio de la aplicación, por esta razón encapsula los datos y funcionalidades principales del sistema,

sin tener en cuenta la representación específica de la salida o el comportamiento de la entrada de la aplicación (Li and Cui, 2005). Esta componente no dispone de referencias al Controlador o a la Vista, sino que el propio sistema es el encargado de proveer las interfaces necesarias para la comunicación entre el Modelo y sus Vistas, cuando ocurra un cambio en los datos por el accionar del Controlador.

La **Vista** refiere a la interfaz de usuario, dado que es la componente encargada de visualizar los datos, de forma adecuada, en respuesta a las solicitudes del usuario. Su interacción con el Modelo ocurre a través de una referencia al propio Modelo, pero se limita a la realización de consultas sobre la información contenida, puesto que esta relación no implica modificaciones en dicha información (Gamma et al., 1993). En la práctica, es común la existencia de diferentes instancias de la Vista para un mismo conjunto de datos del Modelo.

1.6 Proceso de diseño en la Ingeniería de Software

Según Taylor (Taylor, 1959) el diseño no es más que el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso, o sistema, con los suficientes detalles como para permitir su realización física. El diseñador tiene la responsabilidad de obtener un modelo que se construirá en una etapa posterior.

Dentro del ciclo de vida de la IS, el diseño de software es la actividad en la que los requerimientos de un producto informático son analizados para obtener una descripción de su estructura interna, que se usará como base para su posterior construcción (Kendall and Kendall, 2005). Este proceso es de vital importancia para la obtención de productos informáticos que satisfagan las necesidades de los clientes.

1.6.1 Descomposición del Diseño de Software

Desde el punto de vista técnico, Cataldi (Cataldi, 2000) descompone este proceso en cuatro subprocesos:

El diseño de datos, modela el dominio de la información e incluye las estructuras de datos necesarias para la implementación del software.

El diseño arquitectónico, especifica las relaciones entre las componentes estructurales de la aplicación.

El diseño procedimental, transforma los elementos estructurales de la arquitectura en una descripción algorítmica.

El diseño de las interfaces, establece los mecanismos de comunicación del sistema con otras aplicaciones y con los usuarios.

1.7 Consideraciones parciales

En el presente capítulo se han expuesto los conceptos fundamentales sobre el diseño de GUIs. Además se han abordados algunos conceptos útiles para el desarrollo de esta investigación, como por ejemplo los patrones de arquitectura, específicamente el estilo MVC; tomándolo como referencia debido a que Java proporciona soporte para la implementación de dicha arquitectura en aplicaciones desarrolladas en ese lenguaje y dispone de la biblioteca Swing que es considerada un framework MVC para el diseño y desarrollo de GUIs.

Se estimó conveniente elegir NetBeans IDE 8.0.2 para el desarrollo de nuestra GUI, por ser una herramienta donde la construcción de interfaces de usuario es relativamente rápida, es código abierto, existe bastante documentación, debido a que tiene un gran número de usuarios que lo utilizan y se decidió trabajar con él, principalmente, por generar código java al igual que UCShell 3.0, donde se encuentra compilado el prototipo del SE.

CAPÍTULO 2. INTERFAZ GRÁFICA DE USUARIO PARA *"CTExperto"*

Más allá de su finalidad esencial de interacción entre hombre y máquina, las interfaces de usuario conforman también la cara visible o escaparate de las aplicaciones tal y como las percibe el usuario final que las usa (Moreno, 2003b). Por lo cual, es indispensable contar con GUIs con un mínimo de calidad para ofrecer al usuario sensaciones de seguridad, fiabilidad, ergonomía, sencillez de uso y precisión en la aplicación suministrada.

En el presente capítulo se expondrán las particulares de la herramienta utilizada en el desarrollo de la aplicación *"CTExperto"* y se detallan las principales características de diseño del sistema.

2.1 Introducción al trabajo con NetBeans

Como se ha dicho anteriormente la herramienta seleccionada para la realización de este proyecto fue Netbeans, dado sus características en el desarrollo de interfaces y las particularidades del prototipo del SE.

2.1.1 Iniciando NetBeans

Para poder utilizar GUI Builder en un proyecto, éste se debe crear de una manera específica, siguiendo los pasos que se detallan a continuación:

1. Elegimos "File > New Project" o se pulsa directamente en el botón de "New Project" en la barra de herramientas.
2. Se selecciona la carpeta que pone "Java", y de ahí la opción que dice "Java Application". pulsando "Next".

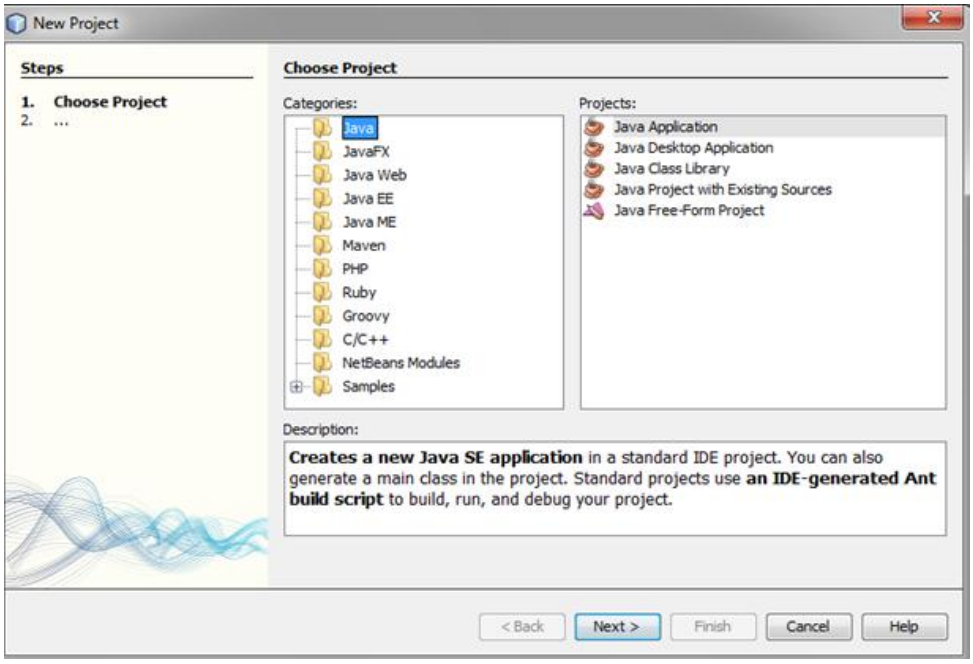

Figura 2.1: New Project.

3. Se introducen los datos deseados en el formulario, dejando o no marcadas las casillas "Create Main Class" y "Set as Main project". Finalmente se da click en el botón "Finish".

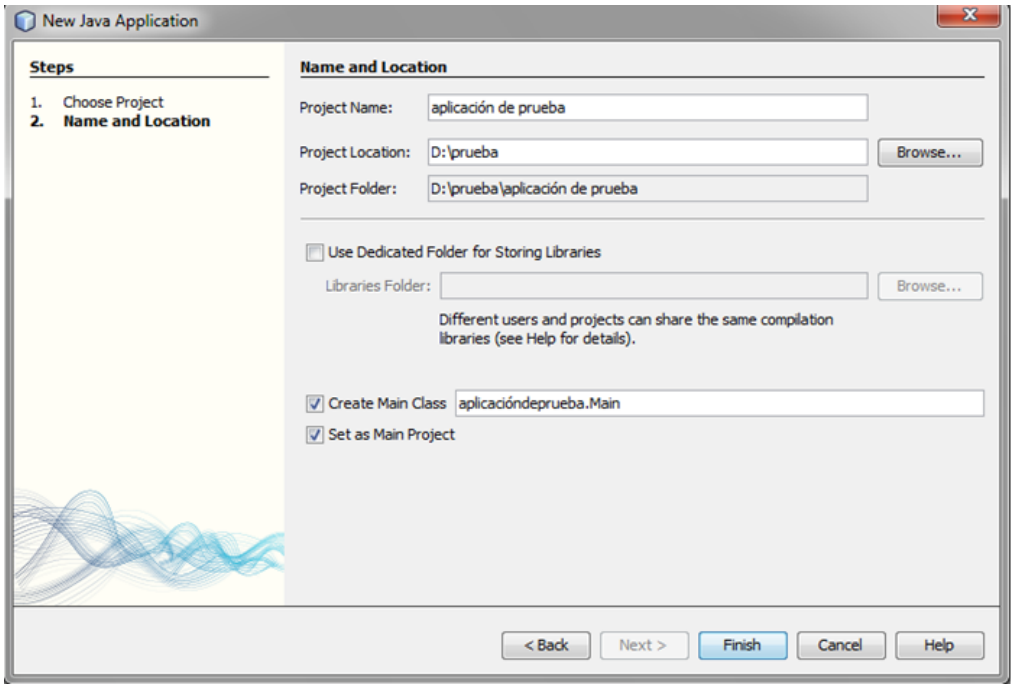

Figura 2.2: New Java Application.

4. Ahora se crear el archivo que contendrá la GUI. Para ello, se da click, con el botón derecho sobre el ícono del proyecto que fue creado anteriormente y se elige "New > JFrame Form".

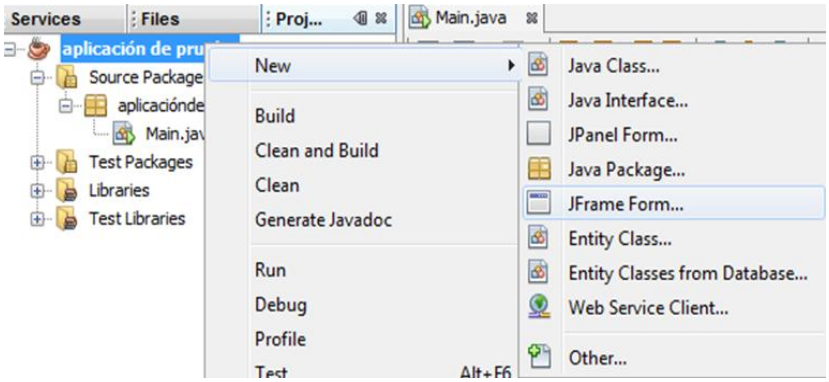

Figura 2.3: Agregar una nueva JFrame Form.

5. Se introducen los datos en el formulario poniéndole el nombre deseado y el paquete que contendrá la interfaz. Finalmente se presiona el botón "Finish".

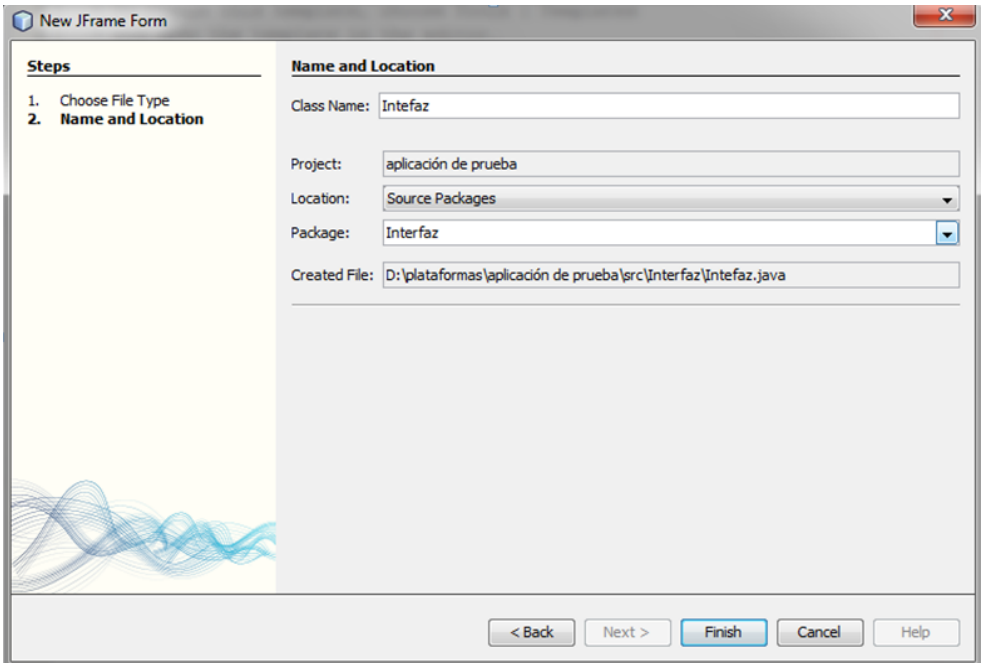

Figura 2.4: New JFrame Form.

6. Seguidamente se abrirá el archivo que fue creado, mostrando la interfaz del GUI Builder.

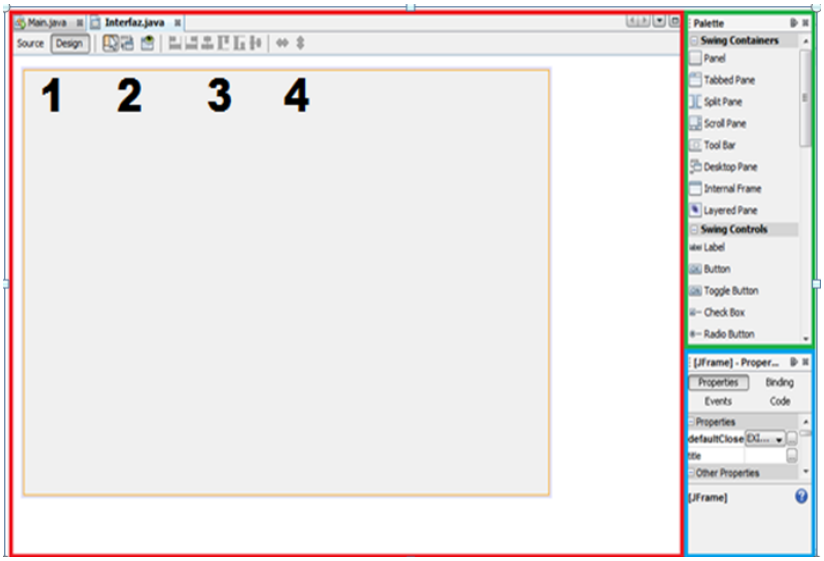

Figura 2.5: Interfaz del GUI Builder.

Zona de Diseño

En la figura 2.5 se muestra la zona de diseño, contenida en el cuadro rojo. En ella se colocaran los distintos elementos de la interfaz que se desea crear, cambiando sus parámetros (color, tamaño, etc). En la parte superior tiene una barra de herramientas con diferentes elementos:

1. Source – Design: estos dos botones permiten cambiar entre modo de vista de código y gráfico. En el gráfico se pueden añadir los distintos elementos de la interfaz, mientras que en el de vista de código se le da funcionalidad.
2. En esta parte presenta 3 botones muy útiles.
    - Modo Selección: permite seleccionar los elementos de la interfaz y moverlos o cambiar su tamaño.
    - Modo Conexión: en él se define la relación entre dos elementos de la interfaz, sin tener que entrar en la vista de código.
    - Vista previa: aparece una interfaz preliminar para poder evaluar su funcionalidad antes de compilar.
3. Botones de autoajuste: permiten alinear los elementos de la interfaz de forma automática.
4. Pautas de auto redimensionamiento: indican si al ampliar la ventana principal de la interfaz los elementos que contenga se redimensionan con ella o no.

Paleta

Es la parte contenida en el rectángulo verde. En ella se elige qué nuevo elemento añadir al diseño de la interfaz. Contiene varios apartados diferenciados:

- **Swing Containers:** en esta parte se encuentran los elementos que sólo sirven para contener a otros, pero que por sí mismos no hacen nada. Ejemplos de ello son las barras de herramientas o los menús tabulares.
- **Swing Controls:** aquí se almacenan los elementos mediante los que se crea o almacena información y órdenes, y que pueden estar contenidos en los descritos anteriormente o no. Como ejemplos, se tienen botones, etiquetas, barras deslizantes o tablas.
- **Swing Menús:** aquí hay distintos elementos que ayudan a la creación de barras de menús, añadiendo los menús propiamente dichos y sus elementos.
- **Swing Windows:** en esta sección tenemos una serie de ventanas que aparecen como diálogos, y que son útiles a la hora de crear eventos de advertencia y similares.
- **AWP:** estos elementos son similares a los visto con anterioridad en Swing Containers y Swing Controls, pero con un diseño anticuado y simple, pues AWP es una biblioteca de clases para el desarrollo de interfaces anterior a Swing, la cual no es ni más ni menos que una extensión de la primera.
- **Java Persistence:** en este último apartado se hallan los elementos correspondientes a consultas sobre bases de datos.

Propiedades

Es la parte que está en el recuadro azul de la figura 2.5. Aquí se mostrarán las propiedades del elemento seleccionado en ese momento, permitiendo a su vez su alteración.

A continuación se detallaran algunas de las propiedades más relevantes de los distintos componentes utilizados en la aplicación:

background: Modifica el color de fondo del componente.

border : Modifica el borde del componente.

foreground: Modifica el color del texto del componente

Font: Modifica el estilo y el tamaño de la letra en el componente.

icon: Permite colocar un ícono o mostrar una imagen en el componente.

enabled: Permite habilitar o no el componente

opaque: Permite darle transparencia al componente.

selectionModel: En el caso de las listas (List) permite cambiar el modo de selección.

lineWrap : Permite ajustar las líneas de texto al tamaño de las área de texto (TexArea).

wrapStyleWord: Permite ajustar las líneas de texto al tamaño de los TexArea teniendo en cuenta los límites de las palabras.

2.2 Diseño de la arquitectura de "*CTExperto*"

La arquitectura de software define un sistema en términos de elementos computacionales, detallando además, las interacciones entre ellos. Los elementos computacionales son entidades tales como bases de datos, filtros o capas de un sistema jerárquico (que son los módulos definidos de acuerdo a las funcionalidades de la aplicación). Las interacciones que ocurren entre estas componentes pueden ser complejas, por ejemplo los protocolos del modelo cliente/servidor, o tan sencillas como los modificadores de acceso de un objeto y las llamadas a sus métodos.

Diseñar la arquitectura de un software, según los autores Garlan y Shaw (Garlan and Shaw, 1993), implica definir sus aspectos estructurales como un conjunto de componentes, las jerarquías de control, los protocolos de comunicación, la sincronización y acceso de los datos, la asignación de la funcionalidad a los elementos del diseño, la composición de estos elementos, su distribución física, su escalabilidad y su desempeño.

2.2.1 El patrón de arquitectura MVC y el sistema "*CTExperto*"

El patrón arquitectónico MVC, como ya se ha mencionado anteriormente, separa la lógica de negocio de la interfaz visual de la aplicación, lo cual puede ayudar a obtener un diseño de la aplicación con niveles apropiados de cohesión en sus módulos. Este estilo de arquitectura es muy utilizado en el diseño de interfaces gráficas, debido a los bajos niveles de acoplamiento que generalmente permite alcanzar, lo cual es conveniente para obtener ventajas sustanciales en cuanto al diseño.

La aplicación de la arquitectura MVC en el sistema "*CTExperto"* se refleja en la estructura de sus módulos. Su diseño, aporta claridad y facilidades para el mantenimiento del código fuente, en caso de que fuese necesario en un futuro.

Figura 2.6: Estructura de los módulos en "*CTExperto*".

La figura 2.6 muestra la organización de las clases según la arquitectura MVC. El paquete "bomba.controladores" representa al **Controlador**, pues contiene las clases que se encargan de la manipulación de los datos que maneja la aplicación, por ejemplo las clases BomboDisplayer y BombaQuestionMaker son responsables de visualizar las acciones DISPLAYER y visualizar la inferencia de las bases de conocimientos respectivamente. El **Modelo** está integrado por la máquina de inferencia (el paquete Model_inference), que se encuentra en la propia librería UCShell Library. Las funciones de la **Vista**, fueron delegadas en el paquete "bomba.vista", donde se agrupan el conjunto de clases que representan las interfaces visuales de la aplicación, esto incluye los diálogos de interacción con el usuario durante la ejecución del SE. En la estructura de "*CTExperto*" también se incluyen otros paquetes ("bomba.resources") que agrupan recursos estáticos de la aplicación, como por ejemplo los archivos que conforman la ayuda del sistema.

Para concebir el sistema son imprescindibles un conjunto de componentes, las cuales se exponen en el siguiente apartado.

2.3 Componentes del sistema "*CTExperto*"

Un componente es una parte física de un sistema que se encuentra en la computadora y no en la mente del analista (Pressman and Troya, 1988).

El sistema cuenta con una componente que es un ejecutable de java, constituye la interfaz que intercomunica las diferentes componentes. Para acceder a realizar un diagnóstico u obtener explicaciones, el sistema interacciona con otros dos componentes: UCShell Library y las bases de conocimientos compiladas (ficheros .kbo). La figura 2.7 describe el diagrama de componentes.

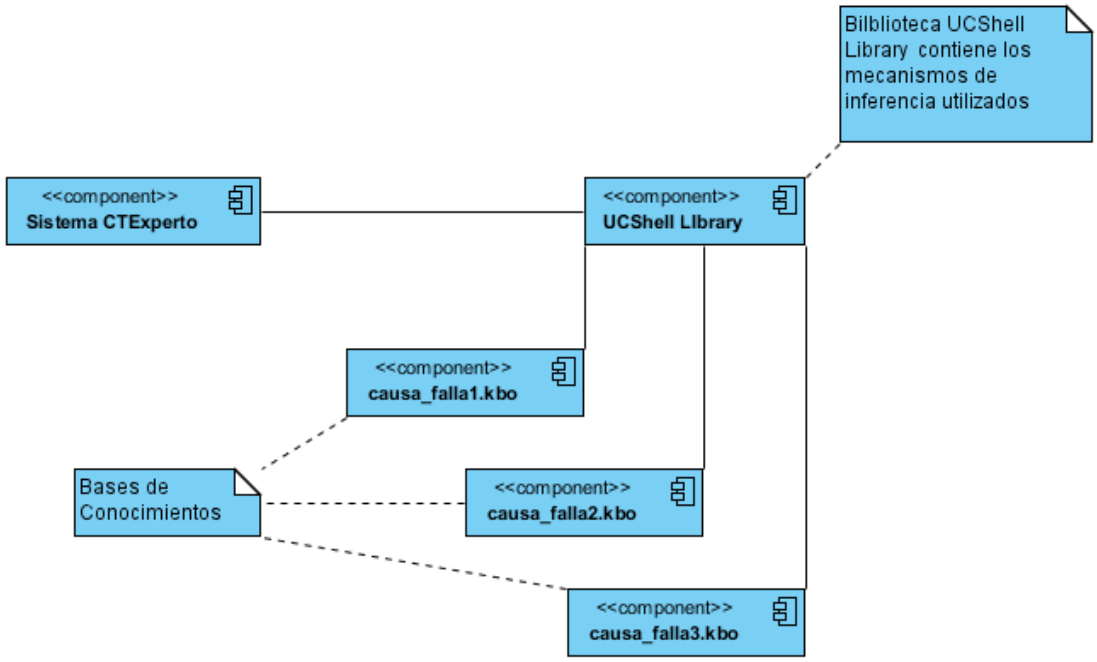

Figura 2.7: Diagrama de componentes del sistema.

Biblioteca UCShell Library

Esta biblioteca permite incorporar el mecanismo de inferencia de UCShell a otras aplicaciones. Contiene, entre otras funcionalidades, un compilador que analiza si la base de conocimiento está sintácticamente correcta y una máquina de inferencia DEPTH FIRST con dos direcciones de búsqueda.

UCShell Library, agrega los archivos "*UCShell IDE 3.0.jar*" es la biblioteca propiamente dicha.

Para inferir hay que hacer uso del paquete "Model_Inference", este contiene la clase "Inference" que es la que implementa los métodos para inferir y además carga la forma interna de la BC. Ver ***Anexo I. Clase "Inference"***

Bases de conocimientos: causa_falla1.kbo, causa_falla2.kbo y causa_falla3.kbo

Estos ficheros contienen las bases de conocimiento compiladas. Son accedidos a través de la clase "Inference" contenida en el paquete "Model_Inference", como se ha dicho anteriormente.

2.3.1 Diagrama de clases

Las clases que conforman a "*CTExperto*", están agrupadas según las tres componentes que comprende el patrón arquitectónico MVC: el Modelo (que contiene la máquina de inferencia), la Vista (que gestiona la interfaz gráfica) y el Controlador (que intercambia información entre la Vista y el Modelo).

El método **void initInference** (QuestionMaker iQuestionMaker, Displayer iDisplayer) es necesario llamarlo antes de realizar la inferencia. En él se le asigna a la clase "*Inference*" los objetos que implementan la interfaz: **Displayer** y **QuestionMaker**.

Luego se llama al método **boolean infer** (**String** filename, **String** errorsFileName, **String** standardOutInferenceFileName, **String** currProjectName): Este método es el que carga la forma interna y realiza la inferencia, devolviendo true o false si tiene éxito o no. Los parámetros que se le pasan al método son los siguientes:

String fileName: El nombre del archivo que contiene la forma interna de la base de conocimiento.

String errorsFileName: el nombre del archivo donde se guardarán los errores detectados durante la inferencia.

String standardOutInferenceFileName: El nombre del archivo donde se guardarán los mensajes enviados por el sistema, en caso de que se envíe alguno.

String currProjectName: La dirección del proyecto actual.

De manera general, cuando se quiere inferir debe crearse un objeto de la clase "*Inference*", luego llamar al método **initInference** pasándole por parámetros los objetos *displayer* y *questionMaker*. Finalmente se llama al método **infer** con los parámetros requeridos. Una vez terminada la inferencia los hechos probados se encontrarán en el atributo *facts* de la clase "*Inference*".

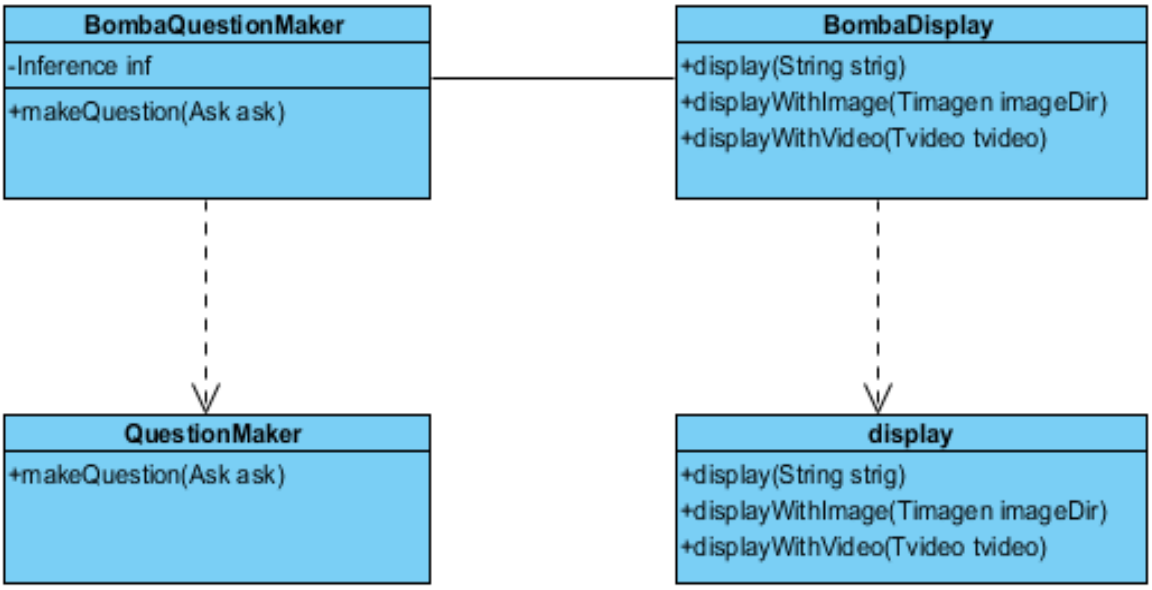

Figura 2.8: Diagrama fundamental de clases

BombaQuestionMaker: esta clase se encarga de visualizar la inferencia en la base de conocimiento. En el proceso de diagnóstico se captan datos de dos tipos de variables, que son donde se colocan los atributos que se preguntan. Estos pueden ser: atributos de preguntas de Selección, donde se toma una opción, y los atributos de Introducir Datos.

Algunos atributos que toman los valores de la forma interna y se utilizan son:

- **BombaQuestionMaker** *questionMaker*: Objeto que implementa la interfaz **QuestionMaker**. Define el comportamiento de las ventanas con las que interactúa el usuario cuando responde las preguntas. El método a implementar en esta interfaz es:

- **Fact makeQuestion** (**Ask** *ask*): En este método se define la forma en que se interrogará al usuario y recibe, por parámetros, un objeto de tipo **Ask** que contiene toda la información relativa a la pregunta. Una vez contestada la interrogante se devuelve un objeto de tipo **Fact** que contiene los detalles de la respuesta dada por el usuario.

BombaDisplayer: se encarga de visualizar cada una de la acciones DISPLAY de la base de conocimiento.

- **BombaDisplayer** *displayer*: Objeto que implementa la interfaz **Displayer**. Define cómo se mostrarán los datos al usuario: textos, imágenes o videos. Los métodos a implementar son:

- **void display** (String cadena): Se define cómo el sistema mostrará una cadena.
- **void displayWithImage** (String imageDir): Se define cómo el sistema mostrará una imagen, la dirección de la misma es imageDir.
- **void displayWithVideo** (String video): Se define cómo el sistema mostrará un video. La ubicación del video está contenida en la cadena video.

2.3.2 Diagrama de actividades

Para explicar cómo proceder a realizar el diagnóstico se utiliza el diagrama de actividad representado por la figura 2.9. Una vez seleccionado el síntoma, procede a realizar diagnóstico y se obtiene el resultado.

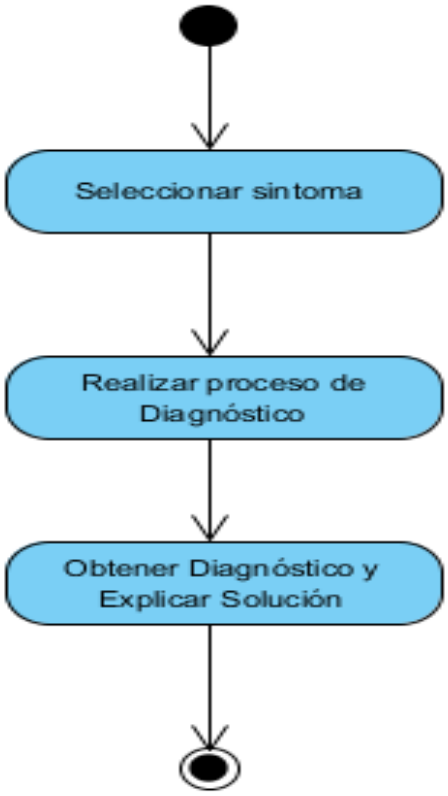

Figura 2.9: Diagrama de actividades

2.3.3 Diagrama de caso de uso

La Figura 2.10 muestra el diagrama de casos de usos, los usuarios y las funcionalidades asociadas a cada uno, donde la diferencia principal en los roles definidos radica en que solo el experto podrá modificar la base de conocimiento del sistema.

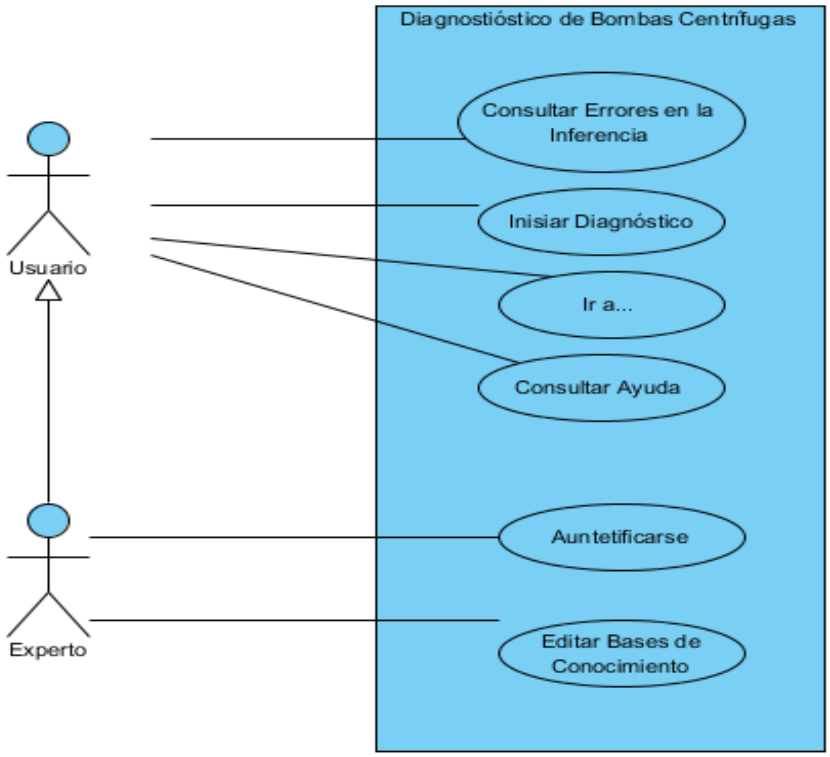

Figura 2.10: Diagrama de actores y casos de uso.

Caso de uso Consultar Errores en la inferencia:

Intervienen: Usuario y Experto.

Propósito: Ver los errores en la inferencia, en caso de que existan.

Descripción: El usuario o el experto pueden consultar los errores del sistema en caso de que se produzca alguno.

Caso de uso Iniciar Diagnóstico:

Intervienen: Usuario y Experto.

Propósito: Realizar la inferencia de las fallas.

Descripción: Es el caso de uso principal del sistema, incluye la selección del síntoma, y como resultado del proceso se muestra el diagnóstico.

Caso de uso Ir a…:

Intervienen: Usuario y Experto.

Propósito: Navegar dentro de la aplicación.

Descripción: El usuario puede acceder a cualquier ventana de la aplicación o salir de ella si lo desea.

Caso de uso Consultar Ayuda:

Interviene: Usuario y Experto.

Propósito: Ofrecer una ayuda para el uso de la aplicación.

Descripción: El usuario puede consultar la ayuda para facilitar el aprendizaje de la aplicación. También cuenta con un" *Manual para el mantenimiento de bombas centrífugas"*, con el cual el usuario puede informarse sobre cualquier duda y ejercitarse con ejercicios propuestos en el documento.

Caso de uso Autentificarse:

Interviene: Experto.

Propósito: Obtener los permisos necesarios para realizar las distintas operaciones que puede efectuar en el sistema.

Descripción: El actor introduce los datos de identificación. Si se verifica que los datos introducidos son correctos entonces es identificado como experto del sistema, por tanto tiene control total de realizar cualquier cambio en las bases de conocimiento.

Caso de uso Editar Bases de Conocimiento:

Actor que interviene: Experto.

Precondición: El usuario debe haberse autentificado como experto del sistema.

Propósito: Permitir al experto administrar la BC.

Descripción: El experto puede editar la base de conocimiento, esto incluye tanto agregar como eliminar reglas, hacer modificaciones en imágenes complementarias o en opciones de reparación de fallas y realizar actualizaciones de la BC.

2.4 Generalidades de la aplicación "*CTExperto*"

Antes de comenzar a describir el funcionamiento de la interfaz gráfica de la aplicación "*CTExperto*" es necesario abordar algunos aspectos de carácter general relacionados con el software, que deben ser dominados por el usuario. Vale señalar que el objetivo principal de este software es facilitar la comunicación entre el SE y los usuarios u operadores que trabajen con bombas de alimentación de agua, y a su vez servir de apoyo técnico y de conocimiento; por ende, la misma va destinada a un grupo de usuarios que deben dominar los principales conceptos relacionados con el tema.

Para obtener respuestas más acertadas del software, es importante que el usuario conozca las principales partes que conforman las bombas centrifugas y algunos aspectos relacionados con su funcionamiento. En esta aplicación solo se trabajan sobre tres síntomas fundamentales: sobrecalentamiento, vibraciones y esfuerzo excesivo de la bomba.

2.4.1 Análisis de la interfaz gráfica

La interfaz de la aplicación "*CTExperto*" se diseñó para dos tipos de usuarios con diferentes roles: uno es el experto, que suministra el conocimiento y que puede modificar la base de conocimiento, accediendo a su vez al proceso de diagnóstico y el otro es el técnico, que puede realizar el diagnóstico de la falla y obtiene los resultados. La interfaz se encuentra conformada por dos ventanas principales y un conjunto de ventanas auxiliares utilizadas para la comunicación con el usuario; díganse mensajes de error, avisos, preguntas y cuadros de diálogo. En la figura 2.11 se muestra la pantalla inicial del sistema, un técnico podrá acceder a las distintas funcionalidades del sistema a excepción de la referida a la modificación de la base de conocimiento. Para editar la base de conocimiento será necesario autentificarse y solo al ser reconocido como experto, podrá proceder a dichos cambios.

Figura 2.11: Ventana principal del sistema.

La ventana principal, es por tanto, accesible por todos los usuarios del sistema y con la misma se conduce a iniciar el diagnóstico de la bomba. La interfaz en general se diseña siguiendo el principio fundamental de "Reproducir la imagen mental del usuario" del estándar CUA, con auxilio de una barra de menús y un botón de inicio. Cumpliendo con otro de los principios comunes de cualquier guía de diseño, "*Simplicidad*", con lo cual se facilita el aprendizaje del sistema.

La otra ventana principal de la interfaz "*Síntomas*", considerada la más importante, ya que en ella se desarrolla la aplicación, es aquella a la que se accede una vez presionado el botón "*Iniciar Diagnóstico*" o el menú "Síntomas" de la barra de menús de la primera ventana. En ella se visualizan los síntomas de los cuales se tiene información en las BC. Cada síntoma se acompaña de una imagen que ayude al usuario a identificarlo rápidamente. Cumpliendo así con el principio de "Familiaridad" que ayuda a los usuarios a usar su conocimiento del mundo real para interactuar con el sistema. El usuario puede presionar sobre el síntoma propiamente dicho y comenzar el proceso de diagnóstico. Para ello la aplicación se auxilia

en cuadros de diálogos como por ejemplo en el de la figura 2.12 donde se muestra una pregunta con opciones.

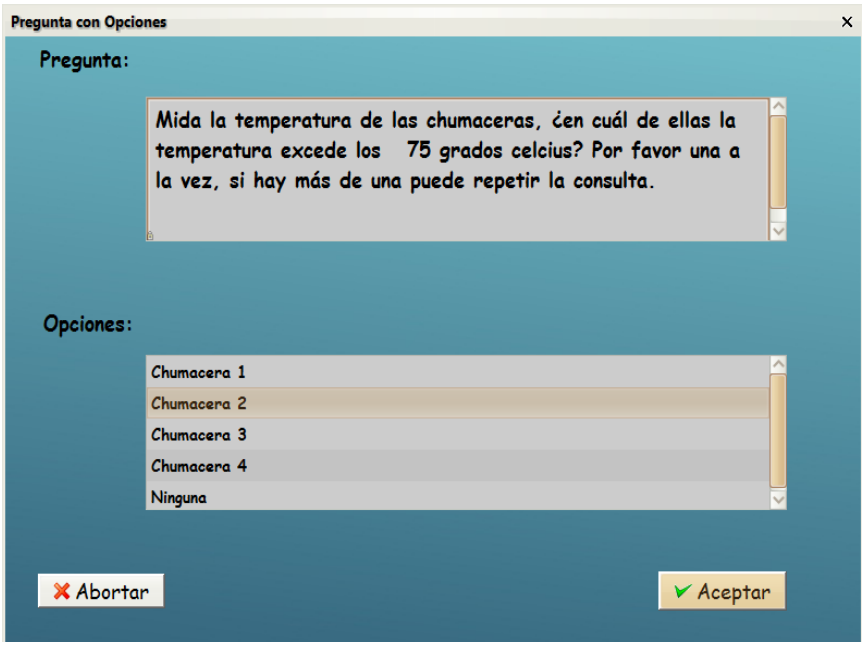

Figura 2.12: Ventana "Pregunta con Opciones".

En este caso si no se realiza la selección antes de presionar el botón "Aceptar" saldrá un mensaje de aviso Figura 2.13. Teniéndose en cuenta otro principio "Retroalimentación" donde todas las acciones del usuario deben tener como respuesta rápida que sea consistente y entendible para el usuario.

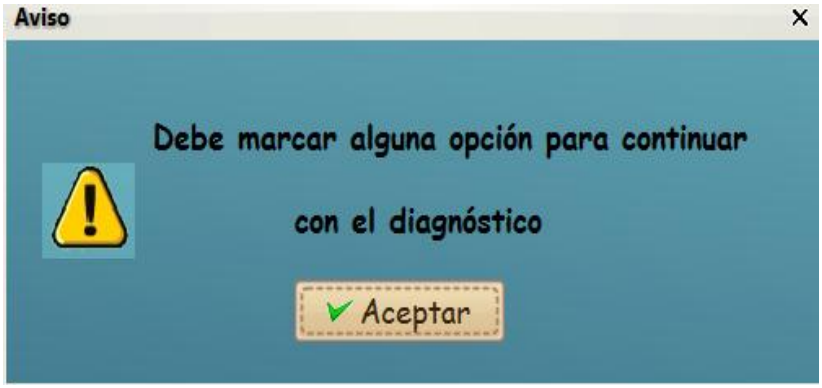

Figura 2.13: Mensaje "Aviso".

Otro tipo de pregunta que el sistema utiliza son las de introducir un valor Figura 2.14. En este caso el usuario edita su respuesta. El botón "*Aceptar*" solo se activa si se introduce un valor numérico.

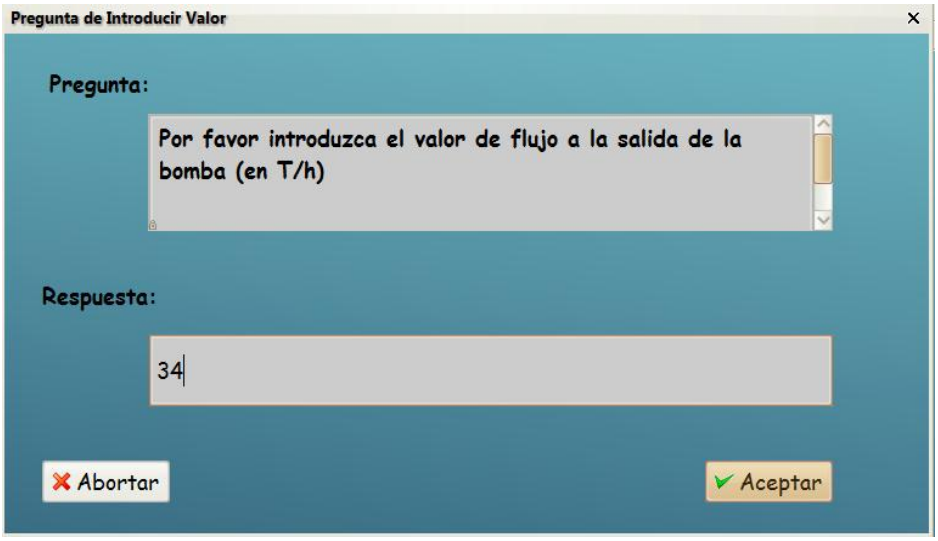

Figura 2.14: Ventana "Pregunta de Introducir Valor".

Los resultados del diagnóstico se visualizan en una ventana como se muestra en la figura 2.15. En este caso el resultado está acompañado de una imagen de la causa.

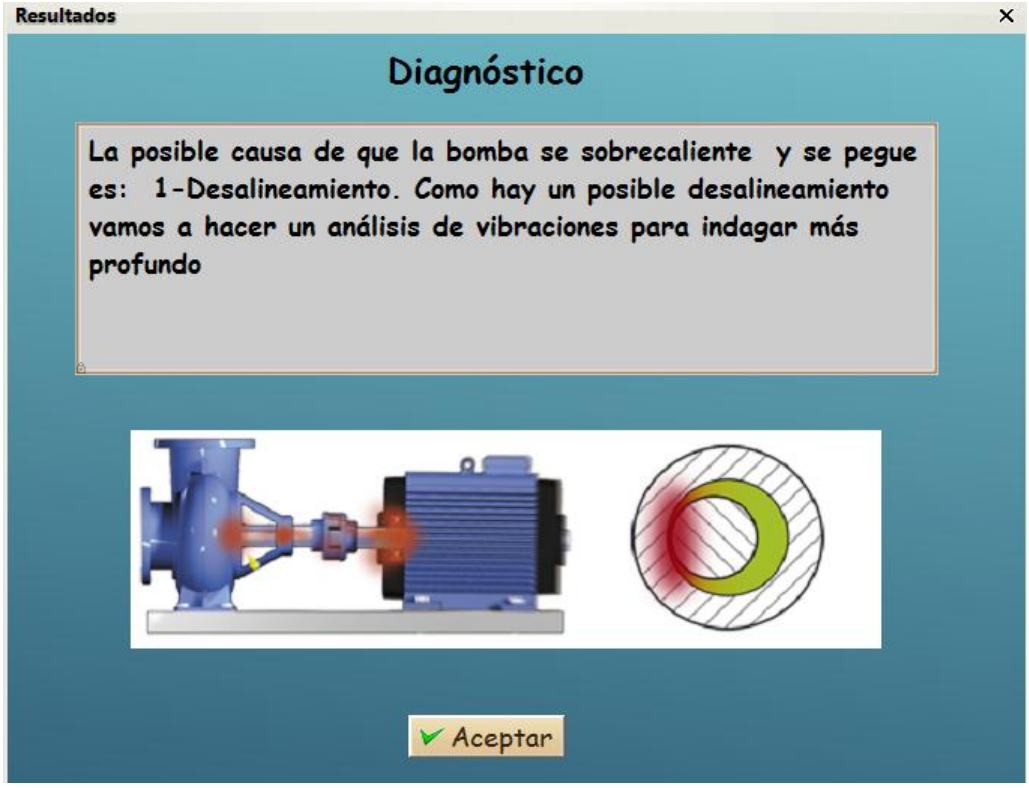

Figura 2.15: Ventana "Resultados".

De esta manera el usuario puede realizar el proceso de diagnóstico de forma sencilla y rápida.

2.5 Consideraciones parciales

El entorno de trabajo y las características que posee el IDE NetBeans 8.0.2 permiten el desarrollo del diseño de la GUI de la aplicación de manera sencilla y relativamente rápido.

En cara a implementar la comunicación de la interfaz diseñada se utilizó la arquitectura MVC en la estructura de los módulos del sistema por aportar claridad y facilidades para el mantenimiento del código.

La interfaz de la aplicación se diseñó para dos tipos de usuarios con diferentes roles. Se encuentra conformada por dos ventanas principales y un conjunto de ventanas auxiliares utilizadas para la comunicación con el usuario; díganse mensajes de error, avisos, preguntas y cuadros de diálogo que facilitan la interacción con el sistema.

CAPÍTULO 3. PRUEBAS Y ANÁLISIS DE LOS RESULTADOS

Se hace una descripción de cómo el usuario debe trabajar con el sistema implementado, los requisitos del mismos, tanto de software como de hardware. También aparece resumido un manual de usuario para disminuir la curva de aprendizaje de la aplicación.

3.1 Requerimientos del sistema

Para el correcto funcionamiento del software se necesitan un mínimo de requerimientos técnicos tanto de hardware como de software, siendo un software desarrollado en Java que ofrece la ventaja de ser multiplataforma.

Requerimientos de hardware:

- Al menos 64 MB de memoria RAM.
- La instalación básica necesita de 20 MB de espacio disponible en disco más 150 MB para la instalación del JAVA©-RunTimeEnviroment (JRE) Versión 8 update 40 si no está instalado, aunque pudiera requerir más dependiendo del tamaño que pueda tener las bases de conocimiento.
- Computador Pentium de 266MHz o superior.

Requerimientos de software:

- Sistema Operativo Windows 2000, XP, Vista (x86, x64), Windows 7 (x32, x64), Windows 8 y Windows 10.
- Sistema Operativo Linux que tenga instalado alguno de los siguientes administradores gráficos de ventanas para X: Common Desktop Environment (CDE), GNOME, The K Desktop Environment, Xfce Desktop Environment.
- Máquina virtual de Java (JRE) en su versión 8 update 40 o mayor.

- CTExperto.jar: permite el intercambio de información entre el usuario y el SE implementado en UCShell 3.0.
- Es necesario las bases de conocimiento que representan cada síntoma. En el momento actual se cuentan con tres:
    - causa_falla1.kbo (sobrecalentamiento)
    - causa_falla2.kbo (vibraciones)
    - causa_falla3.kbo (esfuerzo excesivo)

3.2 Validación de la interfaz de la aplicación

Para evaluar la aceptación por parte de los usuarios de la aplicación y el efecto sobre el diagnóstico de fallas que conlleva el uso de la misma se ha realizado un estudio estadístico sobre varios de los operarios de la termoeléctrica Carlos M. de Céspedes, considerando un grupo de usuarios. En este estudio se han empleado tanto cuestionarios sobre aspectos de aceptación y uso de la aplicación como la propia opinión de algunos especialistas de las cuestiones técnicas relativas a la misma.

Para la utilización de la aplicación "*CTExperto*" primeramente se ofrece una explicación breve de manera introductoria sobre cómo se trabaja y para que se utiliza la misma.

En el siguiente punto se describen los resultados más relevantes del estudio realizado.

Test de Validación

En concreto, el sistema fue ofrecido como una nueva potencialidad de diagnóstico de fallas en bombas centrífugas de alimentación de agua.

El test de validación estudiado se centró en los siguientes aspectos:

- Facilidad de uso.
- Tasa de errores de los usuarios inducidos por un mal entendimiento del funcionamiento de la interfaz.
- Capacidad de aprendizaje por parte del usuario.
- Satisfacción subjetiva del usuario.

La metodología empleada en el test fue la realización de diagnósticos por parte de cinco trabajadores de la Termoeléctrica "Carlos M. de Céspedes", a los que se les solicitó

diagnosticar posibles síntomas de las bombas. Finalmente, se realizó una corta entrevista (***Anexo II. Entrevista*****)** a cada uno de los usuarios, por separado, con el fin de medir su satisfacción más allá del éxito de sus diagnósticos.

En general los resultados del test fueron satisfactorios. Los beneficios expresados por parte de los usuarios se centran en decir que el sistema ofrece "la ventaja de ser un producto informático simple y fácil de usar". Es decir, mayor rapidez en el proceso de diagnóstico una vez se comprende el funcionamiento de la interfaz. En contra se señalaron alguna dificultades como la aparición, en ocasiones, de un pequeño tiempo (1 o 2 segundos) de espera, entre la navegación de una ventana a otra. También se observó que el sistema no siempre tiene éxito en el diagnóstico, por lo que es necesaria una actualización de las bases de conocimientos.

Como se ha dicho anteriormente el test se vio acompañado por una encuesta (***Anexo II. Encuesta***) con el objetivo principal de preguntar a los usuarios sus impresiones sobre aspectos de la interfaz.

Con los objetivos bien marcados, la encuesta evalúa algunos aspectos de la interfaz y utilidad de la aplicación en general mediante una escala de "*Likert*" dividida en cinco puntos: "Mucho", "Bastante", "Algo", "Poco" y "Nada". Los elementos así medidos son:

- Aspectos técnicos y estéticos relativos a la calidad del entorno visual.(Preguntas 1,2,3, 4 y 7)
- Aspectos funcionales y utilidad relativos a la eficacia y facilidad de uso.(Preguntas 5,6)

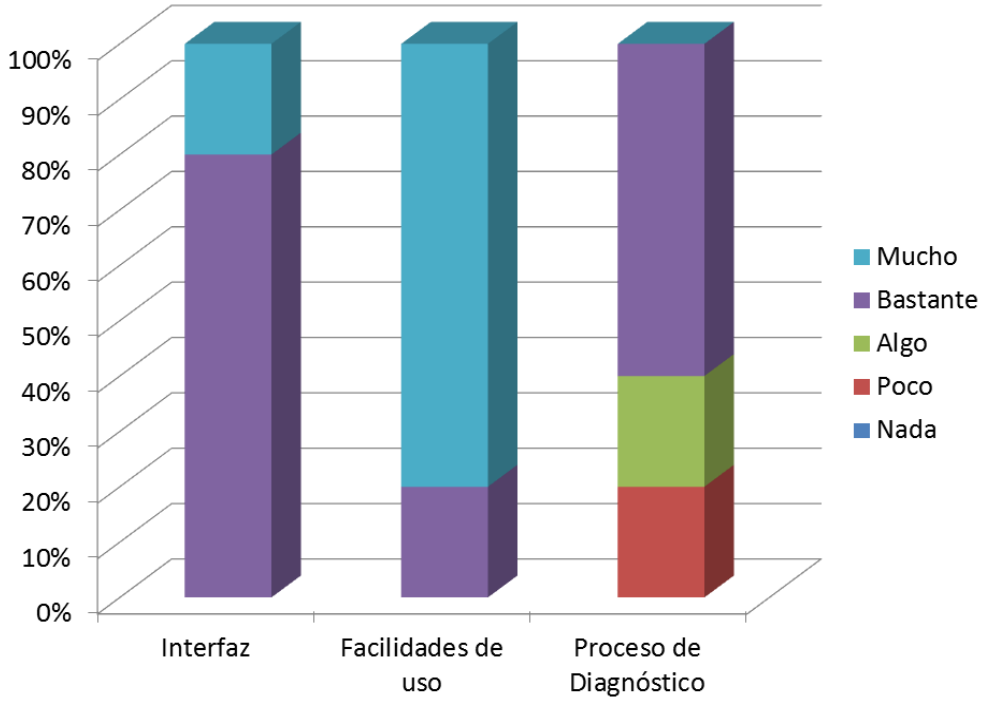

Figura 3.7: Valoración que los usuarios hacen de "*CETxperto*".

La figura 3.7 refleja el grado de satisfacción de los usuarios con la aplicación "*CTExperto*". La facilidad de uso ha sido bien valorada (el 100% de los usuarios creen que es muy fácil de usar). Hay que mencionar que en las tres características evaluadas, más de un 80% de los usuarios las valora positivamente. La característica peor valorada es el proceso de diagnóstico.

En cuanto a la utilidad de la aplicación, la figura 3.8 muestra como un 80% de los usuarios concuerdan con que, la misma, resulta útil.

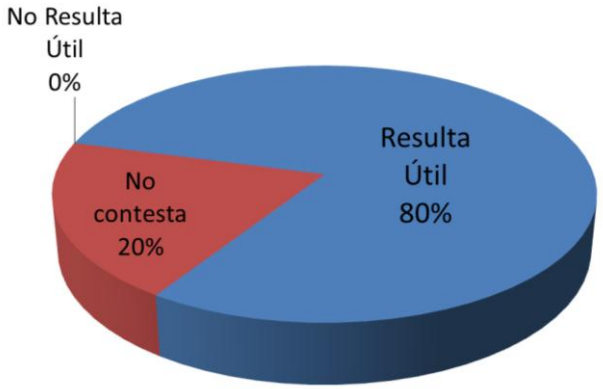

Figura 3.8: Opinión de los usuarios sobre la utilidad de "*CTExperto*".

3.3 Manual de usuario

La aplicación "*CTExperto*" como se ha mencionado anteriormente cuenta con dos ventanas principales, la ventana inicial "CTExperto" Figura 3.1 y la ventana de posibles síntomas.

Figura 3.1: Ventana "*CTExperto*".

En la primera se le da la bienvenida al usuario y a través de ella se accede a iniciar el diagnóstico en la segunda ventana "*Síntomas*". En ella el usuario puede seleccionar el síntoma que presenta la bomba; y cuando se elige el síntoma comienza el proceso de diagnóstico.

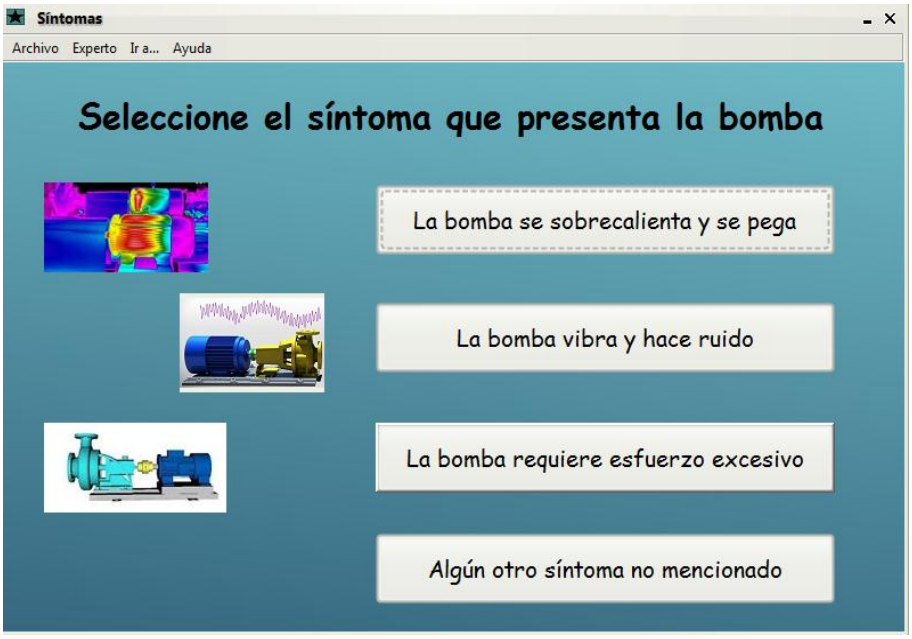

Figura 3.2: Ventana "*Síntomas*".

Como se puede apreciar en la figura 3.2 se muestra en la parte superior una barra de menús, al igual que en la ventana inicial; en la parte interior se observan botones con los posibles síntomas e imágenes asociada a cada uno.

Menús de la ventana "Síntomas":

- **Menú "Archivo" Figura 3.3,** contienen el menú "*Errores en la inferencia*" donde el usuario observa si ocurre algún error en la inferencia.

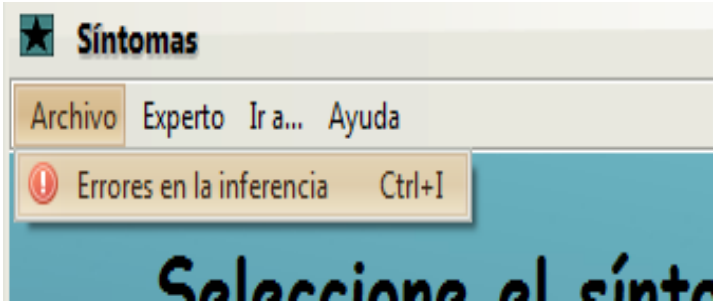

Figura 3.3: Menú "Archivo".

- **Menu "Experto"** Figura 3.4. Es donde el experto realiza las operaciones que le conceden a la BC.

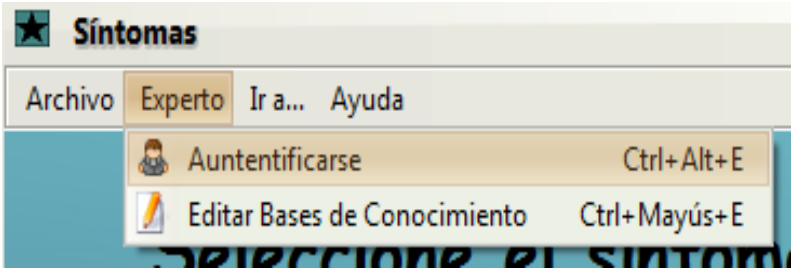

Figura 3.4: Menú "Experto".

- **Menú "Autentificarse"**: Abre una ventana donde le permite al usuario cambiar de rol y pasar de usuario a experto.

- **Menú "Editar Bases de Conocimiento"**: Abre el UCShell 3.0, permitiéndole al usuario experto realizar cualquier modificación en la BC.

➢ **Menu "Ir a…": Figura 3.5** Le permite al usuario navegar de una ventana a otra o salir de la aplicación si lo desea

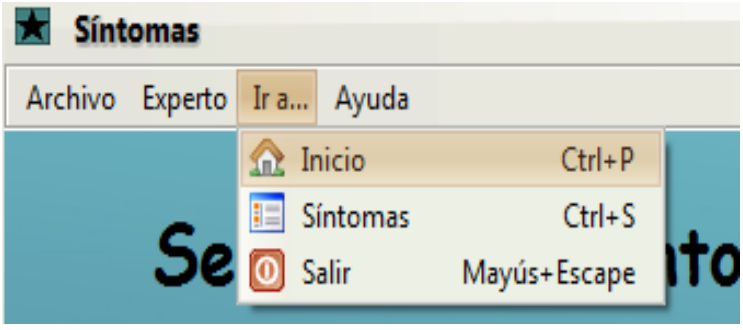

Figura 3.5**:** Menú Ir a…

➢ **Menu "Ayuda": Figura 3.6** Ofrece una ayuda simple para el uso de la aplicación. Cuenta también, con un manual para el mantenimiento de bombas centrífugas, donde puede encontrar actividades para el aprendizaje.

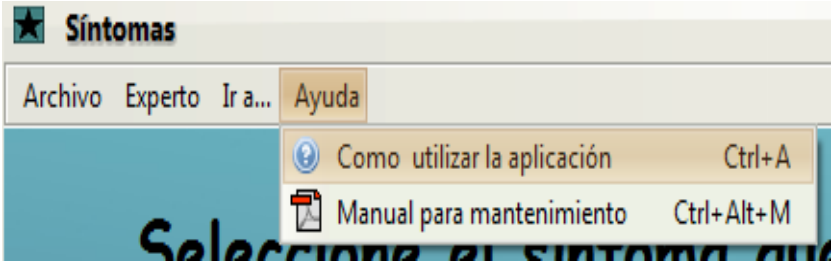

Figura 3.6: Menú Ayuda

3.4 Análisis económico

Debido a que la relevancia del presente trabajo radica en que posibilita, de forma sencilla y ergonómica, realizar el proceso de diagnóstico de las bombas; el análisis económico se realizará teniendo en cuenta los costos por concepto de adquisición del software y el hardware que en ella se utilizaron.

Teniendo en cuenta lo anteriormente dicho, los costos del trabajo están dados por la compra de los medios para su desarrollo. El software para su implementación se obtiene en Internet (NetBeans, 2015)gratuitamente, ya que son de código abierto. El costo de la obtención de una PC, monitor, mouse óptico, teclado, UPS, es en total aproximadamente de 648 cuc (CIMEX, 2014). En otro orden se puede analizar que el salario de un profesor universitario es de 675.00 cup, por lo que si mensualmente consume el 30 % de su trabajo en el mes en función del proyecto estaríamos hablado por concepto de salario de 236.00 cup (9.44 cuc) que totalizado en los 4 meses de trabajo seria 944.00 cup (37.8 cuc). Siendo el costo total de desarrollo de la aplicación aproximadamente de 1181(47.25 cuc).

3.5 Consideraciones parciales

La aplicación “CTExperto” es posible usarla, prácticamente en cualquier sistema operativo, una vez se haya instalado la Máquina virtual de Java (JRE) en su versión 8 update 40 o mayor.

Tras haber realizado la validación de la aplicación se puede afirmar que se cumplió con los principios básicos de diseño de las GUIs, debido a que, de manera general, más del 80 % de los usuarios la valoraron de forma positiva.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Como resultado final de este trabajo, se diseñó e implementó una GUI para el SE "*CTExperto*", con el fin de facilitar el diagnóstico de fallas en bombas de agua de alimentar calderas. A partir de estos resultados, se plantean las conclusiones generales siguientes:

1. La revisión bibliográfica de la literatura especializada arrojó como resultado la importancia de dominar los principios básicos de diseño de GUIs; así como reconocer los principales tipos de aproximaciones para la construcción de interfaces de usuario. De manera especial, en el mundo de las aplicaciones comerciales, se destacan las *herramientas basadas en modelos* y las *RAD: entornos de desarrollo rápido.*

2. La herramienta seleccionada para el desarrollo de la GUI, permite el desarrollo del diseño e implantación de la comunicación, de la misma, con el prototipo del SE desarrollado en UCShell 3.0.

3. La GUI obtenida ha demostrado ser capaz de mejorar notablemente la comunicación entre el sistema y el usuario, facilitando de esta manera el proceso de diagnóstico de fallas en las bombas de alimentación de agua.

4. La validez de la propuesta se confirma mediante la valoración de los usuarios de la aplicación, con lo cual se corrobora que el software permite, de forma sencilla y ergonómica, realizar el proceso de diagnóstico utilizando el sistema independiente al UCShell 3.0 donde fue desarrollado el prototipo del SE.

Recomendaciones

1. Utilizar la aplicación como parte de la metodología de inspección, que utilizan los expertos, para desarrollar este proceso, como elemento que concluya la validación del mismo.
2. Para aumentar las potencialidades de "*CTExperto*" se propone la actualización de las bases de conocimiento, así como incorporar nuevos posibles síntomas que causan fallas en las bombas.

REFERENCIAS BIBLIOGRÁFICAS

ALVAREZ, A. & VALERIO, E. 2009. Generadores de Interfaces de Usuario: QT Designer, NetBeans y Windows Forms Designer.

APPLE COMPUTER, I. 1987. *Human interface guidelines: the Apple desktop interface*, Addison-Wesley Longman.

BASS, L., CLEMENTS, P. & KAZMAN, R. 2003. Software architecture in practice, Addison-Wesley Professional.

BERRY, R. E. 1988. Common user access—a consistent and usable human-computer interface for the SAA environments. *IBM Systems Journal,* 27**,** 281-300.

BEVAN, N. 1995. Usability is quality of use. *Advances in Human Factors/Ergonomics,* 20**,** 349-354.

CARLOS SOTO, M. 2002. *Sistema experto de diagnóstico médico del sindrome de Guillian Barré* Universidad Nacional Mayor de San Marcos. Facultad de Ciencias Matemáticas.

CASTILLO, J. J. & LÓPEZ, J. V. 1998. *Ergonomía: conceptos y métodos*, Editorial Complutense.

CATALDI, Z. 2000. Una metodología para el diseño, desarrollo y evaluación de software educativo. Facultad de Informática.

CIMEX. 2014. *Listado de Precios* [Online]. Available: http://tecun.cimex.com.cu ed.

CRUZ OCAMPO, R. J. 2012. Integración de técnicas de ingeniería inversa en el desarrollo de interfaces de usuario dirigido por modelos.

DESIGN, I. O.-O. I. 1992. IBM Common User Access Guidelines. *Que, Carmel, Ind*.

EGAS CLAVIJO, P. G. 2015. Primefaces crud generador para Netbeans.

ESPINOSA, I. G. 2013. *Sistema inteligente para el diagnóstico de defectos en calderas de vapor acuotubulares.*, Universidad "Marta Abreu" de las Villas.

FREEMAN, E. 2004. Head first design patterns. " O'Reilly Media, Inc.".

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. 1993. Design patterns: Abstraction and reuse of object-oriented design. *Springer*.

GARLAN, D. & SHAW, M. 1993. An introduction to software architecture. Advances in software engineering and knowledge engineering. 1, 33.

GRANOLLERS, T. & LORÉS, J. Esfuerzo de Usabilidad: un nuevo concepto para medir la usabilidad de un sistema interactivo basada en el Diseño Centrado en el Usuario. V Congreso Interacción Persona Ordenador, 2004. 3-7.

KENDALL, K. E. & KENDALL, J. E. 2005. Análisis y diseño de sistemas. Pearson educacion.

LI, Y. & CUI, D. 2005. Improvement and Application of MVC Design Patterns**.**, 9, 035.

LOOK, J. 1999. Feel Design Guidelines. *Inc. Sun Microsystems*.

LÓPEZ, E. 2009. Interfaz Genérica para Sistemas de Adquisición basada en Software

Libre. UCLV.

LUISA, M., LUÍS, J., VISITACIÓN, M. & RAMÓN, J. 2010. Diseño de interfaces de usuario para aplicaciones colaborativas a partir de modelos independientes de la computación.

MANSO RODRÍGUEZ, R. A. 2007. La coautoría en la revista ACIMED en el período 2005-2006: un análisis mediante interfaces gráficas. *Acimed*, 15**,** 0-0.

MICROSYSTEMS, S. 1990. *OPEN LOOK graphical user interface application style guidelines*, Addison Wesley Publishing Company.

MITCHELL, J. 2003. An architectural pattern for adaptable middleware infrastructure. *In:* IEEE (ed.) *IEEE International Conference.*

MONTERO, F., LÓPEZ-JAQUERO, V., LOZANO, M. & GONZÁLEZ, P. 2005. De Platón al Desarrollo de Interfaces de Usuario.

MORENO, P. J. M. 2003a. Especificación de interfaz de usuario.

MORENO, P. J. M. 2003b. *Especificación del interfaz de usuario: de los requisitos a la generación automática.* Universitat Politècnica de València.

NETBEANS. 2015. *NetBeans IDE 8.0.2 Download* [Online]. Available: https://netbeans.org/downloads/8.0.2/.

PRESSMAN, R. S. & TROYA, J. M. 1988. *Ingeniería del software*, McGraw Hill.

ROBBINS, J. E., HILBERT, D. M. & REDMILES, D. F. Argo: A design environment for evolving software architectures. Proceedings of the 19th international conference on Software engineering, 1997. ACM, 600-601.

RODRÍGUEZ, N. H., ROMANO, J. M. G. & VALDERRAMA, J. T. 2004. Revisión de Lenguajes Declarativos para la Descripción de Interfaces de Usuario Independientes del Dispositivo. Interacción.

ROLSTON, D. W., GAMA, A. P. & ZISKIEND, I. T. 1990. *Principios de inteligencia artificial y sistemas expertos*, McGraw-Hill.

SANTO ORCERO, D. 2001. RAD con QT Designer. *Linux Actual: la primera revista en castellano del sistema operativo Gnu/Linux,* 3**,** 32-34.

SIERRA, E. A., HOSSIAN, A. & GARCÍA-MARTÍNEZ, R. 2007. Sistemas expertos que recomiendan estrategias de instrucción. Un modelo para su desarrollo. *Revista Latinoamericana de Tecnología Educativa-RELATEC,* 1**,** 33-47.

SOLANO, A. F., CHANCHÍ, G. E., COLLAZOS, C. A., ARCINIEGAS, J. L. & RUSU, C. A. Diseñando interfaces graficas usables de aplicaciones en entornos de televisión digital interactiva. Proceedings of the 10th Brazilian Symposium on on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction, 2011. Brazilian Computer Society, 366-375.

TAYLOR, E. 1959. An interim report on engineering design. . Massachusetts Institute of Technology.

VANDERDONCKT, J. & FARENC, C. 2000. *Tools for working with guidelines: annual meeting of the special interest group,* Springer Science & Business Media.

ANEXOS

Anexo I. Clase "*Inference*"

```
 /*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package inference;
importcompiler.Actions.Taccion;
importcompiler.Actions.TfindAction;
import compiler.*;
importedu.uci.ics.jung.graph.Forest;
importexception.OperationException;
import java.io.*;
importjava.util.Iterator;
importjava.util.LinkedList;
importjava.util.Stack;
importjavax.swing.JTextPane;

//import visualManager3.Window;

/**
*
* @author lissett
*/
public class Inference {
public static LinkedList<Tvar>externalVars;//Las variables que son externas al sistema
public static LinkedList<Trules> rules; //la listaenlazada de reglas
publicstaticLinkedList<Task>asks; //la lista enlazada de preguntables
public static LinkedList<Fact> facts = new LinkedList<Fact>(); //La listaenlazada de hechos
publicstaticStack<Fact>goals = new Stack<Fact>(); //esta estructura va guardando la traza por
la cual se infirió una variable
publicstaticLinkedList<Taccion>actions;//las acciones declaradas al principio
publicstaticSymTabtable; //la tabla de símbolos
publicstaticintmarkedRulesAmount;//la cantidad de reglas marcadas
publicstaticStringcurrProjectName; //La dir del proyecto actual
publicstaticQuestionMakerqm; //La interfaz que hace las preguntas, hay que implementarla
publicstaticDisplayerdisplayer;//El que muestra los resultados
```

```
public static class facts {
public facts() {
}
}
//ESte método es el que inicializa los valores de QuestionMaker y Displayer
//En caso de no hacerlo habrá errores durante la inferencia
public void initInference(QuestionMakeriQuestionMaker, Displayer iDisplayer) {
Inference.qm = iQuestionMaker;
Inference.displayer = iDisplayer;
}
public static void clearInference()
{
externalVars = new LinkedList<Tvar>();
rules = new LinkedList<Trules>();
asks = new LinkedList<Task>();
facts = new LinkedList<Fact>();

goals = new Stack<Fact>();
actions = new LinkedList<Taccion>();
}
//Este metodo desencadena la inferencia. Se le pasa el fichero donde esta la forma intera y
donde se van a guardar
//los mensajes que va enviando el mismo.
// Se le pasa ademas si esta debugueando y los paneles donde va a mostrar los mensajes de
debuggeo
publicboolean infer(String fileName,
String errorsFileName,
String standardOutInferenceFileName,
String currProjectName
) throws OperationException {
//JTextPaneoutputPane,
//JTextPanecurrentEditorPane
try {
File file = new File(fileName);
if (errorsFileName != null) {
System.setErr(new PrintStream(new File(errorsFileName)));//el segundo parámetro es el
nombre del fichero donde se escribirán los errores
}
if (standardOutInferenceFileName != null) {
System.setOut(new PrintStream(new File(standardOutInferenceFileName)));//el tercer
parámetro es el nombre del fichero en el cual se imprimirá la salida standard
}
Inference.currProjectName = currProjectName;
String absoluteFileName=file.getAbsolutePath();
Tpunto_partida t2;
SymTab t;
try (ObjectInputStream is = new ObjectInputStream(new FileInputStream(absoluteFileName)))
{
```

```
t2 = (Tpunto_partida) is.readObject();
t = (SymTab) is.readObject();
}
if (t2.getAsks_list() != null) {

Inference.asks = t2.getAsks_list().toLinkedList();
}
if (t2.getVar_list() != null) {
Inference.externalVars = t2.getVar_list().toLinkedList();
}
if (t2.getRules_list() != null) {
Inference.rules = t2.getRules_list().toLinkedList();
}
Inference.actions = t2.getEncabezamiento().getActions_list().toLinkedList();
Inference.table = t;
Inference.start();
FileReaderfR = new FileReader(new File(errorsFileName));//Leo del fichero de los errores
if (fR.read() == -1) {//Si está vacío el archivo
return true;
}
} catch (IOExceptionioE) {
System.err.println("Error de E/S");
//ioE.printStackTrace();
} catch (java.lang.Exception ex) {
System.out.println(ex.getMessage());
//ex.printStackTrace();
}
return false;
}
public static void start() throws OperationException {
for (Iterator<Taccion> it = actions.iterator(); it.hasNext();) {
Taccionaccion = it.next();
/*
if (isDebugging) {
Window.outputPane.writeOutputWord("Action: " + accion.getAccion() + "\n",
Color.MAGENTA);
Window.currentEditor.gotoLine(Integer.parseInt(accion.getLine()));
Window.suspendThread();

}*/
accion.execAction();
}/*
if (Window.isDebugMode) {
Window.currentEditor.gotoLine(-1);
}*/
}
public static String howTheValueWasReached(String varName) {
TfindAction finder = new TfindAction("FIND");
```

```
doublevarCode = inference.Inference.table.getCodByName(varName);
if (inference.Inference.facts.size() != 0) {
Fact hecho = inference.Inference.getFactByCode(varCode);
returnfinder.howIReachTheValue(hecho);
}
return "";
}
public Forest<String, Integer>getInferenceTree(String varName) {
TfindAction finder = new TfindAction("FIND");
doublevarCode = inference.Inference.table.getCodByName(varName);
if (inference.Inference.facts.size() != 0) {
Fact f = inference.Inference.getFactByCode(varCode);
returnfinder.getInferenceTree(f);
//return finder.howIReachTheValue(hecho);
}
return null;
}
public static Task findAskByCode(double code)
{
for (Iterator<Task> it = asks.iterator(); it.hasNext();) {
Task ask = it.next();
if (ask.getVar().getCodigo() == code) {
return ask;
}

}
return null;
}
public static void setFactAsVisited(Fact f) {
int i = 0;
for (Iterator<Fact> it = facts.iterator(); it.hasNext();) {
Fact fact = it.next();
if (fact.equals(f)) {//Si lo encontré
f.setVisited(true);//Lo marco como visitado
facts.set(i, f);//Lo inserto modificado en la posición en la que estaba
return;
}
i++;
}
}
public static TrulesgetRuleByNumber(double number) {
for (Iterator<Trules> it = rules.iterator(); it.hasNext();) {
TrulescurrRule = it.next();
if (currRule.getNum() == number) {
returncurrRule;
}
}
return null;
```

```
}
public static Fact getFactByCode(double varCode) {
for (Iterator<Fact> it = facts.iterator(); it.hasNext();) {
Fact fact = it.next();
if (fact.getVar() == varCode) {
returnfact;
}
}
returnnull;
} }.
```

Anexo II. Encuesta

Evaluación de la interfaz de la aplicación "CTExperto".

Estimado usuario:

Se desea realizar una evaluación a la interfaz del Sistema Experto "CTExperto" para lo cual solicitamos su colaboración en el llenado de la presente encuesta. Por favor, marque con una cruz en cada pregunta, aquella opción de respuesta que Ud. considere la más cercana a la realidad.

Sus respuestas serán confidenciales y servirán únicamente para mejorar la interfaz de la aplicación "CTExperto".

1.- ¿Consideras que el diseño de la interfaz: *estructura, organización*, etc., de la aplicación "CTExperto" son adecuados?

___Sí ___No

2.- ¿Crees que los elementos utilizados en la interfaz facilitan su utilización y aprendizaje?

____ Mucho ____ Bastante ____ Algo ____ Poco ____ Nada

3.- El tipo de letra utilizado, así como el tamaño de la misma, ¿crees que son los adecuados?

____ Mucho ____ Bastante ____ Algo ____ Poco ____ Nada

4.- ¿Crees que es adecuado el diseño de los siguientes recursos?

	Mucho	Bastante	Algo	Poco	Nada
Errores en la Inferencia					
Editar Bases de Conocimiento					
Ir a…					
Como utilizar la aplicación					
Manual para Mantenimiento					

5.- ¿Crees que el uso de la aplicación "CTExperto" puede ayudar a facilitar el proceso de diagnóstico de bombas centrífugas?

___Sí ___No

6.- ¿En qué medida consideras que la utilización de la aplicación "CTExperto" te ha facilitado el proceso de diagnóstico de bombas centrífugas?

____ Mucho ____ Bastante ____ Algo ____ Poco ____ Nada

7.- ¿Qué mejoras introducirías en la interfaz de la aplicación "CTExperto"?

_____

_____

_____

Muchas Gracias por su colaboración.

Anexo II. Entrevista

Prueba de Usabilidad

Entrevistador:

Fecha:

Aplicación:

Estimado Usuario:

Le agradezco su disposición de participar en esta "Prueba de Usabilidad" que nos ayudará a detectar problemas en la aplicación "CTExperto", si es que los tuviera. Vamos a comenzar con algunas preguntas que permitirán saber quién es Usted y cómo utiliza la aplicación.

Presentación del Usuario

1.- ¿Cuál es su nombre?

2.- ¿A qué se dedica [*Profesión, Actividad*]?

3.- ¿Qué experiencia tiene en el campo de la automática?

4.- ¿Qué experiencia tiene con el manejo de bombas centrífugas?

5.- Cuando desea encontrar el origen de una falla de una bomba centrífuga, ¿cómo realiza el diagnóstico?

6.- ¿Usa un Sistema Experto? ¿Cuál? Si no usa un Sistema Experto, ¿cómo lo hace?

Secciones de Preguntas

Le vamos a pedir que durante las acciones que desarrolle, vaya manifestando en voz alta lo que vaya pensando con el fin de que podamos entender qué le sugiere lo que va mostrando cada pantalla.

Las preguntas que le surjan durante el proceso de diagnóstico de las fallas sólo las podremos responder al final de la prueba. Una vez agradecemos su ayuda. Ahora, vamos a la aplicación que vamos a probar...

1.- ¿Le parece adecuada la selección de la imagen de presentación en la ventana principal?

2.- ¿Al ver la ventana principal, pudo distinguir de una sola mirada cuál era el contenido más relevante que se ofrecía la aplicación? ¿Cómo logró hacer esa distinción?

3.- ¿Le pareció adecuada la forma en que se muestran las imágenes e iconos en la interfaz de la aplicación? ¿Son adecuadas para representar el contenido del que trata cada opción?

4.- ¿Considera que gráficamente la interfaz está equilibrada, muy simple o sobrecargada?

5.- ¿Tras una primera mirada, le queda claro cuál es el objetivo de la aplicación? ¿Qué facilidades ofrece para el diagnóstico de fallas? ¿Las puede enumerar?

6.- ¿Cree que las facilidades para el diagnóstico de fallas que se ofrecen la aplicación son de utilidad para su caso personal?

7.- ¿Qué es lo que más te llamó la atención positivamente o negativamente de la utilidad que ofrece la aplicación?