Universidad de las Ciencias Informáticas

Facultad 6

Título: alasARBOGEN: aplicación informática para la representación de árboles genealógicos.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

Lien Le Sánchez

Reynaldo Rosado Roselló

Tutores:

Msc. Yanet Villanueva Armenteros

Ing. Alfonso Claro Arceo

Mayo, 2008

"El éxito consiste en obtener lo que se desea. La felicidad, en disfrutar lo que se obtiene".

Ralph Waldo Emerson

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

Lien Le Sánchez

_____

Firma del autor

Reynaldo Rosado Roselló

_____

Firma del autor

Msc. Yanet Villanueva Armenteros

_____

Firma del tutor

Ing. Alfonso Claro Arceo

_____

Firma del tutor

DATOS DE CONTACTO

Tutores:

Msc. Yanet Villanueva Armenteros
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: villanueva@uci.cu

Ing. Alfonso Claro Arceo
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: aclaro@uci.cu

AGRADECIMIENTOS

En primer lugar a la Revolución y al Comandante en Jefe Fidel Castro Ruz, creador de este magnífico proyecto de la UCI.

A nuestros padres, guía y inspiración en todo momento.

De forma especial a Alfonso y Liusmila, por dedicarnos tanto tiempo y su asesoría constante.

A Ramses por siempre estar presente cuando lo necesitamos.

A Yanet y Alieski por su asesoría y atención.

A Reynaldo Álvarez por estar siempre junto a nosotros buscando solución a cada problema.

No podemos dejar de mencionar a un grupo de personas que siempre que los molestamos nos supieron dedicar su tiempo con paciencia y dedicación ellos son, Adita, Tania, Aidacelys y Gilberto.

DEDICATORIA

Lien Le Sánchez

Dedico este trabajo a mis padres Clara Sánchez Martínez y Thang Le Manh, por ser mis ideales a seguir y mi fuente de inspiración.

A mi hermano y a Lissenny por su apoyo y ayuda.

A mi tía Idalmis Sánchez, a Omar y mis primos Maylien y Yusniel por apoyarme y quererme.

A Mirian Díaz y George Obregón por quererme como una hija.

A mis familiares en sentido general, por todo su apoyo.

A Pedro por ser el amor de mi vida, estar siempre a mi lado, comprenderme y apoyarme incondicionalmente.

A mis amigas de siempre, Sandra, Lissete, Yadira, Aliekna y Aida por ser tan especiales.

A mis amigos Ramses, Maikel, Reynaldo Álvarez y Yadir por todo su apoyo durante estos 5 años.

A mis amistades de la UCI y al secretariado de la FEU de laFacultad 6 por siempre estar ahí cuando los necesito.

Reynaldo Rosado Roselló

A mis padres, por educarme y guiarme siempre por el camino correcto, en especial a mi mamá a la cual le deseo mucha salud…

A Katia, el amor de mi vida, sin dudas lo mejor que ha podido pasar…

A mi hermana, a la cual le deseo muchos éxitos…

A mi otra familia en Puerto Padre, mis abuelos Reri y Enedina, mi tía Beti, mi sobrina Susan, a Rerito y a Jorge…

A todos en la casa de Betancourt por dejarme formar parte de su familia…Pedrin, Marina, las chiquillas Kuki y Yulaisky, Luci, Osvaldo…

A Patricia, mi otra mama y a Pedro, gracias por sus consejos y por ayudarme tanto…

A mis otros hermanitos, Evelyn y Pedri, los quiero mucho…

A Rey, por saber ser mi hermano, por estar siempre ahí, te agradezco todo lo que has hecho por mí…

A mis amigos de siempre, Eslavy, Pedro y Peña por estar siempre en las buenas y en las malas.

A la FEU, en ella aprendí las materias de la vida, las que no te enseñan en la escuela...

A mis compañeros de la FEU, Alfonso, Cesar, Yosbel, Yadainys, Javier, a los muchachos del secretariado de la facultad 6…

A todos mis profesores y compañeros, a la facultad 6…

RESUMEN

Con el surgimiento del Centro Nacional de Genética Médica (CNGM) al frente de la Red Nacional de Genética en Cuba, se desarrollan diversas investigaciones y estudios. Al iniciarse éstos estudios genéticos en nuestro país surge la necesidad de contar con una herramienta informática capaz de representar el árbol genealógico de un individuo, elemento indispensable para estos estudios.

Para darle solución a este problema se desarrolla en la Universidad de las Ciencias Informáticas de conjunto con en CNGM, alasARBOGEN, una aplicación informática para representar árboles genealógicos, primera de su tipo en el país. Actualmente la aplicación esta liberada por la Dirección de Calidad en la UCI e instalada en el CNGM y en los centros provinciales de genética en Cuba. La aplicación permite, insertar individuos y relacionarlos, exportar imágenes del árbol, de forma general o de una parte seleccionada, exportar reportes de texto con los principales datos de cada individuo, entre otras funcionalidades.

PALABRAS CLAVE

Árbol Genealógico, Genética, alasARBOGEN, CNGM.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Desde el triunfo de la Revolución Cubana se inicia un proceso de profundas transformaciones en Cuba. La Salud Pública se convierte en una de las principales prioridades para el gobierno cubano; se desarrolló un programa que permitía llevar la salud a toda la población.

Con el inicio de la Batalla de Ideas surge la necesidad de acelerar las investigaciones relacionadas con los estudios genéticos en Cuba, por tal motivo el Comandante en Jefe Fidel Castro Ruz, el 5 de agosto de 2003, inaugura el Centro Nacional de Genética Médica (CNGM). Con el objetivo de llevar un monitoreo más efectivo de los problemas de salud cuya causa fuese genética.

Junto al CNGM fue concebida una red nacional que incluye centros provinciales y municipales de genética médica en Cuba. Con el objetivo de llevar a cabo el programa nacional para el diagnóstico, manejo y prevención de enfermedades genéticas y defectos genéticos

Esta red de instituciones desarrolla acciones asistenciales, docentes y de investigación en el campo de los problemas de salud de carácter genético en la población cubana, con el propósito de mejorar la calidad de vida y elevar el bienestar en el país.

El CNGM, con 5 años de creado, ha alcanzado un extraordinario nivel científico, debido fundamentalmente a las investigaciones que ha llevado a cabo en el área de los problemas de salud relacionados con la genética.

El CNGM, desde su fundación se ha preocupado por la necesidad de conocer la población cubana, de lograrse un estudio mas completo se podría identificar los problemas de causa genética que se pudieran presentar y actuar en consecuencia. A pesar de todos los esfuerzos realizados, en la actualidad se han podido identificar las características de zonas muy específicas que han sido de interés en algún momento determinado, debido a lo difícil que es la recogida masiva de la información de la población y su procesamiento de forma manual.

Una parte del trabajo de los genetistas consiste en representar el árbol genealógico de cada paciente que visita las consultas de genética. La función del mismo es ayudar al análisis e interpretación de las afectaciones genéticas de una familia. Actualmente la mayoría de los genetistas de nuestro país representan los árboles genealógicos de forma manual. Esto trae consigo un grupo de problemáticas e inconformidades: cualquier modificación implicaría la reelaboración íntegra del árbol genealógico; problemas de límite de espacio para representar el árbol genealógico cuando se necesitan varias

generaciones; imposibilidad de almacenar toda la información necesaria relacionada con un individuo, se hace engorroso archivar en formato duro el árbol de cada paciente, se tiene en cuenta que un especialista atiende a cientos de familias distintas; frena todo el intercambio entre los especialistas de diferentes áreas y la capacidad de presentar investigaciones a nivel nacional e internacional que requieran árboles genealógicos.

Una de las soluciones encontradas por el CNGM fue la adquisición de un software nombrado Cyrillic para representar árboles genealógicos. Sin embargo dicho software es una herramienta de software propietario, el cual no cubre las prestaciones demandadas por los especialistas y el país para el estudio de las enfermedades genéticas. El mismo requiere del pago de licencias por parte de cada una de las instituciones donde se vaya a utilizar, que constituye un gasto considerable para el país, que no puede cubrir, además se encuentra en idioma inglés y no es adaptable a las especificidades de los estudios cubanos al no estar diseñado para nuestro sistema de salud.

De ahí que se propone como **problema científico:** ¿Cómo contribuir al estudio genético en Cuba con la representación de árboles genealógicos de los pacientes?

Se define como **objeto de estudio:** El proceso para la representación gráfica de los árboles genealógicos.

El **campo de acción**: El proceso para la representación gráfica de los árboles genealógicos para la Red Nacional de Genética Médica en Cuba.

Para dar solución al problema científico expuesto se define como **objetivo general** de la investigación, desarrollar una aplicación informática que represente gráficamente el árbol genealógico de un individuo.

Para dar cumplimiento al objetivo general se plantea como **objetivos específicos:**

- Identificar las funcionalidades que tendrá la aplicación informática para la representación de árboles genealógicos.
- Diseñar la aplicación informática.
- Implementar la aplicación informática.

En función de los objetivos específicos enunciados, se definen como tareas de la investigación:

- Investigación del estado del arte para la representación gráfica de árboles genealógicos.
- Investigación de las tendencias y tecnologías para desarrollar aplicaciones de tipo escritorio.
- Análisis y selección de las tecnologías para desarrollar la aplicación.

- Modelación del negocio y las actividades del flujo de trabajo de requerimientos.
- Realización de las actividades del flujo de trabajo de diseño.
- Realización de las actividades del flujo de trabajo de implementación.
- Validación de la aplicación.
- Confección de los manuales de usuario e instalación de la aplicación.
- Instalación y puesta en marcha de la aplicación.

Aporte práctico esperado del trabajo:

Aplicación informática capaz de representar árboles genealógicos que cumpla con las exigencias y funcionalidades que requieren los especialistas de genética médica en Cuba.

El presente trabajo está estructurado en 4 capítulos. A continuación se describe de manera resumida el contenido que se expone en cada capítulo.

Capítulo 1: Fundamentación Teórica: En este capítulo se aborda el estado actual en el desarrollo de las aplicaciones de construcción de árboles genealógicos a nivel mundial. También son abordados otros temas como, los árboles genealógicos en la genética, la relación que existe actualmente entre ellos y su importancia. Además, incluye un estudio de las herramientas, las metodologías y las tecnologías en las que se apoya el desarrollo de la aplicación en función de un análisis de las tendencias actuales. Se ofrece también una descripción de los aspectos a tener en cuenta para realizar un árbol genealógico.

Capítulo 2: Características del Sistema: Este capítulo está destinado a describir el objeto de estudio, así como de explicar todo lo referente al problema existente en el Centro Nacional de Genética Médica, el cual conllevó al desarrollo de este trabajo. Se presenta además la propuesta del sistema y se especifican los requerimientos funcionales y no funcionales. Se realiza la definición de los casos de uso y la descripción textual de los mismos.

Capítulo 3: Diseño del sistema: En este capítulo se traducen los requisitos a una especificación que describe como implementar el sistema, a través del diseño, enfocado a cómo el sistema cumple sus objetivos teniendo en cuenta los requisitos funcionales y no funcionales, se realizan los diagramas de clases y los diagramas de interacción más relevantes según los casos de uso definidos en el capítulo anterior. Se explica además la arquitectura utilizada y los principales patrones de diseño utilizados.

Capítulo 4: Implementación: A través de la realización de este capítulo se describe cómo fue implementada la aplicación en términos de componentes. Se describen algunos fragmentos de códigos no triviales en la clase VisorDeSimbolos y la funcionalidad de guardar una familia. También se propone

un patrón para estandarizar el formato de guardar los árboles genealógicos. Se presenta la validación de las pruebas de calidad desarrolladas por la Dirección de Calidad en la UCI. Además se hace un análisis de la solución obtenida.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

En este capítulo se aborda el estado actual en el desarrollo de las aplicaciones de construcción de árboles genealógicos a nivel mundial. También son abordados otros temas como, los árboles genealógicos en la genética, la relación que existe actualmente entre ellos y su importancia. Además, incluye un estudio de las herramientas, las metodologías y las tecnologías en las que se apoya el desarrollo de la aplicación en función de un análisis de las tendencias actuales. Se ofrece también una descripción de los aspectos a tener en cuenta para realizar un árbol genealógico.

1.1 LOS ÁRBOLES GENEALÓGICOS Y LA GENÉTICA

Tanto los historiadores familiares como los genealogistas, siempre han sabido la importancia de preservar el pasado con el fin de mejorar el futuro. De hecho, todos nuestros antepasados nos legaron una herencia con muchas riquezas, las cuales van desde los genes mismos hasta los valores morales, la cultura y los bienes materiales.

El árbol genealógico es una representación gráfica que expone los datos genealógicos de un individuo en una forma organizada y sistemática, sea en forma de árbol o tabla. Puede ser ascendiente, es decir que expone los antepasados o ancestros de un individuo o puede ser descendiente, es decir que expone todos los descendientes del individuo. [1]

Un genograma es un formato para dibujar un árbol genealógico que registra información sobre los miembros de una familia y sus relaciones sobre por lo menos tres generaciones [2] Mediante un genograma se realiza la representación gráfica del árbol genealógico, a través de ciertos símbolos y nombres se representan a las personas que integran el árbol y también se representa la relación que existe entre ellos. Los genogramas nos permiten visualizar una fuente rica de hipótesis acerca de cómo un problema clínico puede estar conectado con el contexto familiar, y la evolución del problema y del contexto con el tiempo.

Los genogramas muestran:

- la estructura de una familia,
- las relaciones entre los miembros de una familia.

La mayor parte de los genogramas incluye información básica sobre el número de familias, el número de niños de cada familia, el orden de nacimiento y muerte. Algunos genogramas también incluyen la información sobre desórdenes que ocurren en la familia como: alcoholismo, depresión, enfermedades, y alianzas.

Pero el valor principal del genograma es describir gráficamente la relación biológica y legal que existe entre los diferentes miembros de la familia.

En todos ellos la simbología a utilizar es la misma, por ejemplo, cada miembro de la familia es representado como un cuadrado, un círculo o un rombo (Fig. 1), en dependencia de su género.

Fig. 1 Género

La persona clave (o paciente identificado) alrededor de quien se construye el genograma, se identifica en algunos ocasiones con una línea doble, o sea, en dependencia del sexo un círculo o un cuadrado doble, y en otros casos con una flecha apuntando hacia el paciente identificado. [3]

En la actualidad toda la humanidad tiene mucho que ganar de la investigación que se está llevando a cabo en el campo de la genética humana. Por suerte para todos, esta investigación está derrumbando fronteras impensables hace tan solo unos pocos años. Puede decirse que se está produciendo una auténtica revolución en el conocimiento de la genética que trae consigo la prolongación y mejora de la calidad de vida de los afectados por enfermedades genéticas. Algunos de los recientes avances incluyen: acceso a más y mejores pruebas de diagnóstico prenatal; mejoras en tecnología que permite vivir a las personas con enfermedades graves y a los recién nacidos que presentan deformaciones genéticas, así como prolongar la expectativa de vida de las personas con enfermedades genéticas y finalmente, poder llevar a cabo estudios de poblaciones a gran escala para identificar a los portadores de genes dañinos.

Un buen historial clínico incluye información sobre el paciente, sus hermanos y hermanas, su(s) hijo(s) y sus antecedentes. En el caso de las mujeres, el doctor tendrá que preguntar por datos referentes a abortos, mortinatos, nacimientos prematuros, muerte de niño por enfermedad y muerte de algunos de sus hermanos.

Los datos de los individuos y toda la información vinculada con las relaciones existentes entre los familiares va a conformar el árbol genealógico de la familia, esta información posibilitará al especialista realizar la investigación de las enfermedades existentes y emitir un criterio basándose en lo que pudo obtener de la consulta con el paciente.

La elaboración de los árboles genealógicos para la investigación de enfermedades genéticas es de gran importancia. Mediante su uso se han analizado infinidad de casos de las más disímiles enfermedades lográndose obtener así el nivel de prevalencia de la misma, las causas de su traspaso, sus rasgos y características.

1.2 APLICACIÓN DE LOS ÁRBOLES GENEALÓGICOS EN ESTUDIOS GENÉTICOS

1.2.1 Estudios genéticos en hemofílicos y familiares

El laboratorio de Genética del Hospital Sant Pau es centro de referencia para el estudio molecular de la hemofilia A y B en Barcelona. El servicio de inmunología en su sección de Biología Molecular, colabora con el servicio de hematología, prestando su apoyo técnico para el encauzamiento de la realización del estudio genético de enfermos, portadores y familiares de hemofilia A y B, mediante la confección de los árboles genealógicos familiares y la definición de los casos de portadores y familiares a estudiar en cada caso. El servicio de inmunología envía las muestras para su estudio al servicio de genética del Hospital Sant Pau y funciona como interlocutor en todo el proceso de diagnóstico. Desde que existe esta colaboración, aproximadamente desde el año 1997, se han estudiado varios casos de hemofilia tanto A como B. En las familias de hemofilia A, el estudio ha sido informativo y se han realizado diagnóstico de portadora en el embarazo y diagnóstico prenatal; y en las familias de hemofilia B, el estudio ha sido también informativo y se ha realizado solamente diagnóstico prenatal.[4]

1.2.2 Genealogía y estudios del ADN

La genética es considerada por muchos como una formidable nueva herramienta para la genealogía. La genealogía, por medio del ADN, se concentra en analizar secciones inactivas del resto del ADN, algunas veces llamada "ADN Basura".

Los servicios de la genealogía pueden reportar a la medicina y a la biología los avances obtenidos recientemente por el descubrimiento del ADN Mitocondrial (ADNmt). Este ha sido uno de los hallazgos más relevantes sucedidos en la historia reciente de la genética. Mediante su estudio, los científicos han sido capaces de "rastrear" el origen de los seres humanos hasta sus primeros ancestros, viajando millones de años en el pasado. [5]

Podemos apreciar que el uso de los árboles genealógicos vinculados a la genética ofrecen resultados muy satisfactorios, y permiten prevenir enfermedades que en algún momento constituyeron un peligro para la humanidad.

1.3 SISTEMAS PARA ELTRABAJO CON ÁRBOLES GENEALÓGICOS

Fueron analizados como parte de la labor investigativa, algunos de los software que representan árboles genealógicos existentes en la actualidad. La gran mayoría de ellos están orientados a trabajar en base a la construcción del árbol familiar, sin llegar a profundizar en el seguimiento de las enfermedades de un individuo; pues son software que se difunden en Internet y están a disposición de los que navegan a través de ella. Estos programas centran su interés en elaborar un árbol genealógico de la familia, añadiéndole fotos y el resto de los datos que la aplicación permita. Existe una gran diversidad de aplicaciones de este tipo. A continuación mencionamos algunos ejemplos.

1.3.1 GenoPro

El GenoPro es una aplicación que permite construir árboles genealógicos y genogramas. Es uno de los software para dibujar árboles genealógicos que existido en el mundo y ha tenido gran aceptación: permite construir el árbol familiar y le encuentra al individuo su pasado, presente y futuro. Se dice que dentro de los existentes para este tipo de trabajo, es uno de los más fáciles de aprender y de manipular.

GenoPro es capaz de dibujar la familia sin importar la complicación y una vez logrado esto, permite salvar en un Metafile, que puede ser insertado en Word, Power Point u otra aplicación de Windows. Brinda la posibilidad de añadir una ilimitada cantidad de imágenes para cada individuo y familia, y ayuda a crear un CD con el árbol familiar.

Es una herramienta que te posibilita generar detallados reportes, analizar y compilar estadísticas detalladas acerca de tus ancestros; cuenta también con un poderoso generador de reportes en diferentes idiomas, dicho reporte puede ser impreso.

Se ha conocido que la aplicación es utilizada en la actualidad por terapeutas, psicólogos, trabajadores sociales y agencias de protección del niño para atender relaciones de padres y familia; los hospitales y los profesionales médicos suelen usarlo para archivar y mantener un seguimiento de las enfermedades de la familia.

Este programa utiliza el genograma para ayudar a representar un árbol familiar, pues este software se basa en la realización del árbol genealógico de una familia para una enfermedad determinada, sin dejar de mencionar otras que puedan existir para individuos en particular y no en común para todos los integrantes de la familia en cuestión.

Para poder utilizar este software hay que pagar su licencia porque es propietario.

1.3.2 BitGen II

BitGen II es otro de los productos que aparecen en el mercado. Es una aplicación en español diseñada con el objetivo de facilitar la tarea de la elaboración de árboles genealógicos desde la computadora.

La interfaz que BitGen II presenta para la elaboración de un árbol genealógico consta de 4 pergaminos bordeados por reglas para medir adecuadamente la posición de cada elemento gráfico. En esta interfaz se encuentra además, los controles de movimiento y la barra de herramientas desde donde podemos indicar todas las funciones que se quieran realizar. Permite además agregar una fotografía o documento que tengan relación con la persona. Algo que es digno de destacar, por ser muy positivo, es que cuenta con dos grandes bases de datos, incorpora un diccionario jerárquico con la explicación de los diferentes elementos que se pueden encontrar. [6]

Este producto que comentamos es la segunda versión de un software al que aún le falta dinamismo, rapidez y un funcionamiento más cercano al de otras aplicaciones destinadas a trabajar en entornos operativos Windows. Además de ser un software propietario.

1.3.3 GDS (Sistema General de Documentación Familiar)

Está basado en la estructura de dos apellidos. Actualmente se encuentra disponible en español, catalán e inglés. Sus características extendidas lo convierten en un sistema que va más allá de la pura genealogía, pues es un repositorio de toda la información familiar, sea de estructura familiar, como de almacenamiento de documentos, fotografías, voz, video y multimedia. [7]

Independientemente de las características que se pueden encontrar en otros programas de genealogía, vale destacar las siguientes funciones:

- Multilenguaje: es posible un cambio de idioma sin parar la ejecución del programa.
- Cálculo de la relación familiar entre personas, aún en relaciones lejanas, por uniones conyugales múltiples.
- Álbum de fotos.
- Álbum temático.
- Creación de documentos de texto, archivos de voz, o entrada de fotografías desde el escáner, desde cualquier directorio o desde una cámara digital.
- Aceptación de cualquier documento multimedia, que pasa a ser parte de la base de datos.
- Exportación a archivos de texto para ser explotados por otros programas.
- Creación de su propia Web con todo o parte del contenido de la base de datos.

Entre las principales desventajas de este software se encuentra el ser propietario y utilizar la estructura de dos apellidos.

1.3.4 GenealogíaMac

De modo general, las opciones de este software se dividen en tres áreas funcionales: Editar, Visualizar y Publicar.

Editar permite introducir nuevas personas y familias, establecer la pertenencia de hijos a las familias y especificar la relación entre los padres. Se incorpora una persona en la ficha familiar como hombre, mujer o hijo. De este modo, también es posible registrar relaciones homosexuales. Es posible asignar a una persona o a un hecho una foto, o cualquier otro archivo de imagen, video o audio.

Visualizar ofrece múltiples posibilidades para la representación gráfica de los datos recopilados. La tabla de antepasados, la tabla de descendientes y el genograma. GenealogíaMac permite configurar el gráfico según sus preferencias, con o sin imágenes. Las líneas de la vida sirven de orientación general y la estadística muestra las edades que alcanzaron las personas en el árbol genealógico calculando el promedio. En la lista se detallan todas las personas del árbol genealógico, ordenadas según el principio que convenga. El resumen contiene los datos de la ficha familiar actual. Naturalmente se pueden imprimir todas y cada una de las representaciones. Para alcanzar óptimos resultados, tendrá a su disposición diversas opciones de impresión.

Publicar puede servir tanto para realizar copias de seguridad como para distribuir el árbol genealógico entre los familiares. Además, permite exportar el árbol completo en formato HTML para su publicación en internet. Para este fin el programa está provisto de las herramientas para configurar la presentación deseada para el internet e incluir las imágenes correlacionadas. GenealogíaMac genera copias comprimidas de sus imágenes automáticamente para garantizar una visualización óptima en internet. [8]

Este software es propietario e incluye elementos innecesarios en los estudios genéticos cubanos.

1.3.5 Cyrillic

Cyrillic posee todas las características necesarias para dibujar árboles genealógicos. Es el más completo, funcional y fácil de usar. La aplicación está diseñada para el manejo de datos genéticos en el sistema operativo Windows. Todas las herramientas están dispuestas de forma que se pueda, rápida y fácilmente, manejar los datos de los que disponga. [9]

Cyrillic puede almacenar un elevado rango de datos para cada individuo, incluyendo información personal, clínica y genética. Entre las características que posee, las más importantes son:

- Los datos se actualizan automáticamente mientras se dibuja.
- Un manejo perfeccionado para casos de gemelos y embarazos múltiples.

Este software es privativo, permite relaciones genéticas entre individuos de un mismo sexo, esto genéticamente no tiene sentido. En ocasiones da errores que pueden reiniciar la computadora.

La automatización del proceso de construcción de los árboles genealógicos es algo necesario en la actualidad para el trabajo de los genetistas, se realizó un estudio de los productos existentes en el mundo que se dedican a este fin, con vistas a adaptar las principales funcionalidades a las necesidades de los genetista cubanos. Del estudio se llegó a la conclusión que ninguna de las herramientas existentes soluciona el problema de los genetistas cubanos, puesto que, muchos de estos son software propietario y no se adaptan a las especificaciones genéticas del país

1.4 APLICACIONES DE ESCRITORIO

Cuando se programa una aplicación de tipo escritorio se tiene un juego de controles de interfaz de usuario mucho más rico que con HTML, también se tiene un control exacto del posicionamiento de los elementos de la pantalla. Si necesita integración con algún hardware especial, tal como en los sistemas de Punto de Venta (POS), máquinas de control numérico, impresoras y scanners, es mucho

más fácil hacerlo desde una aplicación de escritorio que desde un navegador. Cuando es necesaria la integración con otros productos, es también mucho más fácil hacerlo desde una aplicación escritorio que desde un navegador. Muchas veces es necesario intercambiar datos con algún procesador de texto. Hacer esto desde una aplicación de escritorio es más sencillo, pero no tanto desde una aplicación desarrollada para la web.

A partir de los elementos anteriores y la petición del cliente, se decide desarrollar una aplicación de escritorio.

1.5 METODOLOGÍA DE DESARROLLO

La metodología de desarrollo es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a realizar nuevo software [10]

Sintetizando lo anterior, una metodología representa el camino para desarrollar software de una manera sistemática. Las metodologías persiguen tres necesidades principales:

- Mejores aplicaciones, que conduzcan a una mejor calidad.
- Un proceso de desarrollo controlado.
- Un proceso normalizado en una organización, no dependiente del personal.

Los procesos se descomponen hasta el nivel de tareas o actividades elementales, donde cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo.

1.6.1 Extreme Programming (XP)

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código, XP promueve la adaptabilidad de los procesos de desarrollo basándose en los principios y prácticas que presenta. Quienes trabajan usando XP deben seguir procesos disciplinados, pero más que eso, deben combinar la disciplina con la adaptabilidad necesaria del proceso. Es una metodología de desarrollo de software que es utilizada para proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. [11]

Los objetivos de Programación extrema son:

- Lograr la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita.

- Potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y los desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

1.6.2 Rational Unified Proccess

RUP es un proceso de desarrollo de software en el que se asignan tareas y responsabilidades que tiene como objetivo asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles. [12]

El Proceso Unificado es un proceso de desarrollo de software que contiene un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software (Fig. 2)**.** Más que un simple proceso, el proceso de desarrollo de software es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, tipos de organizaciones, niveles de actitud y tamaños de proyecto. Está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Utiliza el Lenguaje Unificado de Modelado (Unified Modeling Languaje, UML) para preparar todos los esquemas de un sistema software. Garantiza la elaboración de todas las fases de un producto de software orientado a objetos. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos.

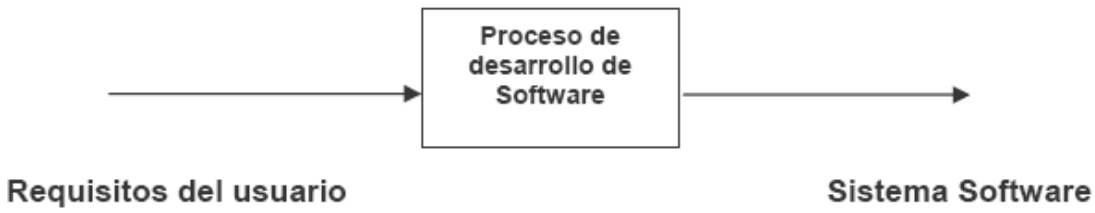


Fig. 2 Un proceso de desarrollo de software

Los verdaderos aspectos definitorios del Proceso Unificado, y que lo convierten en único, se resumen en tres frases claves: dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental.

- **Dirigido por los casos de uso**

Teniendo en cuenta que la razón de ser de un sistema es brindar servicios a los usuarios, RUP define caso de uso como el conjunto de acciones que debe realizar un sistema para dar un resultado de valor

a un determinado usuario, y los utiliza tanto para especificar los requisitos funcionales del sistema, como para guiar todos los demás pasos de su desarrollo, dígase diseño, implementación y prueba.

- **Centrado en la arquitectura**

La arquitectura es una vista del diseño completo con las características más importantes. Esta no solo incluye las necesidades de los usuarios, sino también otros aspectos técnicos como el hardware, el sistema operativo, el sistema de gestión de base de datos, los protocolos de red; con los que debe coexistir el sistema. En otras palabras, la arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.

- **Iterativo e incremental**

La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos. Cada uno de estos mini-proyectos se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. En cada iteración los desarrolladores seleccionan un grupo de casos de uso, los cuales se diseñan, implementan y prueban. La selección de lo que se implementará en una iteración se basa en casos de uso de mayor utilidad y los riesgos más importantes. Permite mantener la motivación del equipo pues puede ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos.

Para desarrollar la propuesta que presenta este trabajo se ha decidido utilizar como metodología el Proceso Unificado de Modelado (RUP) porque captura varias de las mejores prácticas en el desarrollo moderno de software en una forma que es aplicable para un amplio rango de proyectos y organizaciones. Es una guía de cómo utilizar de manera efectiva UML y le proporciona a cada miembro de un equipo fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las actividades críticas de desarrollo. Esta metodología (RUP) crea y mantiene modelos, en lugar de enfocarse en la producción de una gran cantidad de papeles de documentación. En el caso de XP, se consideran que son metodologías relativamente jóvenes, para proyectos cortos y simples.

1.6 ROLES Y ARTEFACTOS

Un rol define el comportamiento y responsabilidades de un individuo [13], es una definición abstracta de un conjunto de actividades realizadas y de artefactos obtenidos. Los roles son realizados típicamente por un individuo, o un conjunto de individuos, trabajando juntos en equipo. Un miembro del equipo de proyecto cumple normalmente muchos roles. Los roles no son individuos; sino que, describen cómo los individuos se comportan en el negocio y qué responsabilidades tienen.

La metodología Proceso Unificado de Desarrollo (RUP) define un conjunto de roles, pero de acuerdo con la necesidad se desarrollarán los roles de analista, diseñador y programador pertenecientes a los grupo de analistas y desarrolladores definido por dicha metodología.

Analista: Agrupa los roles que están involucrados fundamentalmente en la captura y gestión de los requisitos del sistema, que pueden estar representados por una o varias personas entre los que se encuentran: Analista de Procesos del Negocio, Diseñador del Negocio, Analista del Sistema y Especificador de Requerimientos.

Los artefactos a realizar desempeñando el rol de analista se describen a continuación:

- **Modelo de casos de uso del negocio:** Es un modelo que describe los procesos de un negocio y su interacción con elementos externos.
- **Realización de los casos de uso del negocio:** Describe cómo los trabajadores del negocio, entidades del negocio y los eventos del negocio interactúan en la realización de un caso del uso del negocio.
- **Modelo de casos de uso:** Es un modelo del sistema que contiene actores, casos de uso y sus relaciones.
- **Especificación de requerimientos del software:** Es la captura de los requerimientos del software para el sistema o una parte de éste.
- **Glosario:** Es un documento que define los términos comunes que se utilizan para describir el proyecto.

Diseñador: Es el responsable del diseño de una parte del sistema que incluye: las restricciones de los requisitos, la arquitectura y el proceso de desarrollo del proyecto.

Los artefactos a realizar desempeñando el rol de diseñador se describen a continuación:

- **Clase del diseño:** Es una descripción de un grupo de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semánticas.
- **Realización de caso de uso:** Describe cómo un caso de uso específico es realizado dentro del modelo de diseño en términos de colaboración de objetos.

Arquitecto de Software: Es el responsable de la arquitectura de software, que incluye las decisiones técnicas claves que restringen el diseño global y la implementación para el proyecto.

Los artefactos a realizar desempeñando el rol de arquitecto de software se describen a continuación:

- **Documento arquitectura de software (vista de casos de uso):** Representa los casos de uso significativos para la arquitectura, ya que describen alguna funcionalidad crítica que debe priorizarse durante el desarrollo del software.
- **Modelo de diseño:** Es un modelo de objeto que describe la realización de casos de uso, y sirve como una abstracción del modelo de implementación y el código fuente.
- **Modelo de implementación:** Representa la composición física de la implementación en términos de subsistemas de implementación, y elementos de implementación (directorios y archivos, incluyendo código fuente, datos y archivos ejecutables).

Implementador: Es el responsable del desarrollo y prueba de los componentes, en acuerdo con las normas adoptadas en el proyecto para la integración de subsistemas más grandes.

Los artefactos a realizar desempeñando el rol de implementador se describen a continuación:

- **Elemento de implementación:** Son los componentes físicos que forman una implementación, que incluyen archivos y directorios. Incluyen los archivos de código de software (origen, binario o ejecutable), los archivos de datos y los archivos de documentación, como los archivos de ayuda en línea.
- **Artefactos de instalación:** Se refiere al software y las instrucciones documentadas requeridas para instalar el producto.

1.7 LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es una notación para escribir programas, a través de los cuales se puede comunicar con el hardware y dar así las órdenes adecuadas para la realización de un determinado proceso. Un lenguaje esta definido por una gramática o conjunto de reglas que se aplican a un alfabeto constituido por el conjunto de símbolos utilizados. [14]

1.7.1 Visual Basic

Visual Basic es un lenguaje de programación desarrollado por Alan Cooper para Microsoft. El lenguaje de programación es un dialecto de BASIC, con importantes añadidos. Su primera versión fue presentada en 1991 con la intención de simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilitará la creación de interfaces gráficas.

Visual Basic constituye un entorno de desarrollo integrado (IDE) que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un depurador, un compilador, y un constructor de interfaz gráfica o GUI.

1.7.2 C++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, como extensión del lenguaje de programación C. Se puede decir que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (RTTI)

C++ está considerado por muchos como el lenguaje más potente, permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta mucho su aprendizaje.

1.7.3 Pascal

Pascal es un lenguaje de programación de alto nivel, desarrollado por Niklaus Wirth. Se convirtió en uno de los lenguajes extensamente usados en lo cursos de introducción a la programación, que fue

bien recibido como lenguaje de enseñanza para estudiantes universitarios. Es un lenguaje con técnicas de programación estructurada, dicha programación hace programadores disciplinados y muy legibles. [15]

1.7.4 Java

Java es un lenguaje de programación orientado a objetos. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros. Posee las estructuras mínimas de un lenguaje de programación tradicional, sin añadir ninguna estructura más. Es independiente de la plataforma. Se compila a un formato de código de byte que puede ser leído e interpretado por muchas plataformas, incluyendo Windows 95, Windows NT, Solaris 2.3, GNU\Linux y Mac OS.

1.7.5 C#

Fue creado por Anders Heljsberg quien creo además lenguajes y entornos como por ejemplo Delphi, Turbo Pascal y Visual J++. Es una combinación de los mejores elementos de otros lenguajes que han tenido una gran difusión como C++, el Java o el Visual Basic.

Se tuvo como idea fundamental combinar, por ejemplo, la potencia de lenguajes como C++, con la sencillez de lenguajes como Visual Basic y además, que la migración a este lenguaje por los programadores de C-C++-Java, fuese lo más inmediata posible.

C# es un leguaje bastante joven, diseñado por Microsoft específicamente para la plataforma.Net, sin embargo esto no significa que esté inmaduro, todo lo contrario, al mismo se han incorporado las mejores características de otros lenguajes, así como nuevas potencialidades.

Teniendo en cuenta todo lo antes mencionado, podemos decir que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes; por esta razón, se suele decir que C# es el lenguaje nativo de .NET y de hecho, gran parte de la librería de clases base de .NET ha sido escrita en este lenguaje.

Es un lenguaje diseñado para lograr una combinación idónea de simplicidad, expresividad y desempeño eficiente.

Cuenta con gestión automática de memoria, implementa una fuerte política de seguridad de tipos, brinda mecanismos como los índices y la instrucción foreach, que hacen más fácil e intuitivo el trabajo, elimina la herencia múltiple (ofrece el uso de interfaces) y facilita el trabajo con propiedades y eventos.

Se dice que su competir más cercano es Java, lenguaje con el que guarda un enorme parecido, tanto en la sintaxis como en las características. En este aspecto, es importante señalar que C# incorpora muchos elementos de los que Java carece (sistema de tipos homogéneos, tablas multidimensionales, operadores redefinibles, entre otros.), hay que mencionar que la velocidad de ejecución del código escrito en C# es ligeramente superior a su respectiva versión en Java y también el desarrollo con componentes visuales es superior a cualquier otro lenguaje.

Las características más significativas que presenta C# son:

- Sencillez: C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET
- Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrando que son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular.
- Orientación a objetos: Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código ha de definirse dentro de las definiciones de tipos de datos, lo que reduce problemas por conflictos de nombre y facilidad del código. C# soporta todas las características propias del paradigma de programación orientadas a objetos: encapsulación, herencia, polimorfismo.
- Eficiencia: En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como insegura y podrá usarse en ellas punteros de forma similar a como se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiente velocidad de procesamiento muy grandes.
- Compatibilidad: Para facilitar la migración de programadores, C# no solo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C#

fragmentos de código escrito en estos lenguajes, sino que además ofrece, la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos, nótese que la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que esta muchas veces esperan recibir o devolver punteros.

Se decide usar el lenguaje C# por las ventajas que el mismo proporciona, tales como, buen desarrollo en el trabajo con componentes visuales, sencillez y eficiencia, y se ha llegado a la conclusión de que siendo utilizado de forma adecuada, garantizará cumplir con los objetivos que se han planteado.

1.8 HERRAMIENTAS A UTILIZAR

1.8.1 Visual Studio .NET

Visual Studio.NET es un conjunto de aplicaciones completo para la creación tanto de aplicaciones de escritorio como de aplicaciones Web de empresa para trabajo en equipo. Aparte de generar aplicaciones de escritorio de alto rendimiento, se pueden utilizar las eficaces herramientas de desarrollo basado en componentes y otras tecnologías de Visual Studio para simplificar el diseño, desarrollo e implementación en equipo de soluciones para empresa.

La última versión en línea de IDEs, Visual Studio.NET soporta los nuevos lenguajes .NET: C#, Visual Basic.NET y Managed C++, además de C++. Visual Studio.NET puede utilizarse para construir aplicaciones dirigidas a Windows (utilizando Windows Forms), Web (usando ASP.NET y Servicios Web) y dispositivos portátiles (utilizando .NET Compact Framework).

El aspecto de Visual Studio.NET es casi idéntico a las versiones anteriores del IDE (Microsoft Visual Studio). Algunas excepciones destacables son: la interfaz más limpia y la mayor cohesión. También es más personalizable con ventanas informativas de estado que automáticamente se ocultan cuando no se usan. Todas las versiones de Visual Studio, también su predecesora Visual C++, incluyen un depurador integrado en el entorno de edición.

La característica más notable del IDE es su soporte de los nuevos lenguajes .NET. Los programas desarrollados en esos lenguajes no se compilan a código máquina ejecutable (como por ejemplo hace C++), sino que son compilados a algo llamado Common Intermediate Language (CIL). Cuando los programas ejecutan la aplicación CIL, esta es compilada en ese momento al código de máquina apropiado para la plataforma en la que se está ejecutando. Mediante este método, Microsoft espera

poder soportar varias implementaciones de sus sistemas operativos Windows. Los programas compilados a CIL pueden ejecutarse solo en plataformas que tengan una implementación de .NET Framework. Es posible ejecutar programas CIL en Linux o en Mac OS X utilizando algunas implementaciones .NET que no pertenecen a Microsoft, como Mono y DotGNU.

1.8.2 Visual Paradigm

Visual Paradigm es una herramienta que provee soporte para la generación de código, tiene integración con diversos IDE's como NetBeans (de Sun Microsystems), JDeveloper (de Oracle), Eclipse (de IBM), JBuilder (de Borland), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en JAVA, .NET, XML e Hibernate.

Tiene dentro de sus características que es portable y posee gran facilidad de uso. Utiliza UML como lenguaje de modelado y entre sus ventajas más relevantes encontramos que permite realizar ingeniería tanto directa como inversa, es una herramienta colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto; genera documentación del proyecto automáticamente en varios formatos tales como web o pdf, y permite el control de versiones. Igualmente se puede destacar su robustez, usabilidad y portabilidad.

Su diseño se centra en casos de uso y se enfoca al negocio que genera un software de mayor calidad. También tiene disponibilidad en múltiples plataformas y usa un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.

Rational Rose Enterprise es el producto más completo de la familia Rational Rose. Soporta la generación de código a partir de modelos en los diferentes lenguajes de programación y de los diversos sistemas operativos de Windows. También domina el mercado de herramientas para el análisis, modelado, diseño y construcción orientada a objetos. Tiene todas las características que los desarrolladores, los analistas, y los arquitectos exigen soporte UML incomparable.

Rational Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas.

Se decide utilizar Visual Paradigm porque se ha convertido una herramienta CASE profesional a tener en cuenta a la hora de pensar un proyecto importante que soporta el ciclo de vida completo y es multiplataforma.

1.9 ESTILO ARQUITECTÓNICO

Un estilo arquitectónico describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción.

Modelo-Vista-Controlador (MVC)

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información y presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario (muestra información del modelo de usuario).

Controlador: Controla el flujo entre la vista y el modelo (los datos). Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual. Entre las ventajas del estilo Modelo-Vista-Controlador se encuentran:

Soporte de múltiples vistas: Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.

Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o

requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

1.10 PATRONES DE DISEÑO

Los patrones de diseño son soluciones probadas y muy bien documentadas a problemas comunes, sirven como estructura, como soporte para el desarrollo de un sistema, no son la solución a cualquier problema, esto quiere decir que al usar patrones de diseño no se desarrolla un sistema de lo mejor, se debe adaptar y utilizar con cuidado y elegancia. Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. [16]

A continuación se recogen las características de los patrones de diseño que se utilizan en la investigación.

1.10.1 Patrones de comportamiento

Los patrones de comportamiento estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal.

1.10.1.1 Patrón STATE

El propósito de este patrón es permitir a un objeto modificar su comportamiento a medida que su estado interno va cambiando, da la impresión de que el objeto “cambia de clase”.

Ventajas

- Dado que el comportamiento específico de cada estado está autocontenido en cada estado concreto, existe una gran flexibilidad para añadir nuevos estados y transiciones, mediante la definición de nuevas subclases.
- Se hacen más explícitas las transiciones entre estados que sí todo estuviera concentrado en una sola clase (esto hace más inteligible el diseño), impidiendo además los estados internos inconsistentes (desde el punto de vista del contexto, los cambios de estado son ATÓMICOS).

1.10.2 Patrones de creación

Los patrones de creación abstraen la forma en la que se crean los objetos, permiten tratar las clases a crear de forma genérica, dejando para más tarde la decisión de qué clases crear o cómo crearlas.

1.10.2.1 Patrón SINGLETON

Es un patrón que garantiza que una clase sólo tenga una única instancia y proporciona un punto de acceso global a la misma.

Ventajas

- El acceso a la instancia única está más controlado.
- Se reduce el espacio de nombres (frente al uso de variables globales).
- Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).

1.10.3 Patrones de Asignación de Responsabilidades (GRASP)

GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

1.10.3.1 Patrón Experto

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. También distribuye el comportamiento entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase sencillas y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

1.10.3.2 Patrón Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. El patrón Creador indica que la clase incluyente del

contenedor o registro es idónea para asumir la responsabilidad de crear la instancia contenida o registrada.

1.10.3.3 Patrón Polimorfismo

El patrón Polimorfismo un importante patrón estratégico en el diseño orientado a objetos. Es un principio fundamental en que se fundan las estrategias globales, o planes de ataque, al diseño como organizar un sistema para que se encargue del trabajo. Un diseño basado en la asignación de responsabilidades mediante el polimorfismo puede ser extendido fácilmente para que realice nuevas variantes.

Conclusiones

Después de hacer un análisis del estado del arte de las metodologías y tecnologías necesarias y actuales en el mundo se decide utilizar la plataforma .NET haciendo uso de lenguaje de programación C#. También se ha definido RUP como la metodología a utilizar y a Visual Paradigm como herramienta CASE en que se realizará la documentación de la aplicación aprovechando las posibilidades que brinda UML. Se definieron también las herramientas a usar para el desarrollo de la aplicación.

CAPÍTULO 2
CARACTERÍSTICAS DEL SISTEMA

Este capítulo está destinado a describir el objeto de estudio, así como de explicar todo lo referente al problema existente en el Centro Nacional de Genética Médica, el cual conllevó al desarrollo de este trabajo. Se presenta además la propuesta del sistema y se especifican los requerimientos funcionales y no funcionales. Se realiza la definición de los casos de uso y la descripción textual de los mismos.

2.1 OBJETO DE ESTUDIO

2.1.1 Objetivos estratégicos de la organización

La Red Nacional de Genética Médica de Cuba está integrada por 184 centros ubicados en todos los municipios y provincias del país, que coordinados por el Centro Nacional de Genética Médica conducen el Programa Nacional para el diagnóstico, manejo y prevención de enfermedades genéticas y defectos congénitos. Esta red de instituciones desarrolla acciones asistenciales, docentes y de investigación en el campo de los problemas de salud de carácter genético en la población cubana, con el propósito de mejorar la calidad de la vida y elevar el bienestar de nuestro pueblo. (Centro Nacional de Genética Médica). [17]

Actualmente se brindan consultas en el CNGM, en los centros de genética provinciales y municipales, en este último es donde se atienden directamente a los pacientes, que según su cuadro clínico el genetista para profundizar su estudio genético decide realizarle su árbol genealógico.

2.1.2 Flujo actual de los procesos

El genetista después de analizar el cuadro clínico del paciente decide si para realizar un estudio genético del mismo es necesario representar su árbol genealógico. Si decide realizarle el árbol genealógico procede a preguntarle todos sus vínculos familiares, todas las relaciones de pareja que ha tenido y datos que puedan aportar al estudio. El genetista recoge toda esta información para realiza el árbol genealógico en hoja de papel y de forma manual.

2.1.3 Análisis crítico de la ejecución de los procesos

Actualmente el proceso de construcción de un árbol genealógico es muy tedioso porque al realizarse de forma manual en una hoja de papel, conlleva a limitaciones en cuanto al tamaño de los árboles genealógico que necesitan varias generaciones. Si el árbol genealógico requiere de modificación esto significaría representarlo nuevamente provocando perdida de tiempo. Debido a la cantidad de estudios realizados por un genetista se hace engorroso el almacenamiento y búsqueda de los árboles genealógicos de cada paciente.

2.2 OBJETO DE AUTOMATIZACIÓN

El sistema permitirá la representación del árbol genealógico de un paciente. Además recogerá los datos del paciente y de sus familiares, facilita la realización de operaciones de inserción y modificación de datos y de individuos, así como las relaciones entre cada uno de los individuos del árbol y las enfermedades que padecen.

Se utilizará el software Microsoft Word o el bloc de nota para exportar un reporte con los datos de todos los individuos representados en el árbol genealógico.

2.3 MODELO DE NEGOCIO

2.3.1 Actores del Negocio

Actor	Descripción
Paciente	El paciente es el que inicia todo el proceso presentándose a la consulta de un genetista.

2.3.2 Trabajadores del Negocio

Trabajador	Descripción
Genetista	Especialista en genética, es el único encargado de crear el árbol genealógico de los pacientes.

2.3.3 Diagrama de Casos de Uso del Negocio

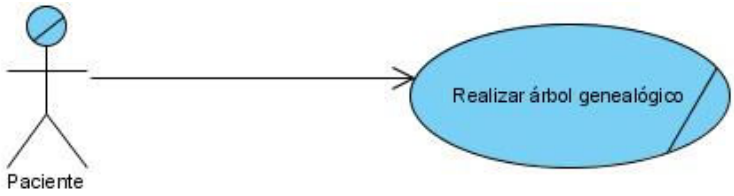


Fig. 3 Diagrama de Caso de Uso del Negocio

2.3.4 Descripción textual del Caso de Uso del Negocio

Caso de Uso:	Realizar árbol genealógico.
Actores:	Paciente(Inicia)
Trabajadores	Genetista
Resumen:	El caso de uso se inicia cuando el paciente se presenta en la consulta de un genetista. El genetista atiende el paciente y comienza a realizarle varios estudios, apoyándose en la realización de su árbol genealógico. Termina el CU.
Precondiciones:	Qué el paciente se presente a la consulta del genetista.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El paciente se presenta a la consulta.	2. El genetista le solicita sus datos personales.
3. El paciente informa los datos solicitados: • Nombre y apellidos • Edad • Sexo • Dirección	4. El genetista escribe los datos en su Registro Diario de Pacientes.
	5. El genetista localiza su historia clínica.
	6. El genetista le pregunta al paciente todos los vínculos familiares y todas las relaciones de pareja que ha tenido.
7. El paciente brinda su información y la de sus familiares: • Datos personales adicionales.	8. El genetista con la información que le va dando el paciente va representando el árbol genealógico en una hoja de papel.

• Vínculos familiares. • Relaciones de parejas. • Datos personales de los familiares. • Datos personales de pareja. • Enfermedades de la familia.	
	9. El genetista analiza el árbol genealógico ya terminado, para obtener un diagnóstico.
	10. El genetista escribe en la historia clínica del paciente el diagnóstico.
	11. El genetista informa el diagnóstico al paciente.
12. El paciente recibe su diagnóstico y se marcha.	
Flujo Alterno de Eventos	
Acción del Actor	**Respuesta del Negocio**
	5.1 El genetista no encuentra la historia clínica y le crea una nueva historia clínica al paciente.
Poscondiciones	Construir un árbol genealógico y obtener un diagnóstico a partir de él.
Mejoras	Informatizar: - Los datos personales y enfermedades de los individuos relacionados con el paciente - La representación del árbol genealógico del paciente.
Prioridad	Alta

2.3.5 Modelo de Objeto del Negocio

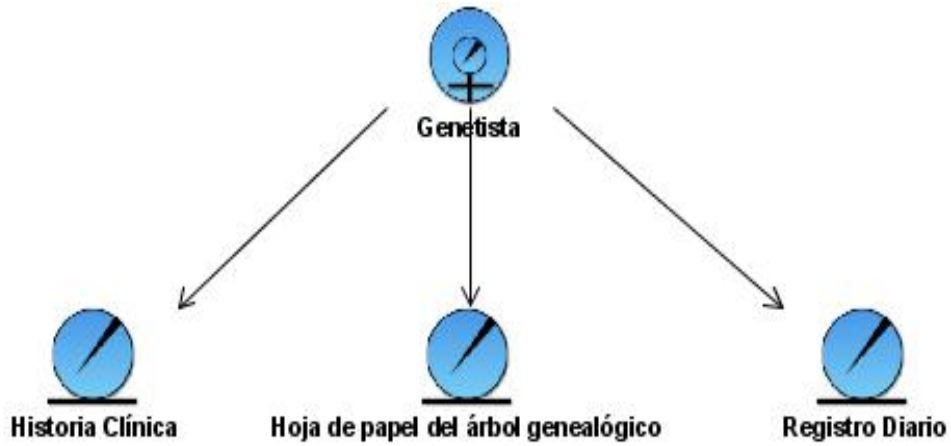


Fig. 4 Modelo de Objeto del Negocio

2.4 ESPECIFICACIÓN DE LOS REQUISITOS DE LA APLICACIÓN

2.4.1 Requerimientos Funcionales

R1. Gestionar gráficamente un individuo

R 1.1 Insertar un individuo.
R 1.2 Modificar datos de un individuo.
R 1.3 Eliminar un individuo.
R 1.4 Mostrar datos del individuo.
R 1.5 Trasladar gráficamente un individuo de posición.

R2. Gestionar relaciones entre individuos

R 2.1 Crear relación entre individuos.
R 2.2 Eliminar relación entre individuos.

R3. Gestionar símbolo

R 3.1 Crear nuevo símbolo.
R 3.2 Modificar símbolo.
R 3.3 Eliminar un símbolo.

R4. Gestionar enfermedad

R 4.1 Crear nueva enfermedad.
R 4.2 Eliminar enfermedad.

R5. Gestionar muestra

R 5.1 Crear nueva muestra.
R 5.2 Modificar muestra.
R 5.3 Eliminar muestra.

R6. Gestionar dimensiones del árbol

R 6.1 Aumentar tamaño de los individuos.
R 6.2 Disminuir tamaño de los individuos.

R7. Gestionar familia

R 7.1 Crear una nueva familia.
R 7.2 Buscar una familia.
R 7.3 Actualizar datos de una familia.
R 7.4 Guardar familia.
R 7.5 Exportar familia.

2.4.2 Requerimientos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener y que harán del producto un sistema confiable y seguro.

- **Apariencia o interfaz externa**

El acceso a la aplicación y su presentación será sencilla y fácil de manejar por el genetista. Tendrá una interfaz amigable y similar al ambiente de desarrollo del sistema operativo Windows. Poseerá el color verde fundamentalmente.

- **Usabilidad**

Para el uso óptimo de la aplicación es necesario tener conocimientos básicos de computación. Está previsto que sea utilizada por genetistas del Centro de Nacional de Genética Médica y todos los centros de genética del país.

- **Confiabilidad**

La aplicación tiene que ser altamente confiable pues a través de ella se manipularán datos confidenciales que serán utilizados en estudios e investigaciones, en dependencia del buen uso de esta información serán los resultados obtenidos.

- **Portabilidad**

El sistema debe permitir ser ejecutado sobre el Sistema Operativo Windows.

- **Software**

Se deberá disponer para el uso de la aplicación del Sistema Operativo Windows 98 o versiones superiores. También debe tenerse instalado el DotNetFramework versión 1.1 o superior.

- **Hardware**

Para el funcionamiento de la aplicación se requieren máquinas con los siguientes requisitos: Procesador Pentium 3 o superior, como mínimo 128 MB de memoria RAM, 300 MB de capacidad del disco duro necesario para la instalación y funcionamiento.

- **Eficiencia**

Para lograr la eficiencia del producto es recomendable una computadora como mínimo con 128 MB de RAM.

- **Restricciones de diseño e implementación**

Será una aplicación de escritorio, el desarrollo de la aplicación será basado en la metodología RUP que utiliza como lenguaje de modelado UML. Se utilizará como lenguaje de programación C#, y herramienta CASE Visual Paradigm para el modelado de los artefactos que se generan en cada uno de los flujos de trabajo definidos por RUP.

- **Requisitos para la documentación de usuarios en línea y ayuda del sistema**

La aplicación poseerá una ayuda y un manual de usuario donde se explicará su uso y se garantiza un buen desempeño de los usuarios a la hora de interactuar con la misma. Además tendrá un manual de instalación que guiará dicho proceso.

- **Rendimiento**

Será una aplicación de escritorio, lo que le confiere una mayor rapidez, no tiene ningún tipo de intercambio a través de la red. Para lograr un buen rendimiento de la aplicación es necesario que el tiempo de respuesta sea el menor posible.

2.5 DEFINICIÓN DE LOS CASOS DE USO DEL SISTEMA

2.5.1 Actores del Sistema

Actor	Descripción
Genetista	Especialista en genética que interactúa con el sistema, es el único encargado de crear el árbol genealógico de los pacientes.

2.5.2 Diagrama de Casos de Uso del Sistema

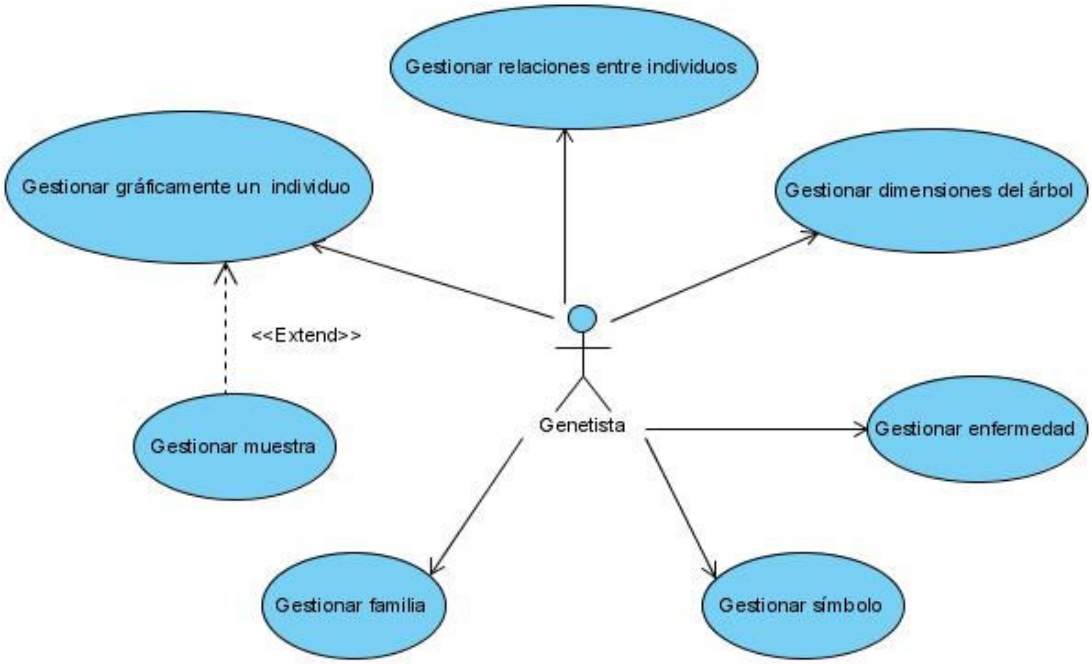


Fig. 5 Diagrama de Caso de Uso del Sistema

2.5.3 Listado de los Casos de Uso del Sistema

2.5.3.1 Gestionar gráficamente un individuo

Caso de Uso:	Gestionar gráficamente un individuo.
Actores:	Genetista
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión gráfica de un individuo: • Insertar un individuo. • Modificar datos de un individuo.

<table>
<tr><td></td><td>• Eliminar un individuo.
• Mostrar datos de un individuo.
• Trasladar gráficamente un individuo de posición.
El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.</td></tr>
<tr><td>Precondiciones:</td><td>Debe estar iniciada la aplicación. Para modificar, mostrar datos y eliminar un individuo debe ser creado.</td></tr>
<tr><td>Referencias</td><td>R1.1, R1.2, R1.3, R1.4, R1.5</td></tr>
<tr><td>Prioridad</td><td>Crítico</td></tr>
<tr><td colspan="2">Flujo Normal de Eventos</td></tr>
<tr><td>Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td>1. El genetista desea realizar una de las siguientes operaciones:
• Insertar un individuo.
• Actualizar datos de un individuo.
• Eliminar un individuo.
• Mostrar datos de un individuo.
• Trasladar gráficamente un individuo de posición.</td><td>2. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente:
• Si decide insertar un individuo, ir a Sección “Insertar un individuo”.
• Si decide actualizar los datos de un individuo, ir a la Sección “Actualizar datos de un individuo”.
• Si decide eliminar un individuo, ir a la Sección “Eliminar un individuo”.
• Si decide mostrar los datos de un individuo, ir a la Sección “Mostrar datos de un individuo”.
• Si decide trasladar de posición un individuo, ir a la Sección “Trasladar gráficamente un individuo de posición”.</td></tr>
<tr><td colspan="2">Sección “Insertar un individuo”</td></tr>
<tr><td>Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td>1. El genetista escoge que tipo de individuo desea insertar: femenino, masculino o desconocido.</td><td>2. El sistema muestra gráficamente el individuo según el tipo seleccionado por el genetista (femenino, masculino o desconocido) y da la posibilidad de registrar sus datos.</td></tr>
<tr><td></td><td>3. El sistema asigna en tiempo de ejecución:
• A la propiedad “Cantidad_de_hijos” se le asigna el valor de la cantidad de hijos representados en el árbol de ese individuo.
• A la propiedad “Cantidad_de_muestas” se le asigna el valor de la cantidad de muestras asociadas a ese individuo.
• A la propiedad “Cantidad_de_relaciones” se le asigna el valor de la cantidad de relaciones representados en el árbol de ese individuo.
• A la propiedad “Hijos” se le asigna el</td></tr>
</table>

	valor de la cantidad de hijos de cada sexo (sea femenino, masculino o desconocido) representados en el árbol de ese individuo. • A la propiedad "Edad" se le asigna el valor de la edad de ese individuo calculada mediante el campo "Fecha_de_nacimiento" y el campo "Fecha_de_muerte".
4. El genetista provee los datos para crear un nuevo individuo. • Muestras (Este parámetro puede tener varios valores) • Símbolo • Símbolo_adicional • Madre • Padre • Color_de_piel • Comentario • Dirección • Fecha_de_muerte • Fecha_de_nacimiento • Intento_de_suicidio. • Nombre • Primer_apellido • Segundo_apellido • Propósito En caso de un individuo femenino además indica si está embarazada.	5. El sistema en tiempo de ejecución si el genetista indica insertar una muestra ejecuta el caso de uso "Gestionar muestra".
	6. El sistema en tiempo de ejecución verifica que los datos estén correctos.
	7. El sistema crea el individuo con sus datos.
Flujos Alternos Sección "Insertar un individuo"	
Acción del Actor	**Respuesta del Sistema**
4.1. Si el genetista no desea proveer los datos del individuo se va a la acción 6 de la Sección "Insertar individuo".	7.1 En caso de que algún dato no esté correcto el sistema en tiempo de ejecución muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 1.1
Sección "Eliminar un individuo"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona en el árbol el individuo que desea eliminar: femenino, masculino o desconocido e indica eliminarlo.	2. El sistema muestra un mensaje de alerta solicitando confirmación.
3. El genetista confirma que desea eliminar el individuo seleccionado.	4. El sistema elimina el individuo seleccionado.
Flujos Alternos Sección "Eliminar un individuo"	

Acción del Actor	Respuesta del Sistema
3.1 El genetista confirma que no desea eliminar y sale de la sección.	
Prototipo de Interfaz	Anexo 1.2
Sección "Modificar datos de un individuo"	
Acción del Actor	**Respuesta del Sistema**
	1. Se ejecuta la Sección "Mostrar datos de un individuo".
2. El genetista indica el o los datos a modificar.	3. El sistema en tiempo de ejecución verifica que los datos estén correctos.
	4. El sistema en tiempo de ejecución actualiza el o los datos modificados.
Flujos Alternos Sección "Modificar datos de un individuo"	
Acción del Actor	**Respuesta del Sistema**
2.1 El genetista no desea modificar los datos y sale de la sección.	4.1 En caso de que algún dato no esté correcto el sistema en tiempo de ejecución muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 1.1
Sección "Mostrar datos de un individuo"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona el individuo del que quiere ver los datos.	2. El sistema muestra los datos del individuo seleccionado.
Prototipo de Interfaz	Anexo 1.1
Sección "Trasladar gráficamente un individuo de posición"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona el individuo que quiere trasladar y señala el lugar hacia donde desea trasladarlo.	2. El sistema verifica si el individuo fue movido a una nueva posición.
	3. El sistema muestra el individuo en el lugar indicado por el genetista.
Flujos Alternos Sección "Trasladar gráficamente un individuo de posición"	
Acción del Actor	**Respuesta del Sistema**
	3.1 Si el genetista no movió al individuo sale de la sección.
Prototipo de Interfaz	
Poscondiciones	El sistema inserta, modifica, elimina, muestra o traslada un individuo.

2.5.3.2 Gestionar relaciones entre individuos

Caso de Uso:	Gestionar relaciones entre individuos.
Actores:	Genetista
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de relaciones entre individuos: • Crear relación entre individuos. • Eliminar relación entre individuos. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta

	las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Debe estar iniciada la aplicación, al menos un individuo representado y para eliminar una relación tiene que haber al menos dos individuos relacionados.
Referencias	R2.1, R2.2
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El genetista desea realizar una de las siguientes operaciones: • Crear relación entre individuos. • Eliminar relación entre individuos.	*2.* El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide crear una relación entre dos individuos, ir a Sección **"Crear relación entre individuos"**. • Si decide eliminar una relación entre dos individuos, ir a la Sección **"Eliminar relación entre individuos"**.
Sección "Crear relación entre individuos"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona al individuo que desea relacionar.	2. El sistema muestra la interfaz correspondiente para la creación de las distintas relaciones.
3. El genetista selecciona una de las siguientes relaciones: • Esposa • Esposo • Hermano • Hermana • Desconocido • Gemelo (homocigótico, dicigótico) • Hijo • Hija • Descendiente • Grupo (cantidad de integrantes) • Padre • Madre • Divorciar	
4. El genetista selecciona al individuo con quien lo va a relacionar.	5. El sistema actualiza en tiempo de ejecución los datos de los individuos relacionados anteriormente: • A la propiedad "Cantidad_de_hijos" se le asigna el valor de la cantidad de hijos representados en el árbol de ese individuo. • A la propiedad "Cantidad_de_relaciones" se le asigna el valor de la cantidad de relaciones

	representados en el árbol de ese individuo. • A la propiedad "Hijos" se le asigna el valor de la cantidad de hijos de cada sexo (sea femenino, masculino o desconocido) representados en el árbol de ese individuo.
	6. El sistema muestra la relación indicada por el genetista.
Flujos Alternos Sección "Crear relación entre individuos"	
Acción del Actor	**Respuesta del Sistema**
4.1 Si el genetista no desea relacionarlo con ningún individuo ya existente marca sobre la interfaz.	4.2 El sistema muestra gráficamente el individuo.
	4.3 El sistema asigna en tiempo de ejecución: • A la propiedad "Cantidad_de_hijos" se le asigna el valor de la cantidad de hijos representados en el árbol de ese individuo. • A la propiedad "Cantidad_de_relaciones" se le asigna el valor de la cantidad de relaciones representados en el árbol de ese individuo. • A la propiedad "Hijos" se le asigna el valor de la cantidad de hijos de cada sexo (sea femenino, masculino o desconocido) representados en el árbol de ese individuo.
	4.4 El sistema crea el nuevo individuo.
	4.5 El sistema muestra la relación indicada.
Prototipo de Interfaz	Anexo 2
Sección "Eliminar relación entre individuos"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona el individuo que desea eliminar alguna de sus relaciones.	2. El sistema muestra la interfaz correspondiente para eliminar la relación entre dos individuos.
3. El genetista selecciona al otro individuo con el cual desea eliminar la relación.	4. El sistema elimina la relación indicada.
	5. El sistema muestra a los individuos seleccionados por el genetista sin relación entre ellos.
Flujos Alternos Sección "Eliminar relación entre individuos"	
Acción del Actor	**Respuesta del Sistema**
3.1 Si el genetista no selecciona al otro individuo con el cuál desea eliminar la relación sale de la sección.	
Prototipo de Interfaz	Anexo 2
Poscondiciones	El sistema crea o elimina una relación entre individuos.

2.5.3.3 Gestionar símbolo

Caso de Uso:	Gestionar símbolo
Actores:	Genetista (inicia)
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de un símbolo : • Crear nuevo símbolo. • Modificar símbolo. • Eliminar un símbolo. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Para eliminar un símbolo, este debe haber sido creado anteriormente por el genetista.
Referencias	R3.1, R3.2, R3.3
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El genetista desea realizar una de las siguientes operaciones: • Crear nuevo símbolo. • Modificar símbolo. • Eliminar un símbolo.	2. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide crear un nuevo símbolo, ir a Sección **"Crear nuevo símbolo"**. • Si decide modificar un símbolo, ir a la Sección **"Modificar símbolo"**. • Si decide eliminar símbolo, ir a la Sección **"Eliminar un símbolo"**.

Sección "Crear un nuevo símbolo"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente para crear un nuevo símbolo.
2. El genetista provee los datos para crear un nuevo símbolo. • Caracterización • Descripción	3. El sistema verifica que los campos tengan valor.
	4. El sistema crea el nuevo símbolo.

Flujos Alternos Sección "Crear un nuevo símbolo"	
Acción del Actor	**Respuesta del Sistema**
	4.1. Si existe al menos un parámetro sin valor entonces el sistema muestra un mensaje de alerta.

Prototipo de Interfaz	Anexo 3.1

Sección "Modificar símbolo"	
Acción del Actor	**Respuesta del Sistema**
2. El genetista selecciona el símbolo que desea cambiarle la descripción.	3. El sistema muestra la interfaz correspondiente para modificar el símbolo

	seleccionado.
4. El genetista provee la descripción para modificar el símbolo.	5. El sistema verifica que el campo tengan valor.
	6. El sistema guarda los cambios en el símbolo.
Flujos Alternos Sección "Modificar símbolo"	
Acción del Actor	**Respuesta del Sistema**
	6.1. Si el parámetro no tiene valor entonces el sistema muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 3.2
Sección "Eliminar símbolo"	
Acción del Actor	**Respuesta del Sistema**
2. El genetista selecciona uno de los símbolos creado por él que desea eliminar.	3. El sistema elimina el símbolo seleccionado por el genetista.
Prototipo de Interfaz	Anexo 3.3
Poscondiciones	El sistema crea, modifica o elimina un símbolo creado.

2.5.3.4 Gestionar enfermedad

Caso de Uso:	Gestionar enfermedad
Actores:	Genetista (inicia)
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de una enfermedad: • Crear nueva enfermedad. • Eliminar enfermedad. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Que la aplicación esté abierta, para eliminar una enfermedad debe estar anteriormente creada por el genetista.
Referencias	R4.1, R4.2
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El genetista desea realizar una de las siguientes operaciones: • Crear nueva enfermedad. • Eliminar enfermedades.	2. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide crear una nueva enfermedad, ir a Sección **"Crear nueva enfermedad"**. • Si decide eliminar una enfermedad, ir a la Sección **"Eliminar enfermedades"**.
Sección "Crear nueva enfermedad"	

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra la interfaz correspondiente para la creación de una nueva enfermedad.
2. El genetista provee el nombre de la enfermedad que desea adicionar.	3. El sistema verifica que el campo tengan valor.
	4. El sistema guarda la nueva enfermedad.
Flujos Alternos Sección "Crear nueva enfermedad"	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea proveer el nombre de la enfermedad va a la acción 1 del Flujo Normal de Eventos.	4.1. Si el parámetro no tiene valor entonces el sistema muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 4
Sección "Eliminar enfermedad"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona la enfermedad que desea eliminar.	2. El sistema elimina la enfermedad seleccionada por el genetista.
Prototipo de Interfaz	Anexo 4
Poscondiciones	El sistema crea o elimina una enfermedad.

2.5.3.5 Gestionar muestra

Caso de Uso:	Gestionar muestra
Actores:	Genetista
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de una muestra: • Crear nueva muestra. • Modificar muestra. • Eliminar muestra. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Debe estar seleccionado un individuo en la aplicación.
Referencias	R5.1, R5.2, R5.3
Prioridad	Secundario
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente a la gestión de la muestra.

2. El genetista desea realizar una de las siguientes operaciones: • Crear nuevas muestra. • Modificar muestra. • Eliminar muestra.	*3*. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide crear una nueva muestra, ir a Sección **"Crear nueva muestra"**. • Si decide modificar los datos de una muestra, ir a la Sección **"Modificar muestra"**. • Si decide eliminar una muestra, ir a la Sección **"Eliminar muestra"**.
Flujos Alternos de Eventos	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea realizar ninguna operación.	2.2. El sistema verifica si hay acciones sin guardar.
	2.3. El sistema muestra un mensaje de confirmación, dando la posibilidad de cancelar, guardar o no lo realizado.
2.4. El genetista indica una de las siguientes operaciones: • Sí • No • Cancelar	2.5. El sistema realiza las siguientes acciones en dependencia de los seleccionado por el genetista: • Si el genetista seleccionó "Sí" entonces el sistema guarda las acciones realizadas que no ha sido guardadas y sale del caso de uso. • Si el genetista seleccionó "No" entonces el sistema no guarda las acciones realizadas que no han sido guardadas y sale del caso de uso. • Si el genetista seleccionó "Cancelar" entonces se va a la acción 2 del Flujo Normal de Eventos.
	2.3.1 Si el sistema verifica que no hay acciones sin guardar entonces sale del caso de uso.
Sección "Crear nueva muestra"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista indica adicionar la muestra y provee los datos de la misma. • Localización • Tipo	
2. El genetista indica guardar la muestra adicionada.	3. El sistema crea la nueva muestra y va a la acción 2 del Flujo Normal de Eventos.
Flujos Alternos Sección "Crear nueva muestra"	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea guardar entonces va a la acción 2 del Flujo Normal de Eventos.	

Prototipo de Interfaz	Anexo 5
Sección "Eliminar muestra"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona la muestra que desea eliminar e indica eliminarla.	
2. El genetista indica guardar que se eliminó la muestra.	3. El sistema elimina la muestra indicada y va a la acción 2 del Flujo Normal de Eventos.
Flujos Alternos Sección "Eliminar muestra"	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea guardar entonces va a la acción 2 del Flujo Normal de Eventos.	
Prototipo de Interfaz	Anexo 5
Sección "Modificar muestra"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista selecciona la muestra que desea modificar y provee los datos de la misma. • Localización • Tipo	
2. El genetista indica guardar los cambios realizados a la muestra seleccionada.	3. El sistema guarda los cambios realizados a la muestra y va a la acción 2 del Flujo Normal de Eventos.
Flujos Alternos Sección "Modificar muestra"	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea guardar entonces va a la acción 2 del Flujo Normal de Eventos.	
Prototipo de Interfaz	Anexo 5
Poscondiciones	El sistema crea, modifica o elimina una muestra.

2.5.3.6 Gestionar dimensiones del árbol

Caso de Uso:	Gestionar dimensiones del árbol
Actores:	Genetista
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de la dimensión del árbol genealógico: • Aumentar tamaño de los individuos. • Disminuir tamaño de los individuos. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Que la aplicación esté abierta, debe existir al menos un individuo en la aplicación.
Referencias	R6.1,R6.2

Prioridad	Auxiliar
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El genetista desea realizar una de las siguientes operaciones: • Aumentar tamaño de los individuos. • Disminuir tamaño de los individuos.	2. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide aumentar el tamaño de los individuos, ir a Sección **"Aumentar tamaño de los individuos"**. • Si decide disminuir el tamaño de los individuos, ir a la Sección **"Disminuir tamaño de los individuos"**.
Sección "Aumentar tamaño de los individuos"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista decide aumentar el árbol genealógico de un individuo.	2. El sistema aumenta las dimensiones del árbol genealógico representado.
Prototipo de Interfaz	Anexo 6.1
Sección "Disminuir tamaño de los individuos"	
Acción del Actor	**Respuesta del Sistema**
1. El genetista decide disminuir el árbol genealógico de un individuo.	2. El sistema disminuye las dimensiones del árbol genealógico representado.
Prototipo de Interfaz	Anexo 6.2
Poscondiciones	El sistema aumenta o disminuye el tamaño del árbol genealógico.

2.5.3.7 Gestionar familia

Caso de Uso:	Gestionar familia
Actores:	Genetista
Resumen:	El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de la familia: • Crear una nueva familia. • Buscar una familia. • Modificar datos de una familia. • Guardar familia. • Exportar familia. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones:	Que la aplicación esté abierta, si decide buscar y modificar familia debe existir la familia, y para exportar y guardar la familia debe estar creada.
Referencias	R7.1, R7.2, R7.3, R7.4, R7.5
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El genetista desea realizar una de las siguientes operaciones: • Crear una nueva familia. • Buscar una familia. • Modificar datos de una familia. • Guardar familia. • Exportar familia.	2. El sistema, en dependencia de la operación que solicita realizar, hace lo siguiente: • Si decide crear una nueva familia, ir a Sección **"Crear una nueva familia"**. • Si decide buscar una familia, ir a la Sección **"Buscar una familia"**. • Si decide modificar los datos de una familia, ir a Sección **"Modificar datos de una familia"**. • Si decide guardar una familia, ir a la Sección **"Guardar familia"**. • Si decide exportar una familia, ir a la Sección **"Exportar familia"**.
Sección "Crear una nueva familia"	
Acción del Actor	**Respuesta del sistema**
	1. El sistema muestra la interfaz visual principal vacía para crear la nueva familia.
	2. El sistema asigna: • Al campo "Individuos" le asigna el valor de la cantidad de individuos representados en el árbol genealógico.
3. El genetista provee los datos para crear una familia. • Dibujado_por • Fecha_de_creación • Última_modificación • Enfermedad • Identificador • Nombre	4. El sistema en tiempo de ejecución verifica que los datos estén correctos.
	5. El sistema crea la familia con sus datos.
Flujos Alternos Sección "Crear una nueva familia"	
Acción del Actor	**Respuesta del Sistema**
	1.1. Si se encuentra otra familia en pantalla, pasar a la Sección "Guardar familia".
3.1. Si el genetista no desea proveer los datos del la familia y se va a la acción 5 de la Sección "Crear una nueva familia".	
	5.1 En caso de que algún dato no esté correcto el sistema en tiempo de ejecución muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 7.1
Sección "Buscar una familia"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente para que el genetista busque la familia.

2. El genetista busca y escoge la familia a utilizar.	3. El sistema muestra en la interfaz principal la familia escogida.
Flujos Alternos Sección "Buscar una familia"	
Acción del Actor	**Respuesta del Sistema**
	1.1 Si se encuentra otra familia en pantalla, pasar a la Sección "Guardar familia".
Prototipo de Interfaz	Anexo 7.2
Sección "Modificar datos de una familia"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente con todos los datos de la familia.
2. El genetista realiza los cambios necesarios y/o actualizaciones pertinentes.	3. El sistema en tiempo de ejecución verifica que los datos estén correctos.
	4. El sistema actualiza el o los datos modificados.
Flujos Alternos Sección "Modificar datos de una familia"	
Acción del Actor	**Respuesta del Sistema**
2.1 El genetista no desea modificar los datos y sale de la sección.	4.1 En caso de que algún dato no esté correcto el sistema en tiempo de ejecución muestra un mensaje de alerta.
Prototipo de Interfaz	Anexo 7.1
Sección "Guardar familia"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente para guardar la familia.
2. El genetista escoge el lugar deseado.	3. El sistema guarda la familia en el lugar escogido por el genetista.
Flujos Alternos Sección "Guardar familia"	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el genetista no desea guardar la familia y sale de la sección.	
Prototipo de Interfaz	Anexo 7.3
Sección "Exportar familia"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz correspondiente para exportar la familia.
3. El genetista escoge el formato en que quiere exportar la familia: • Reporte del Árbol • Imagen del Árbol	4. El sistema muestra una interfaz visual donde permite escoger el lugar donde quiere exportarlo.
5. El genetista elige el lugar donde quiere exportar la familia.	6. El sistema exporta la familia en el lugar elegido por el Genetista en el formato seleccionado
Prototipo de Interfaz	Anexo 7.4
Poscondiciones	El sistema crea, busca, modifica, guarda y exporta una familia.

Conclusiones

En este capítulo quedaron identificados los actores y trabajadores del negocio, se describió detalladamente el caso de uso del negocio y el modelo de objeto del caso de uso del negocio Realizar árbol genealógico.

También se identificaron los requerimientos funcionales y no funcionales del sistema, así como el actor y los casos de uso del sistema. Se realizó el diagrama de casos de uso del sistema, mostrando la relación entre los casos de uso y el actor del sistema. Se describieron los casos de uso definidos y se elaboraron los prototipos de interfaz de los casos de uso descritos.

CAPÍTULO 3
DISEÑO DEL SISTEMA

En este capítulo se traducen los requisitos a una especificación que describe como implementar el sistema, a través del diseño, enfocado a cómo el sistema cumple sus objetivos teniendo en cuenta los requisitos funcionales y no funcionales, se realizan los diagramas de clases y los diagramas de interacción más relevantes según los casos de uso definidos en el capítulo anterior. Se explica además la arquitectura utilizada y los principales patrones de diseño utilizados.

3.1 ESTILO ARQUITECTÓNICO UTILIZADO

Teniendo en cuenta lo expuesto en el Capítulo 1 se utilizó para desarrollar la aplicación el patrón arquitectónico Modelo Vista Controlador. Este patrón de diseño se basa en la separación en tres capas del diseño de las aplicaciones: el modelo de datos, la presentación de los mismos y las acciones de los usuarios. Una ventaja de este patrón es que permite que desde la vista se pueda tener acceso a las clases del modelo, esto es una gran ventaja en aplicaciones como esta, en las que se hace un uso intensivo de componentes visuales y requieren de una actualización constante de las propiedades que se modifican. A continuación (Figura 6) se muestra este patrón de arquitectura.

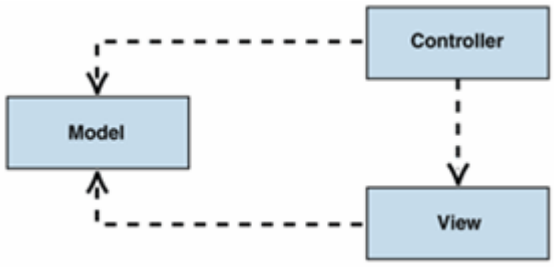


Fig. 6 Diagrama Modelo Vista Controlador

3.2 PATRONES DE DISEÑO UTILIZADOS

Los patrones de diseño son soluciones estándares a problemas comunes en el de desarrollo de software. En este epígrafe se describen los principales patrones de diseño utilizados en la solución de algunos problemas no triviales en la aplicación.

3.2.1 Patrón STATE

Problema: Una instancia de tipo *Desconocido* o *Mujer* puede cambiar su sexo o pasar a estar embarazada respectivamente lo que conlleva en ambos casos a que se comporten de otra manera.
Solución: Para esto aplicamos el patrón Gang of Four (GOF) de Comportamiento, a nivel de objeto STATE.
Aplicación: Una instancia de *Persona* actúa en dependencia de su sexo, es por esto que existen tres tipos de personas según su sexo: *Masculino, Femenino, Desconocido*. Una persona de tipo *Desconocido* tiene como atributo el evento *EventoCambioDeSexo* que al cambiarle su sexo se desencadena el mismo y su responsabilidad es informar que hay un cambio de sexo y cuál es el nuevo sexo, para que en la clase controladora *Familia* mediante el evento *EventoPersonaCambioDeSexo*, atributo de la misma, se cree un objeto de tipo *Persona* en dependencia del nuevo sexo, conservando el valor de sus atributos, garantizando así que según el cambio de estado se comporte como lo que es. En el caso de *Mujer,* al cambiar al estado de mujer embarazada o viceversa, se procede de la misma manera pero con los eventos *EventoCambioDeEmbarazo* y *EventoPersonaCambioDeEmbarazo*, se debe destacar que para el nuevo estado, el comportamiento es distinto y se define la clase *MujerEmbarazada* que es una especialización de la clase *Mujer*.

3.2.2 Patrón SINGLETON

Problema: Se necesita mantener de manera única las instancias de las listas de enfermedades, símbolos y símbolos adicionales en todo el tiempo de ejecución.
Solución: Para esto aplicamos el patrón GOF de Creación, a nivel de objetos SINGLETON.
Aplicación: Para dar solución a este problema se garantiza tener una sola instancia de estos objetos y por eso que se crea la clase *Listas* que tiene como atributo estas 3 listas (símbolos, simbolosAdicionales, enfermedades), todas declaradas como *static* al igual que sus propiedades. En esta clase además se tiene un método para cargar y otro para salvar en los ficheros de configuración por cada lista, estos también se declaran como *static y* permiten donde quiera que se modifican estas configuraciones poderlas salvar y siempre cargar la última configuración.

3.2.3 Patrones GRASP

3.2.3.1 Patrón Experto

Problema: Que las responsabilidades sean acorde a la información con que cuenta cada clase.
Solución: Para dar solución a esto se aplica el patrón Experto de los GRASP.
Aplicación: Es necesario garantizar que en cada cambio que se origine en tiempo de ejecución se repintan los controles con las nuevas características que puedan tener. Es por eso que a pesar de ser *FamiliaVisual* parte de la capa Vista se le asigna la responsabilidad de dibujar el árbol, ya que ella es la que tiene todos los controles que son los que representan a cada persona y controla los cambios que se puede producir en el árbol mediante los eventos. Esta responsabilidad se encuentra en el evento *paint* del control *FamiliaVisual* denominado *FamiliaVisual_Paint.*

3.2.3.2 Patrón Creador

Problema: Que solo cree instancias de objeto quien lo contenga y los registre.
Solución: Para dar solución a esto se aplica el patrón Creador de los GRASP.
Aplicación: La clase *FamiliaVisual* es la que contiene a todas la instancias de *PersonaVisual* y *Familia* las de *Persona*. Cada persona representada existente en el árbol tiene una clase *PersonaVisual* que actúa como vista y una clase *Persona* que pertenece al modelo para almacenar los datos. A la hora de insertar una nueva persona quien capta el mensaje es *FamiliaVisual* y esta crea a *PersonaVisual* que es lo que ella registra, informa a la controladora *Familia* que es la que registra a *Persona* para que cree la instancia de *Persona* correspondiente a la *PersonaVisual* antes creada.

3.2.3.3 Patrón Polimorfismo

Problema: Cómo manejar diferentes comportamientos en dependencia del tipo de objeto..
Solución: Para dar solución a esto se aplica el patrón Polimorfismo de los GRASP
Aplicación: A la hora de adicionar una relación conyugal a una persona se tiene que verificar que no sea de su mismo sexo, o sea el comportamiento de esta responsabilidad es en dependencia del tipo de persona en cuestión. Para esto se crea una responsabilidad en la clase *Persona* que cada una de sus especializaciones la redefine según su tipo y de esta manera se garantiza que el comportamiento sea según el tipo, y se evita programar un método que implique sentencias condicionales o el uso del switch { case }.
Ejemplo:

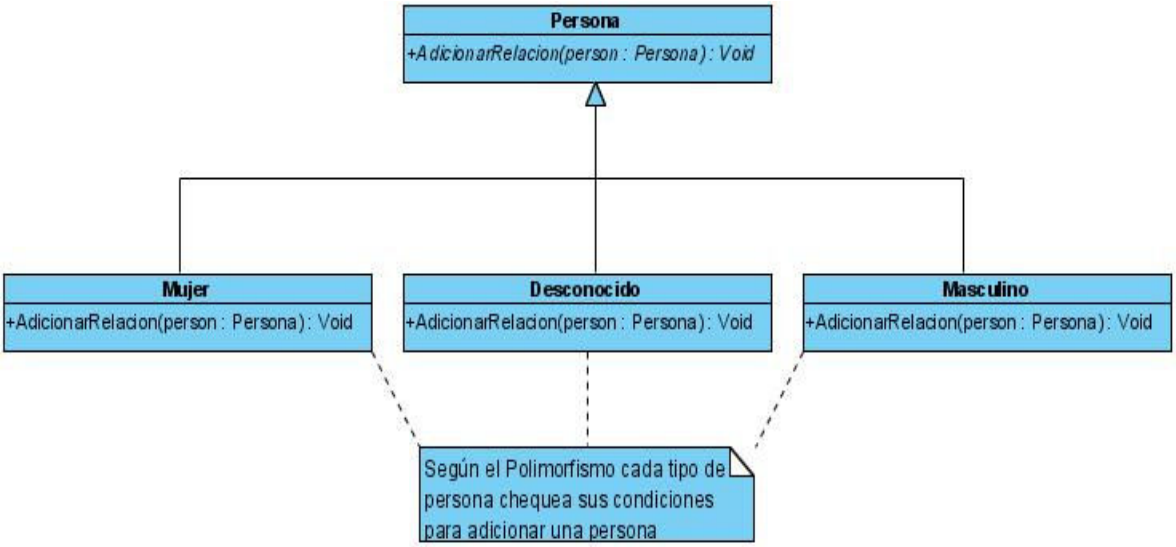


Fig. 7 Ejemplo del patrón Polimorfismo en la clase *Persona*

3.3 DIAGRAMAS DE CLASES

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces de la aplicación. Sirve también para visualizar las relaciones entre las clases que involucran el sistema.

Se describen a continuación algunos aspectos relevantes del diseño.

Por cada evento se representa su Delegado correspondiente. Las clases editores representan editores que se definen para algunas propiedades de la clase Persona y la clase Familia, estos se utilizarán a la hora de visualizar las propiedades de dichas clases en el Visor de Propiedades que es de tipo PropertyGrid. Para un mejor entendimiento de los diagramas que se realizaron por caso de uso. Las clases aparecen con sus métodos y propiedades ocultadas en los diagramas del diseño. Para consultar todas sus propiedades y métodos remitirse al Anexo 8). El diseño de la aplicación por casos de uso se puede apreciar en las figuras: Fig. 8 hasta la Fig. 14.

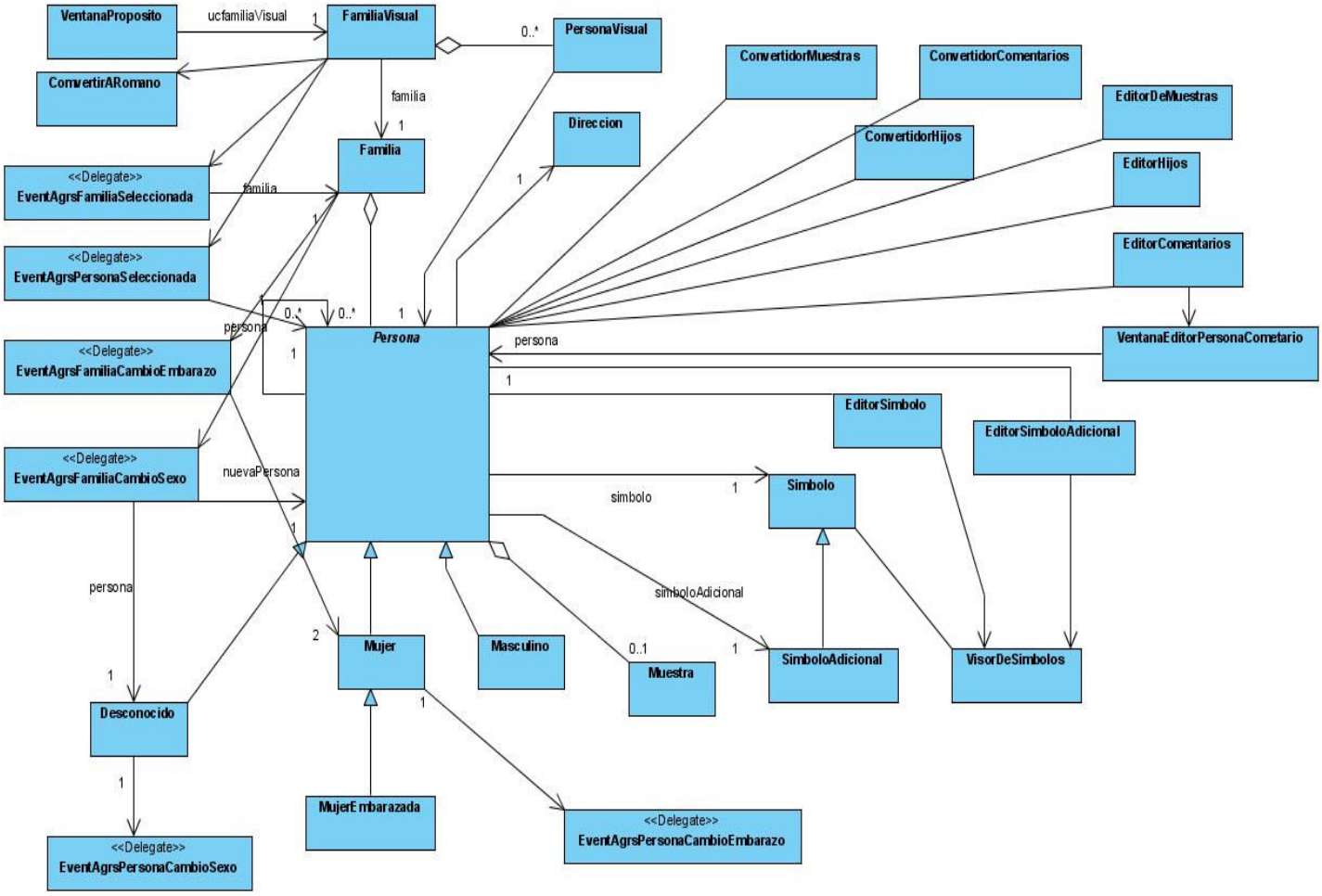


Fig. 8 Diagrama de clases del diseño: CU Gestionar gráficamente un individuo

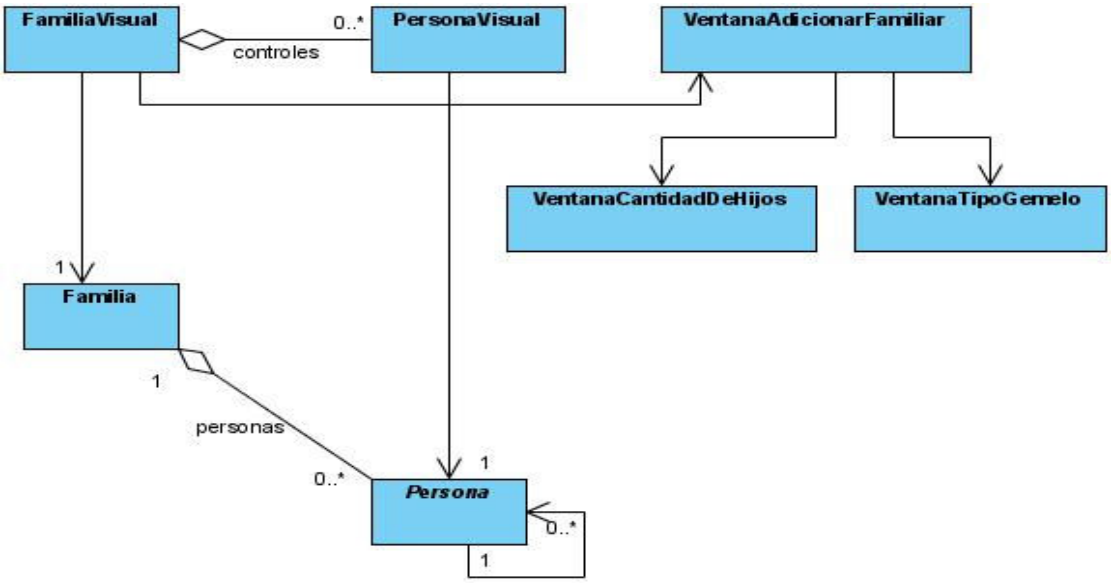


Fig. 9 Diagrama de clases del diseño: CU Gestionar relaciones entre individuos

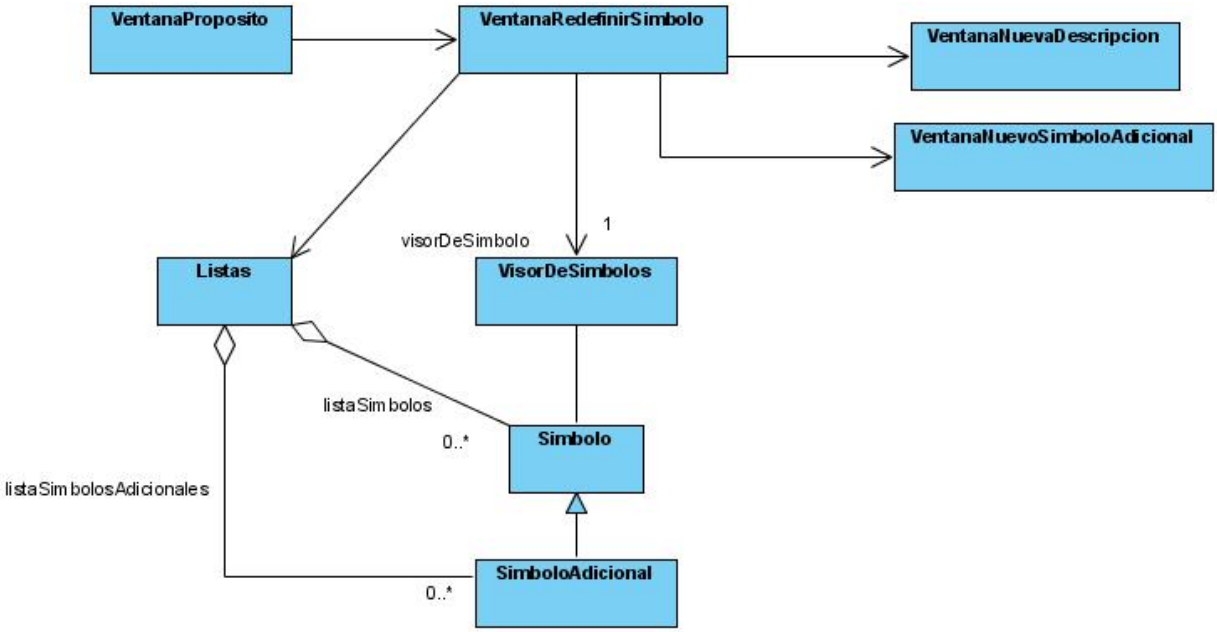


Fig. 10 Diagrama de clases del diseño: CU Gestionar símbolo

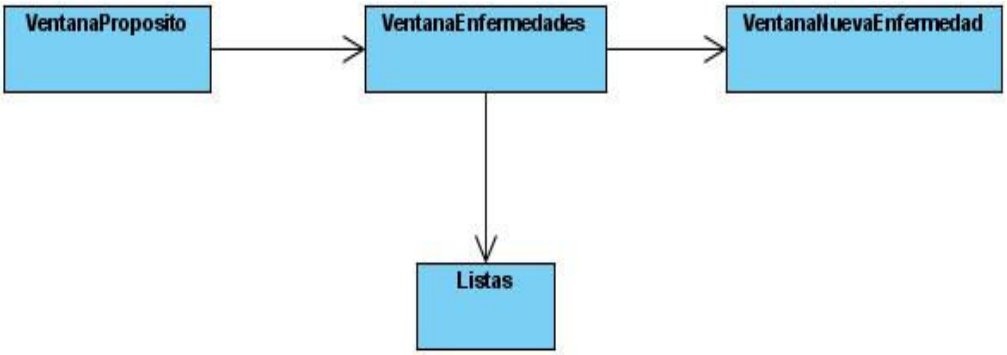


Fig. 11 Diagrama de clases del diseño: CU Gestionar enfermedad

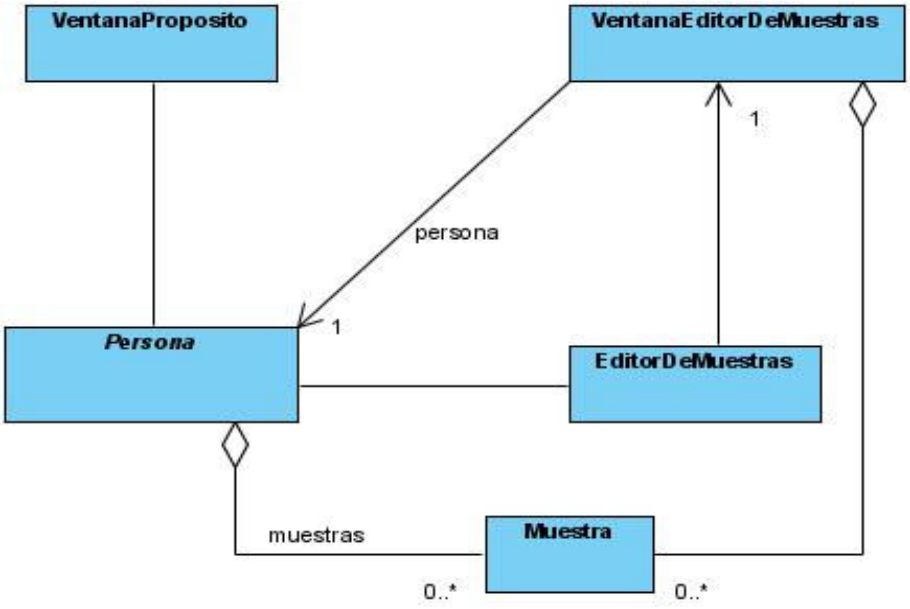


Fig. 12 Diagrama de clases del diseño: CU Gestionar muestra

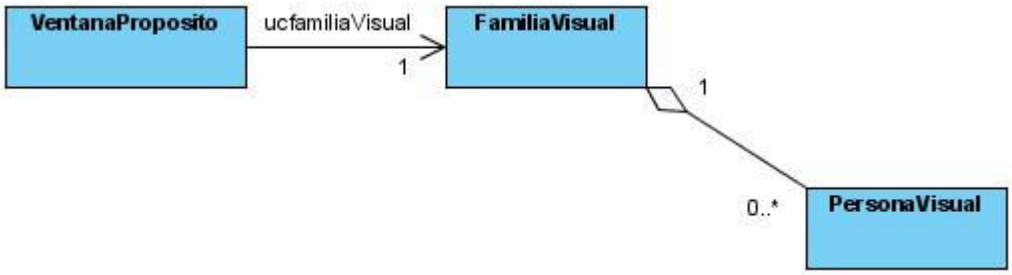


Fig. 13 Diagrama de clases del diseño: CU Gestionar dimensiones del árbol

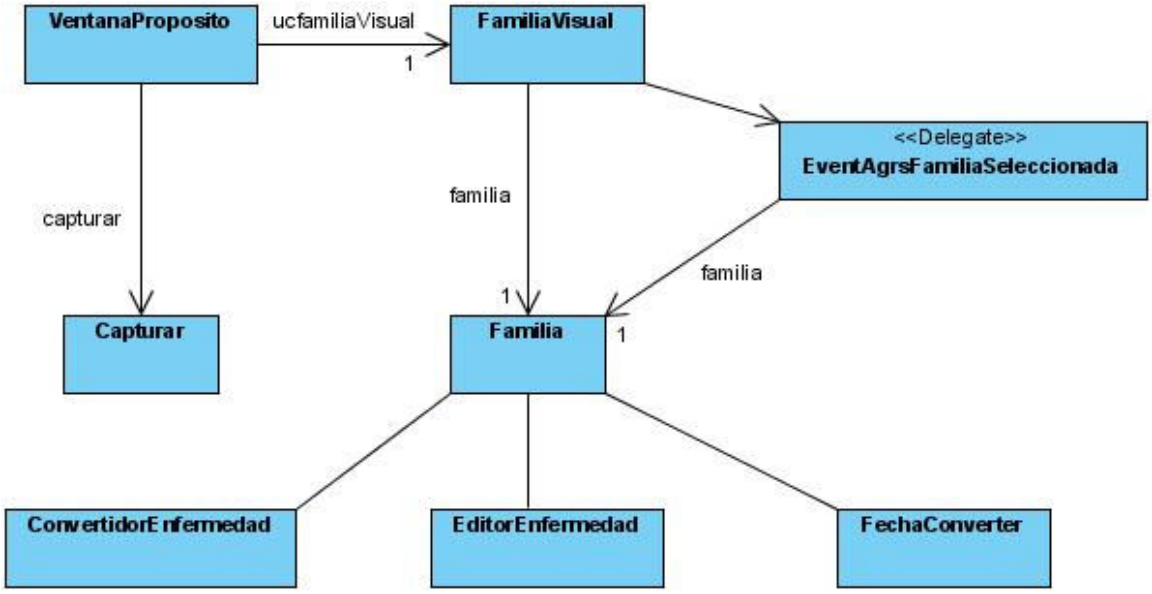


Fig. 14 Diagrama de clases del diseño: CU Gestionar familia

3.4 DIAGRAMAS DE SECUENCIA

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal; muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. [18]

Se realizaron los diagramas de secuencia que representan las interacciones de los objetos de la aplicación. A continuación se muestran los principales, seleccionados en correspondencia con los casos de usos críticos definidos en la realización de los CUS. (Para consultar los restantes diagramas remitirse al Expediente del Proyecto.)

La Fig. 15 representa la interacción entre las clases que intervienen en la Sección Insertar un individuo del CU Gestionar gráficamente un individuo. En este diagrama se evidencia cómo, en dependencia del sexo seleccionado, es el tipo de persona que se crea. Por cada persona a representar es necesario tener la clase PersonaVisual y Persona. La clase PersonaVisual es un componente de FamiliaVisual, por lo que esta última es la responsable de crear instancias de PersonaVisual. En el caso de la clase Persona, sus instancias son creadas por la clase controladora Familia.

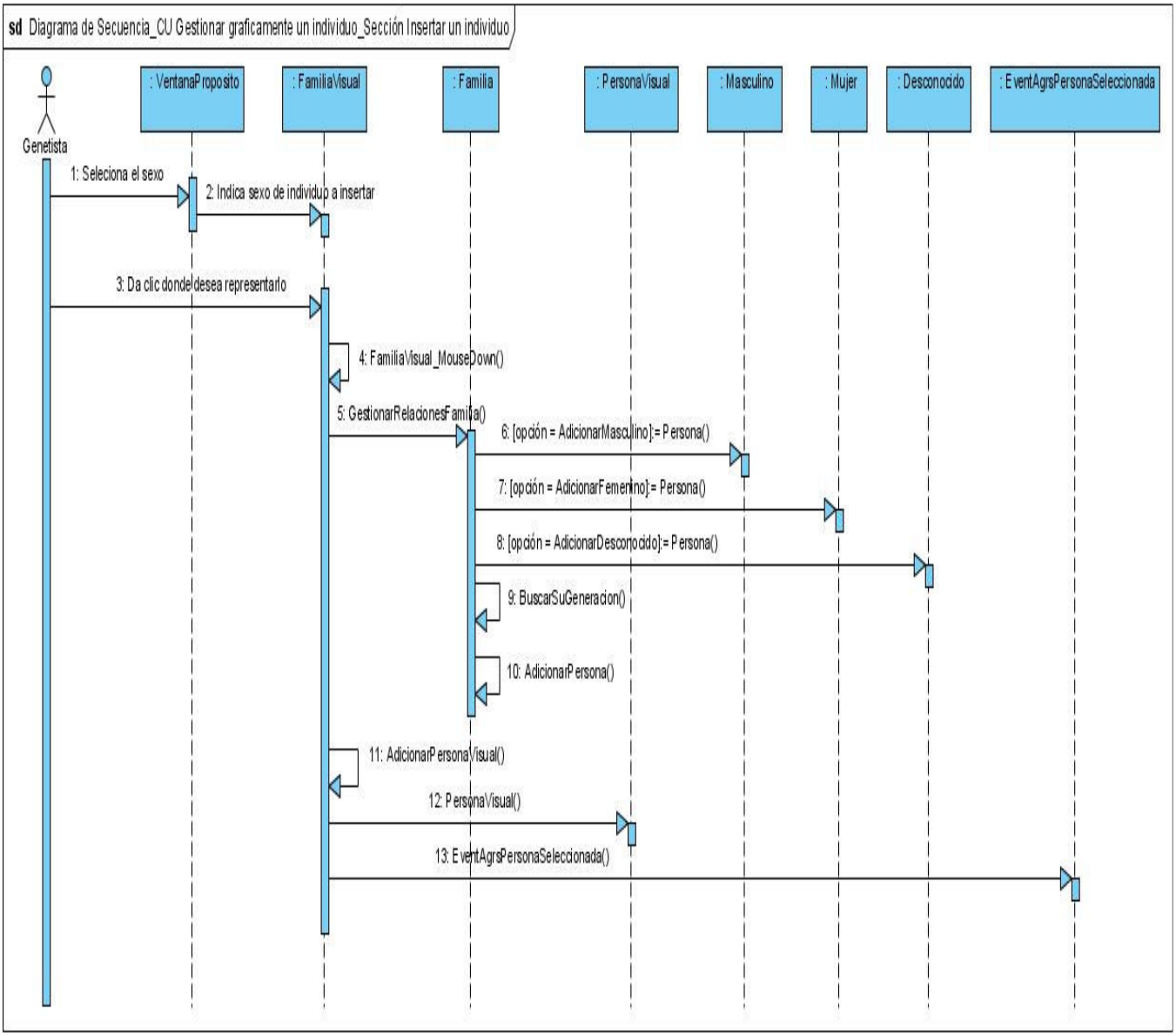


Fig. 15 Diagrama de secuencia del CU Gestionar gráficamente un individuo, Sección Insertar un individuo

La Fig. 16 representa la interacción entre las clases que intervienen en la Sección Eliminar individuo del CU Gestionar gráficamente un individuo.

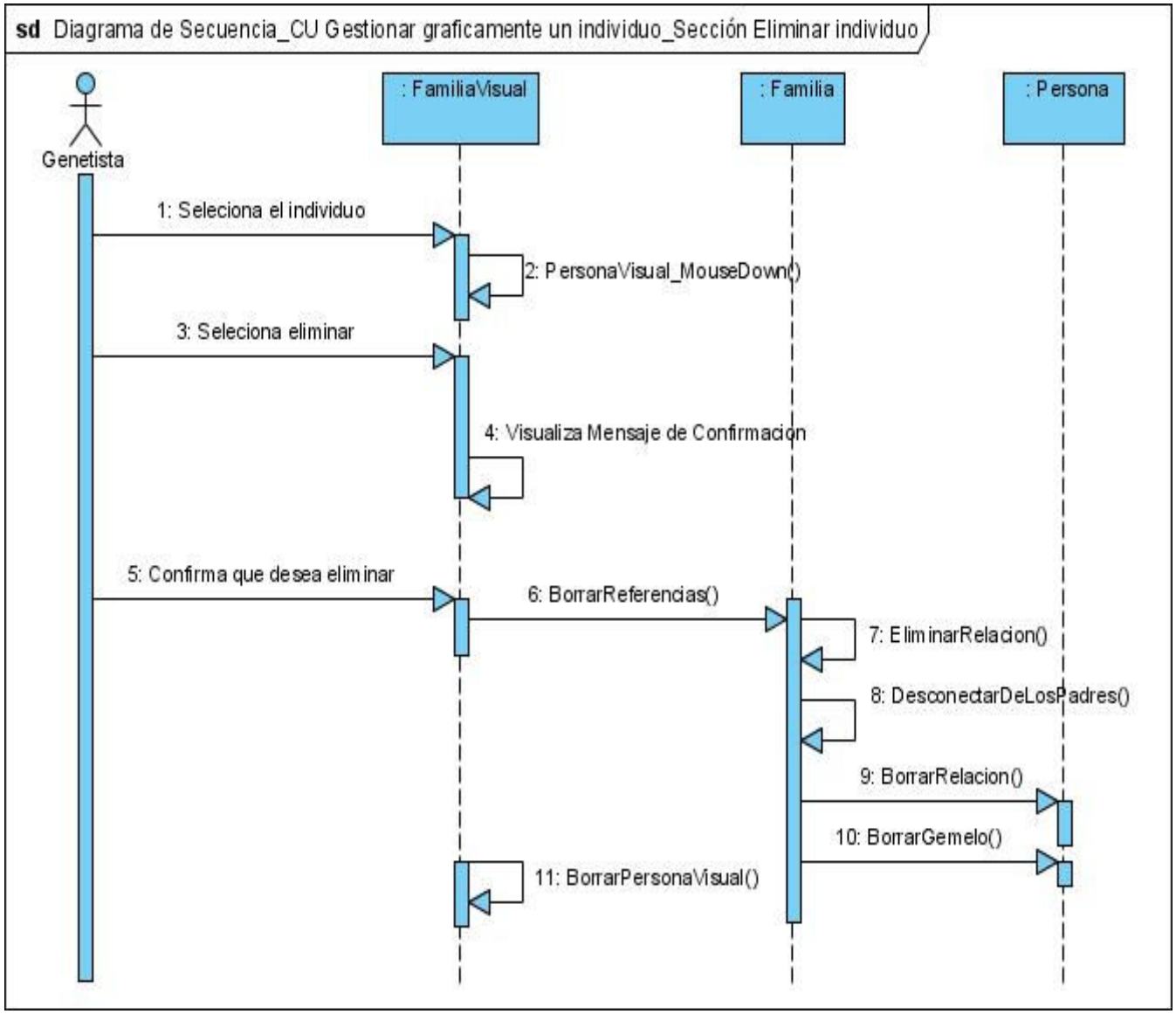


Fig. 16 Diagrama de secuencia del CU Gestionar gráficamente un individuo, Sección Eliminar individuo

La Fig. 17 representa la interacción entre las clases que intervienen en la Sección Crear relación entre individuos del CU Gestionar relaciones entre individuos. Este diagrama representa la interacción entre las clases, al adicionar una relación entre dos individuos. Primeramente se selecciona el tipo de relación que se desea adicionar al individuo en cuestión. Luego se puede marcar sobre la interfaz o un individuo existente. Si se realiza la primera opción el sistema crea un nuevo individuo y la relación entre ambos; en caso contrario, el sistema solo crea la relación.

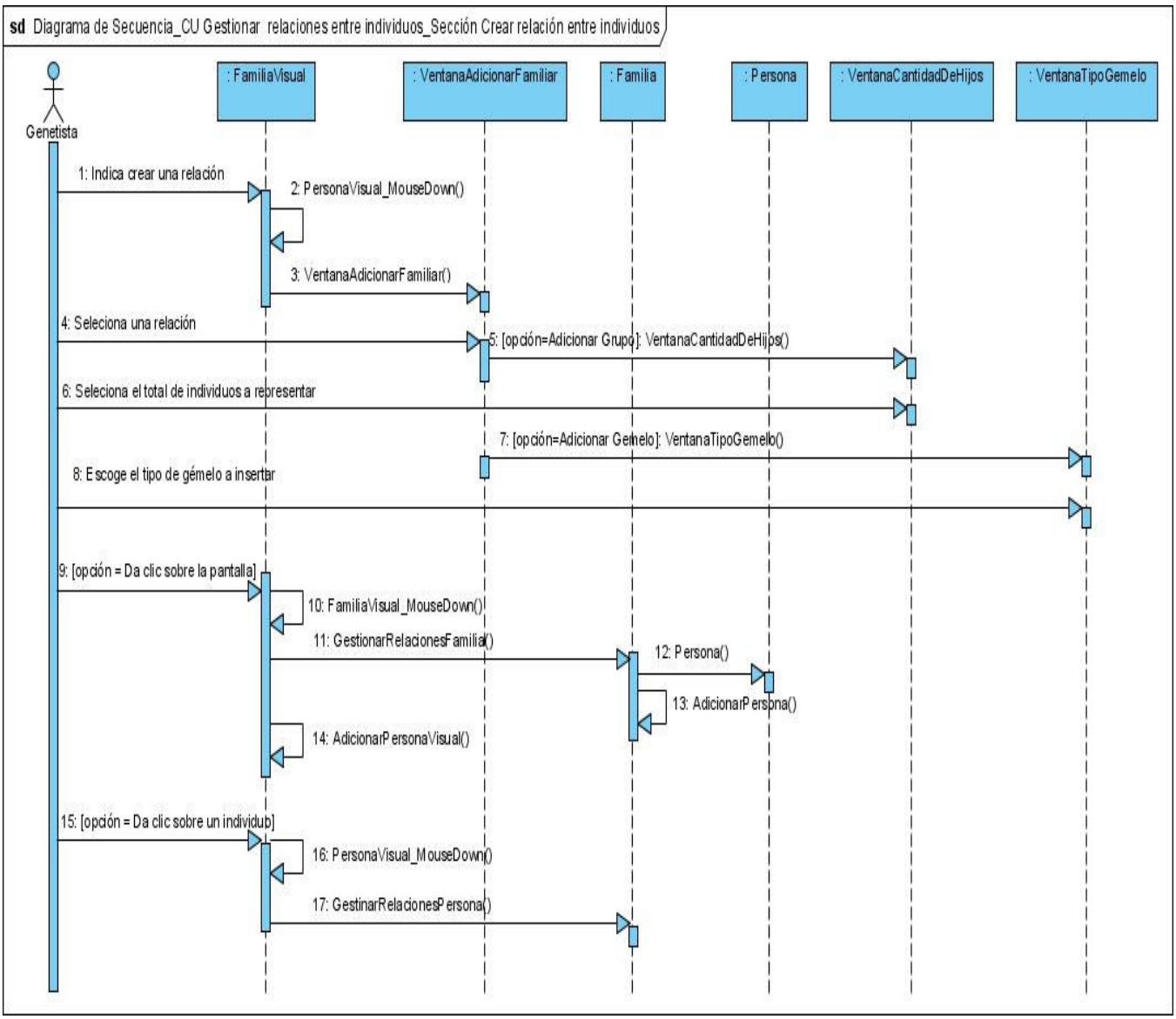


Fig. 17 Diagrama de secuencia del CU Gestionar relaciones entre individuos, Sección Crear relación entre individuos

La Fig. 18 representa la interacción entre las clases que intervienen en la Sección Crear nuevo símbolo del CU Gestionar símbolo. En la clase *Listas se guardan* los símbolos y símbolos adicionales cuando el genetista en la interfaz *VentanaRedefinirSimbolos* indique terminar, para mantener la persistencia de dichos símbolos

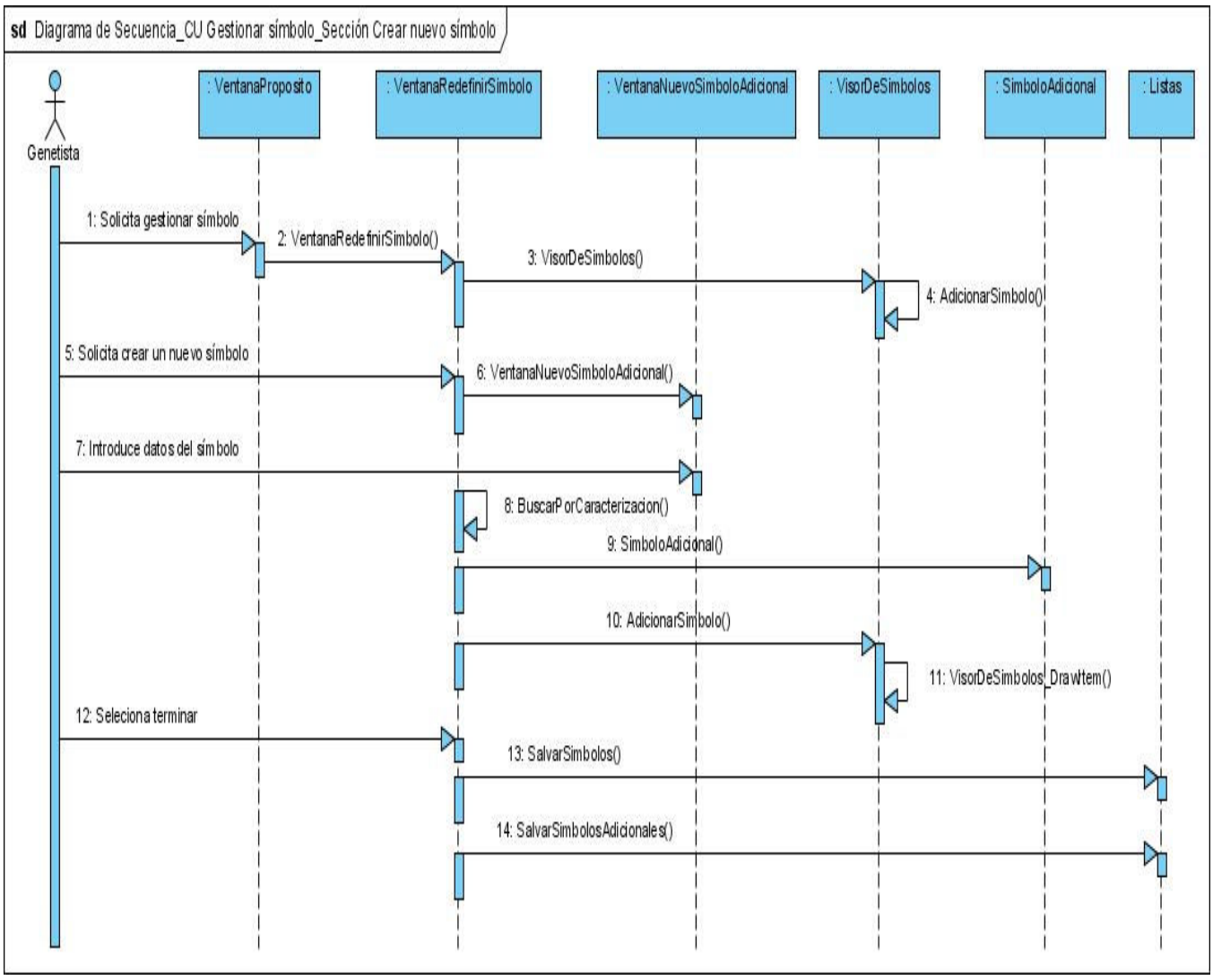


Fig. 18 Diagrama de secuencia del CU Gestionar símbolo, Sección Crear nuevo símbolo

La Fig. 19 representa la interacción entre las clases que intervienen en la Sección modificar símbolo del CU Gestionar símbolo. Después de modificado un símbolo se debe repintar el control para representar estos cambios con el método DrawItem() de la clase *VisorDeSimbolo.*

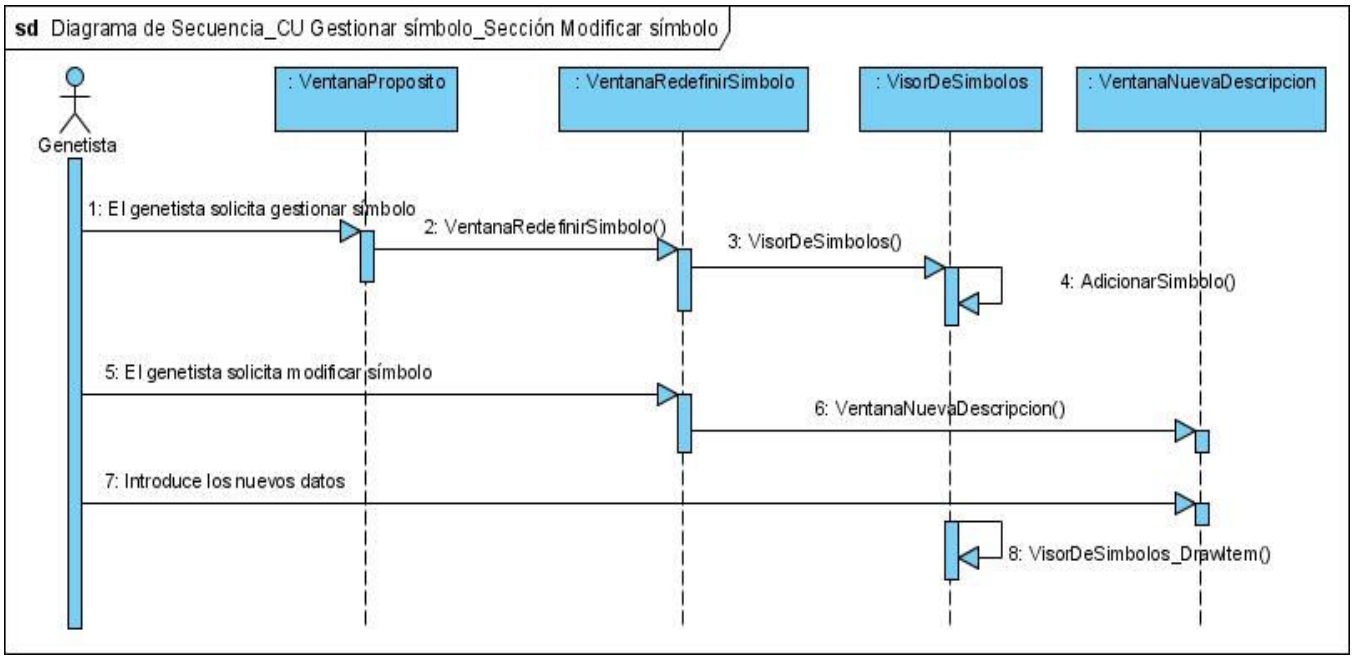


Fig. 19 Diagrama de secuencia del CU Gestionar símbolo, Sección Modificar símbolo

La Figura 20 representa la interacción entre las clases que intervienen en la Sección Crear nueva enfermedad del CU Gestionar enfermedades. En la clase *Listas se guardan* las enfermedades cuando el genetista en la interfaz *VentanaEnfermedades* indique terminar, para mantener la persistencia de dichas enfermedades.

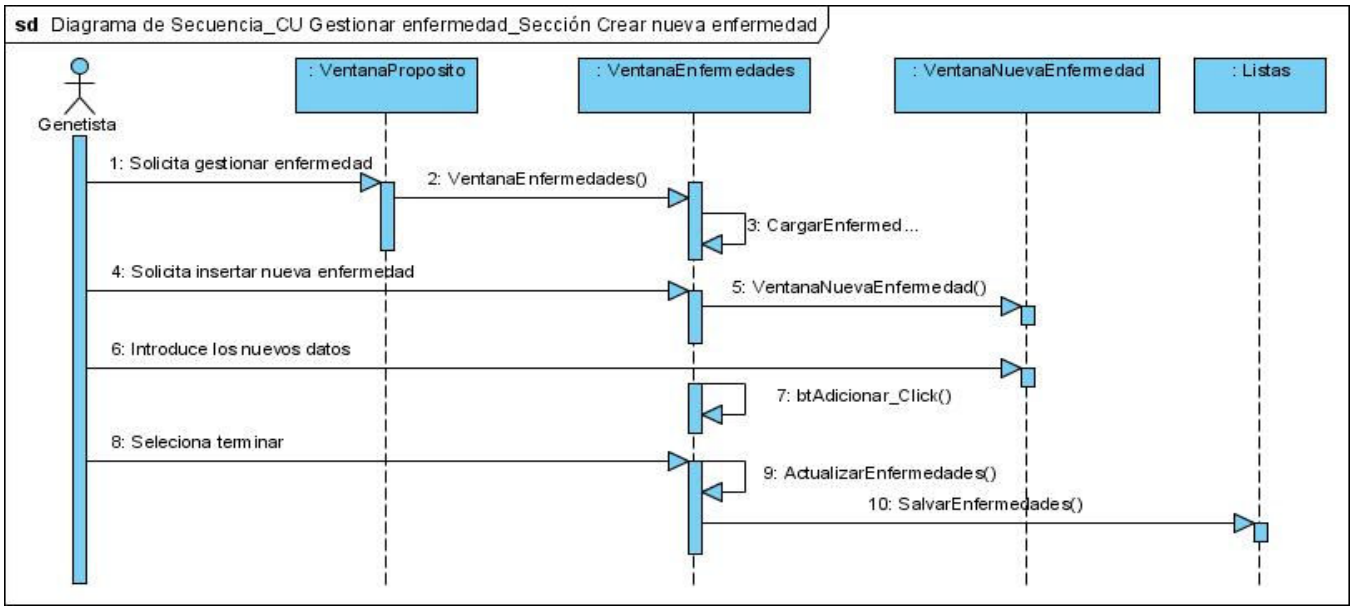


Fig. 20 Diagrama de secuencia del CU Gestionar enfermedad, Sección Crear nueva enfermedad

La Fig. 21 representa la interacción entre las clases que intervienen en la Sección Crear nueva muestra del CU Gestionar muestras. El genetista interactúa con el componente *VisorDePropiedades* de la clase interfaz V*entanaProposito,* que contiene un objeto de tipo clase *Persona* correspondiente al individuo seleccionado. Además se define el editor *EditorDeMuestras* para la propiedad *Muestras* de la clase *Persona*. Al acceder a esta propiedad mediante el *VisorDePropiedades,* se invoca la clase *EditorDeMuestras*, que construye a la clase *VentanaEditorDeMuestras* que permite al usuario gestionar las muestras.

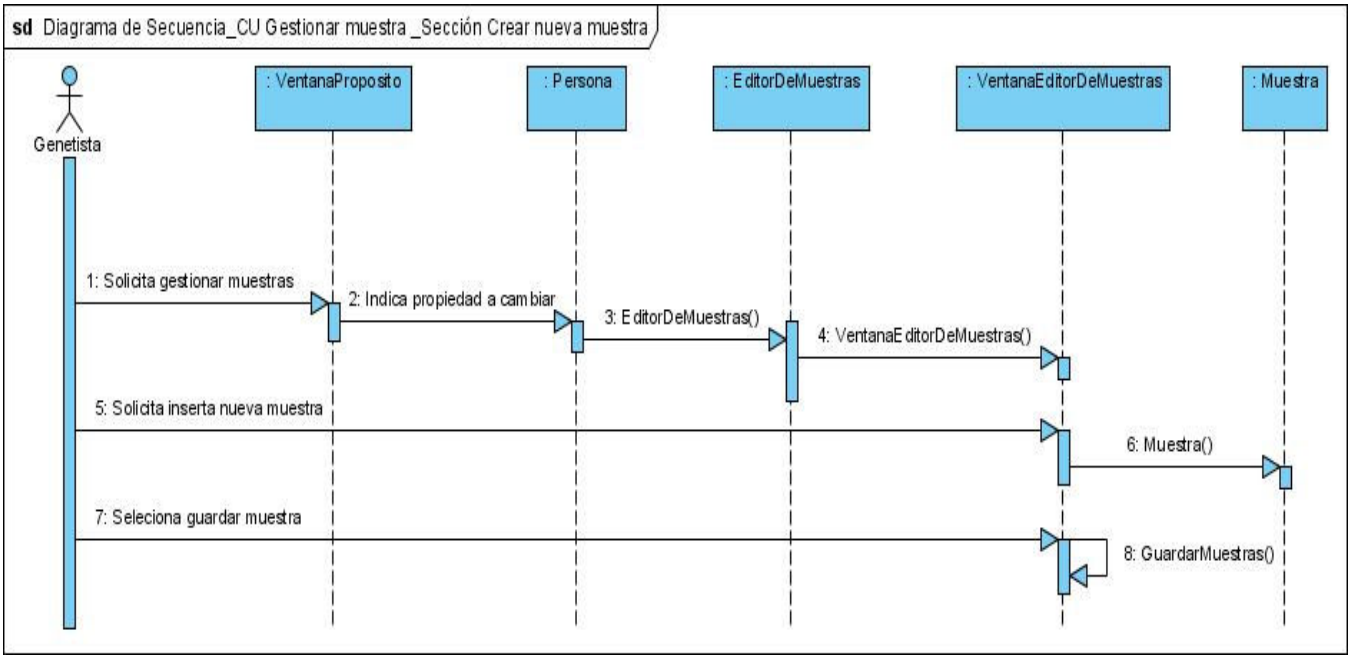


Fig. 21 Diagrama de secuencia del CU Gestionar muestra, Sección Crear nueva muestra

La Figura 23 representa la interacción entre las clases que intervienen en la Sección Actualizar datos de una familia del CU Gestionar familia. La familia, al igual que la persona, al estar seleccionada muestra sus propiedades en el *VisorDePropiedades*. En este caso se le define una clase *EditorEnfermedades* a la propiedad Enfermedad, permitiendo mostrar todas las enfermedades definidas anteriormente. Además a esta misma propiedad se le define un convertidor de tipo *ConvertidorEnfermedades* para personalizar la forma de mostrar la enfermedad en el *VisorDePropiedades*. También se utiliza un convertidor *FechaConverter* para las propiedades de tipo fecha permitiendo personalizar el formato de visualización. Debido a lo anteriormente explicado se utilizan condicionales para los casos en que se seleccione una de estas propiedades. En el resto de las propiedades se actualiza de forma directa mediante el *VisorDePropiedades*.

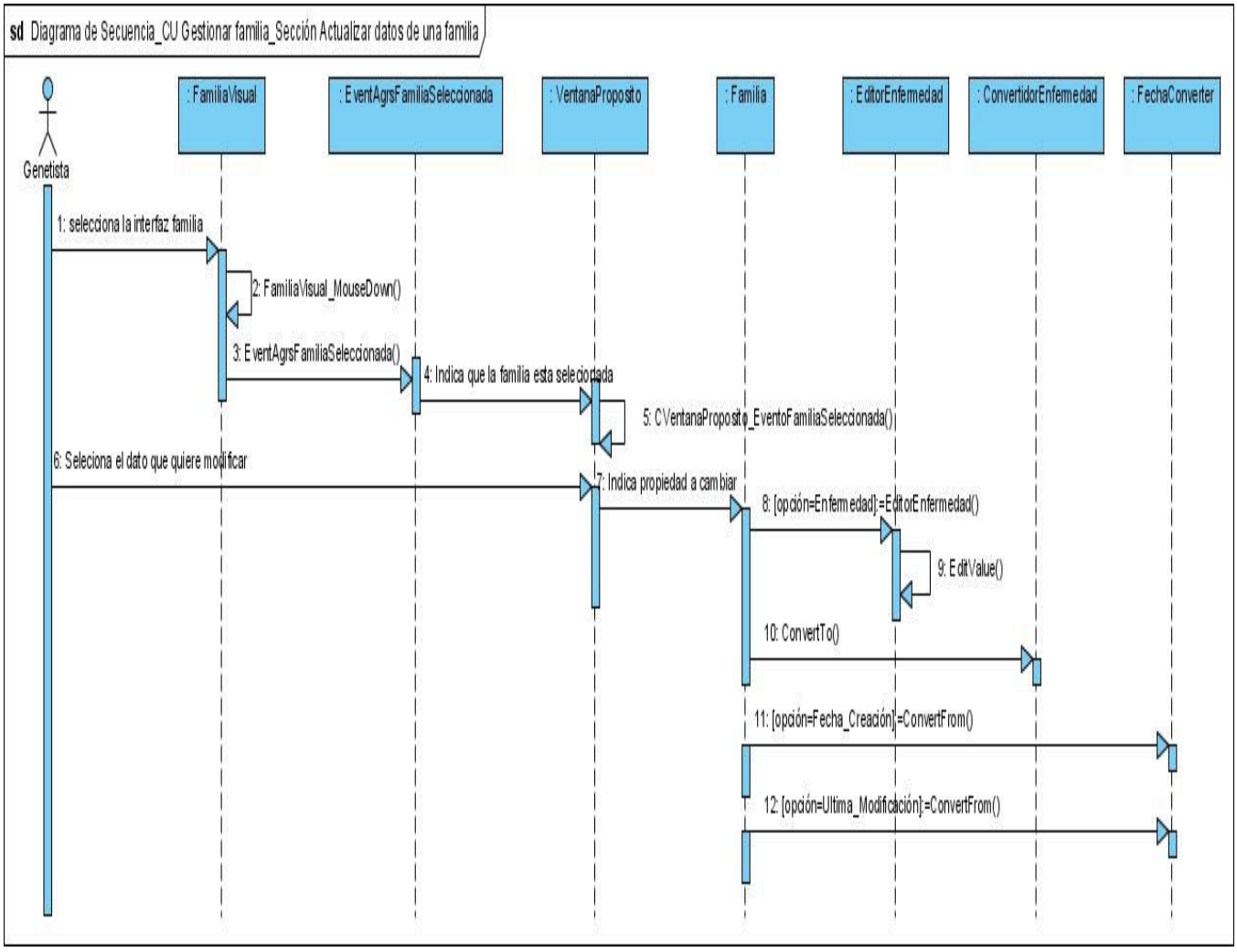


Fig. 22 Diagrama de secuencia del CU Gestionar familia, Sección Actualizar datos de una familia

Conclusiones

Como resultados de este capítulo, se representan los diagramas de clases del diseño y los diagramas de interacción del sistema de los cuales se representan los más significativos por cada caso de uso. En todos los casos se hace una explicación de forma general del diseño así como de los temas menos comunes, ejemplo, los editores de las propiedades de una clase. Además de la fundamentación del uso de los patrones de diseño y arquitectura utilizados.

CAPÍTULO 4
IMPLEMENTACIÓN DEL SISTEMA

A través de la realización de este capítulo se describe cómo fue implementada la aplicación en términos de componentes. Se describen algunos fragmentos de códigos no triviales en la clase *VisorDeSimbolos* y la funcionalidad de guardar una familia. También se propone un patrón para estandarizar el formato de guardar los árboles genealógicos. Se presenta la validación de las pruebas de calidad desarrolladas por la Dirección de Calidad en la UCI. Además se hace un análisis de la solución obtenida.

4.1 DIAGRAMA DE COMPONENTES

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes. Un diagrama de componentes muestra las organización y dependencias lógicas entre componentes software, sean estos componentes de código fuente, binarios, archivos, paquetes, bibliotecas cargadas dinámicamente o ejecutables.

El diagrama de componentes está compuesto por el fichero alasARBOGEN.exe que es el ejecutable de la aplicación y los ficheros: enfermedades.txt, simbolosAdicionales.txt y simbolos.txt que son los responsables de guardar toda la configuración de la aplicación. El fichero HelpArgen.chm es el que permite mostrar la ayuda de la aplicación.

A continuación se muestra el diagrama de componentes de la aplicación desarrollada.

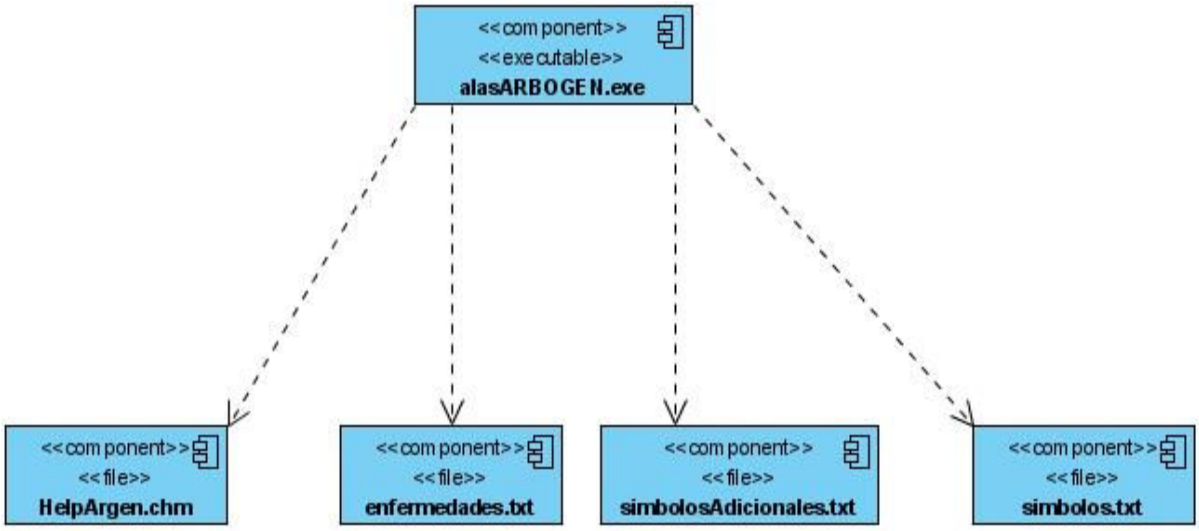


Fig. 23 Diagrama de Componentes

En el lenguaje C#, por cada clase se genera un fichero nombre.cs y por cada Forma dos archivos nombre.cs y nombre.resx, de los cuales el punto resx es el encarado de guardar todas las configuraciones necesarias en cada forma y el .cs contiene la programación de la clase.

4.2 FRAGMENTOS DE CÓDIGO

A continuación se describen algunas clases y responsabilidades no triviales que se utilizan en el código, además se explica la clase *VisorDeSimbolo* que es la que interactúa con el usuario al gestionar símbolos. También se expone la responsabilidad *GuardarFamilia* que es la encargada de guardar una familia para posteriormente cargarla y se hace una propuesta de patrón a seguir para guardar archivos de estos tipos de aplicaciones.

4.2.1 Principales responsabilidades de la clase *VisorDeSimbolos*

Símbolos es una clase que forma parte del modelo, la misma tiene como atributos una descripción y un tipo de símbolo de tipo enumerador. Cada símbolo se representa de acuerdo a su tipo. Se hace necesario que a la hora del usuario interactuar con estos le sean representados visualmente de manera que correspondan a cada uno y así garantizar una interrelación dinámica con el mismo.

Para dar solución a lo antes planteado se implementa la clase *VisorDeSimbolos* la cual hereda de *System.Windows.Forms.ListBox* que es una clase del sistema. Esto permite reutilizar eventos del componente y acciones ya definidas en la clase base, como el *SelectedIndex* y el *DrawItem*.

La responsabilidad de esta clase de forma global es dibujar los símbolos definidos con sus descripciones correspondientes. Para eso se le adiciona cada símbolo como un Item.

```
public void AdicionarSimbolo(Simbolo simbolo)
{
      Items.Add(simbolo);
}
```

Esta responsabilidad se invoca a la hora de cargar los símbolos a visualizar.

Para eliminar un símbolo se aprovecha la propiedad de la clase base *SelectedIndex*, permitiendo que el usuario seleccione directamente el símbolo y lo mande a eliminar.

```
public void EliminarSimbolo()
{
      Items.RemoveAt(SelectedIndex);
}
```

Y finalmente en la responsabilidad de VisorDeSimbolos_DrawItem es donde se dibuja cada Símbolo según las reglas del negocio.

```
private void VisorDeSimbolos_DrawItem(object sender,
System.Windows.Forms.DrawItemEventArgs e)
{
if (e.Index>=0)
{
#region Variables para hacer los cálculos

/// Brocha para rellenar los símbolos
SolidBrush brocha = new SolidBrush(Color.Black);

/// Fuentes de las letras
Font font = new Font(Font.FontFamily, e.Bounds.Height - 6, FontStyle.Bold);


/// Rectángulo donde voy a poner la cadena
Rectangle rectString = new Rectangle(e.Bounds.Location, new
Size(e.Bounds.Height, e.Bounds.Height));

/// Formato para las cadenas que voy a visualizar
StringFormat stringFormat = new StringFormat();
stringFormat.LineAlignment = StringAlignment.Center;
stringFormat.Alignment = StringAlignment.Center;

/// Región según el sexo que sea
Region reg = new Region();

/// Región para hacer uniones
Region tempRegion = new Region();
```

```
/// Rectángulo para los cálculos.
Rectangle temp;

/// Punto medio del área de dibujo
int medio = (int)(e.Bounds.Height/2);

/// Puntos para el de sexo desconocido
Point[] puntos = {new Point(e.Bounds.X,e.Bounds.Y + medio), new
Point(e.Bounds.X + medio,e.Bounds.Y), new Point(e.Bounds.Height,e.Bounds.Y +
medio), new Point(e.Bounds.X + medio,e.Bounds.Y+ e.Bounds.Height), new
Point(e.Bounds.X,e.Bounds.Y + medio)};

/// Para la región de femenino y de desconocido
GraphicsPath path = new GraphicsPath();
#endregion

e.DrawBackground();

switch(sexoDeSimbolos)
{
    case Sexos.Femenino:
     e.Graphics.DrawEllipse(Pens.Black,e.Bounds.X+1,e.Bounds.Y+1,
     e.Bounds.Height-1, e.Bounds.Height-2);
     path.AddEllipse(e.Bounds.X+1,e.Bounds.Y+1, e.Bounds.Height-1,
     e.Bounds.Height-2);
     reg = new Region(path);
    break;
    case Sexos.Masculino:
     Rectangle rect = new Rectangle(e.Bounds.X+1,e.Bounds.Y+1,
     e.Bounds.Height-1, e.Bounds.Height-2);
     reg = new Region(rect);
    break;

    case Sexos.Desconocido:
     path.AddLines(puntos);
     e.Graphics.DrawLines(Pens.Black, puntos);
     reg = new Region(path);
    break;
}
/// Dibujo la regiones del color del símbolo
Simbolo simbolo = (Simbolo)(Items[e.Index]);
switch (simbolo.TipoDeSimbolo)
{
    case TiposDeSimbolo.NoAfectado:
     e.Graphics.DrawString("  ", e.Font, Brushes.Black, rectString,
     stringFormat);
    break;
```

Aquí se dibuja cada tipo de símbolo según las reglas de cómo debe representarse cada uno.

```
}

/// Dibujando la descripción.

Rectangle rectcadena = e.Bounds;
rectcadena.X = e.Bounds.Height+20;
StringFormat strFormat = new StringFormat();
strFormat.LineAlignment = StringAlignment.Center;
e.Graphics.DrawString(((Simbolo)Items[e.Index]).Descripción, e.Font,
Brushes.Black, rectcadena, strFormat);

if (simbolo.GetType() == typeof(SimboloAdicional))
{

strFormat.LineAlignment = StringAlignment.Near;
Font a = new Font(e.Font.FontFamily, (int)(Height/2), e.Font.Style);
Rectangle cara = new Rectangle(new Point(e.Bounds.X +
e.Bounds.Height,e.Bounds.Y), new Size(20, (int)(e.Bounds.Height/2 + 3)));
rectcadena.X = Height;
e.Graphics.DrawString(((SimboloAdicional)Items[e.Index]).Caracterizacion ,
e.Font, Brushes.Black, cara, strFormat);

}
e.DrawFocusRectangle();

}

}
```

En la *VentanaRedefinirSimbolo* se adiciona el *VisorDeSimbolo* como un control y mediante este es que se interactúa con el usuario para toda la gestión de los símbolos. También se utiliza a la hora de escoger el símbolo del individuo en el *PropertyGrid* que representa las propiedades de una persona.

A nivel de usuario el control queda de la siguiente manera.

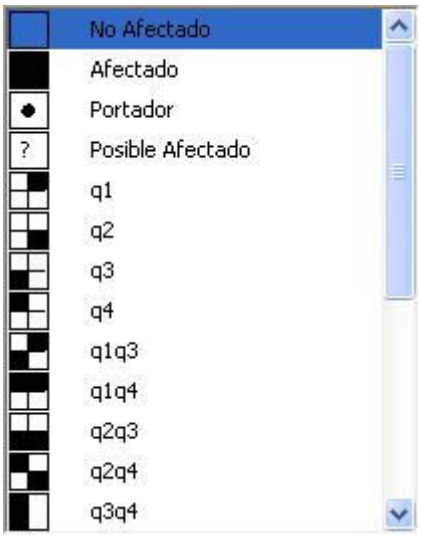


Fig. 24 Visor de Símbolos

4.2.2 Método *GuardarFamilia* y propuesta de patrón para guardar Árboles Genealógicos

Un problema que existente con este tipo de aplicaciones que se dedican a representar árboles genealógicos en el mundo, es que ninguna sigue un patrón para que el archivo guardado de un árbol genealógico pueda ser cargado por cualquier otra aplicación que se dedique a este propósito. Es por eso que se decide proponer un patrón para guardar ficheros de este tipo. Se propone que la extensión de estos sea de tipo (.abg). A continuación se explica la responsabilidad de guardar como se implementó en la aplicación.

En la aplicación la responsabilidad de guardar está en *FamiliaVisual* que es la que tiene el acceso a todos los controles representados. Esta responsabilidad se desarrolla serializando cada propiedad de las personas. En el caso de las listas de persona con las que se relaciona (Relaciones, Hijos, entre otras) el algoritmo a seguir indica que primero se serializa la cantidad de la lista aunque no tenga ningún elemento y esto sirve de referencia para saber lo que sigue en el fichero. En el caso de relaciones con personas lo que se guarda es el identificador correspondiente.

```
public void GuardarFamilia(Stream s)
{
BinaryFormatter b = new BinaryFormatter();
IEnumerator enumComponentes=Controls.GetEnumerator();

b.Serialize(s,Familia.ListaDeGeneraciones);
b.Serialize(s,Familia.Personas.Count);
```

```
while(enumComponentes.MoveNext())
{
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Location);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Sexo);

if((enumComponentes.Current as PersonaVisual).Persona is Desconocido)
b.Serialize(s,((enumComponentes.Current as PersonaVisual).Persona as
Desconocido).CantidadDelGrupo);

if((enumComponentes.Current as PersonaVisual).Persona is Femenino)
{
b.Serialize(s,((enumComponentes.Current as PersonaVisual).Persona as
Mujer).Embarazada);

if(((enumComponentes.Current as PersonaVisual).Persona as Mujer) is
Embarazada)
{
b.Serialize(s,((enumComponentes.Current as PersonaVisual).Persona as
MujerEmbarazada).Tiempo_de_embarazo);
b.Serialize(s,((enumComponentes.Current as PersonaVisual).Persona as
MujerEmbarazada).Tipo_de_embarazo);
}
}

b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Identificador);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Cantidad_de_hijos);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Cantidad_de_muestras);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Cantidad_de_relaciones);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Color_de_piel);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Comentario);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Dirección);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Edad);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Fecha_de_muerte);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Fecha_de_nacimiento);
int cantGemelos = (enumComponentes.Current as
PersonaVisual).Persona.Gemelos.Count;
b.Serialize(s,cantGemelos);

if(cantGemelos!=0)
{
```

```
IEnumerator enumGemelos=(enumComponentes.Current as
PersonaVisual).Persona.Gemelos.GetEnumerator();
while(enumGemelos.MoveNext())
{
b.Serialize(s,(enumGemelos.Current as Persona).Identificador);
}
}
int cantHijos=(enumComponentes.Current as
PersonaVisual).Persona.Hijos.Count;
b.Serialize(s,cantHijos);

if(cantHijos!=0)
{
IEnumerator enumHijos=(enumComponentes.Current as
PersonaVisual).Persona.Hijos.GetEnumerator();
while(enumHijos.MoveNext())
{
b.Serialize(s,(enumHijos.Current as Persona).Identificador);
}
}
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Intento_de_suicidio);

if((enumComponentes.Current as PersonaVisual).Persona.Madre!=null)
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Madre.Identificador);
else
b.Serialize(s,"null");

b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Muestras);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Nombre);
if((enumComponentes.Current as PersonaVisual).Persona.Padre!=null)
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Padre.Identificador);
else
b.Serialize(s,"null");

b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Primer_apellido);
int cantRelaciones=(enumComponentes.Current as
PersonaVisual).Persona.Relaciones.Count;
b.Serialize(s,cantRelaciones);

if(cantRelaciones!=0)
{
IEnumerator enumRelaciones=(enumComponentes.Current as
PersonaVisual).Persona.Relaciones.GetEnumerator();
while(enumRelaciones.MoveNext())
{
b.Serialize(s,(enumRelaciones.Current as Persona).Identificador);
```

```
}
}
int cantDivorcios=(enumComponentes.Current as
PersonaVisual).Persona.Divorcios.Count;
b.Serialize(s,cantDivorcios);

if(cantDivorcios!=0)
{
IEnumerator enumDivorcios=(enumComponentes.Current as
PersonaVisual).Persona.Divorcios.GetEnumerator();
while(enumDivorcios.MoveNext())
{
b.Serialize(s,(enumDivorcios.Current as Persona).Identificador);
}
}
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Segundo_apellido);

if((enumComponentes.Current as PersonaVisual).Persona.Símbolo==null)
b.Serialize(s,new Simbolo(TiposDeSimbolo.NoAfectado));
else
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Símbolo);

if((enumComponentes.Current as
PersonaVisual).Persona.Símbolo_adicional==null)
b.Serialize(s,new SimboloAdicional());
else
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Símbolo_adicional);

b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Propósito);
b.Serialize(s,(enumComponentes.Current as
PersonaVisual).Persona.Generacion);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Numero);
b.Serialize(s,(enumComponentes.Current as PersonaVisual).Persona.Tipo_Gem);

}
}
```

Esta forma de guardar que se utiliza mediante la serialización tiene como desventaja que los ficheros generados solo pueden leerse deserializando. Esta forma de guardar es muy segura por su codificación, pero no sirve para que otras aplicaciones de este tipo puedan cargar estos ficheros. Actualmente este problema de estandarizar los ficheros generados con cualquier aplicación dedicada a representar árboles genealógicos existe por no estar definido un formato o un patrón de cuál es la estructura a seguir para leer los datos de un fichero que contenga la información de un árbol

genealógico guardado. Por todo lo expresado anteriormente se propone que sean ficheros de xml o textos planos para guardar ficheros de estas aplicaciones.

Propuesta de patrón para guardar un árbol genealógico

Familia	
Nombre	
Identificador	
Enfermedad	
Fecha de creación	
Última modificación	
Dibujado por	

Persona		
Nombre		
Primer apellido		
Segundo apellido		
Identificador		
Generación		
Dirección		
Comentario		
Símbolo		
Símbolo adicional		
Fecha de nacimiento		
Fecha de muerte		
Madre		
Padre		
Lista de hijos		
Gemelos		
Sexo		
Color de la piel		
Tipo de gemelo		
Lista de relaciones		
Lista de Muestras		
Propósito		
Lista de divorcios		
Intento de suicidio		
Número		

4.3 VALIDACIÓN DEL SISTEMA

Las pruebas de la aplicación fueron desarrolladas por la Dirección de Calidad en la UCI. Al finalizar el proceso de prueba quedó liberada la aplicación. En el Anexo 9 se muestra el Acta de Liberación del producto, emitido por dicha dirección, que certifica que el mismo cumple con los estándares de calidad aprobados por la UCI para el desarrollo de este tipo de aplicaciones. También fueron liberados los manuales de usuario y de instalación de la aplicación.

4.4 ANÁLISIS DE LA IMPLEMENTACIÓN

En la implementación de la aplicación se detectaron algunos vestigios del diseño que se pueden perfeccionar teniendo en cuenta el uso de patrones de diseño. Un ejemplo de lo antes expuesto se evidencia en las acciones a realizar, estas se definen como un enumerador, lo que conlleva que al chequear la acción a ejecutar en la implementación, se desarrolle un método con sentencias condicionales como se muestra a continuación en la responsabilidad *GestionarRelacionesPersona* de la clase *Familia*.

```
public void GestionarRelacionesPersona(Acciones accion, PersonaVisual
personaEnAccion, PersonaVisual personaTemp)
{
      switch (accion)
      {
            case Acciones.EliminarRelacion:

                EliminarRelacion(personaTemp.Persona,personaEnAccion.Persona
          );

          break;

            case Acciones.Divorciar:

                 if(VerificarCasados(personaTemp.Persona,
personaEnAccion.Persona))
                       Divorciar(personaTemp.Persona,
personaEnAccion.Persona);

                else
                 {
                     MessageBox.Show("Ellos no mantienen relación",
                     "Error", MessageBoxButtons.OK,MessageBoxIcon.Error);
                       personaTemp.VolverPosicionInicial();
                 }
           break;
```

```
      //Aquí se repiten los case en correspondencia a las acciones
definidas.
      }
}
```

Una desventaja de lo anteriormente expuesto tiene lugar a la hora de definir alguna nueva acción, actualmente la solución es insertar nuevas condiciones a chequear, teniendo el programador que entender el código ya existente para poder modificarlo.

Este problema se puede solucionar definiendo en el diseño dicho enumerador como una jerarquía de clases que simulen las acciones y de esta manera se puede aplicar el patrón de diseño Polimorfismo, lo que permitiría que a la hora de la implementación, cuando se chequean las acciones se pueda invocar a estas responsabilidades polimórficas y se evite el uso de sentencias condicionales como las actualmente existentes. Además si en algún momento hay que adicionar una nueva acción, solo tendría que agregarse una clase que simule esta acción acorde a su comportamiento, heredando del padre de las acciones

Otro aspecto que se puede mejorar en el diseño es la representación de las personas en la familia y la relación de la clase persona con objetos de su mismo tipo, ejemplo: lista de hijos, lista de relaciones, que lo que guardan son las referencia a estos objetos. Esta relaciones, teniendo en cuenta que una persona puede estar relacionada con más de una persona a la vez hacen referencia a la misma persona por distintos tipos de relaciones. Una solución a esto sería, simular la lista de personas como un grafo en el diseño, donde los nodos son las personas y las aristas, los tipos de relaciones. Esto permite tener más de una arista entre nodos para representar varias relaciones entre dos individuos. Con un diseño de esta manera se garantiza además utilizar mecanismos de búsquedas que pueden ser de mucha utilidad en versiones superiores, los cuales de la forma diseñada sería muy engorroso.

Vale destacar que como está actualmente implementada la aplicación no afecta las funcionalidades definidas para el sistema en estos momentos, pudiendo influir a la hora de agregarle nuevas funcionalidades a la aplicación en futuras versiones. Teniendo en cuenta las exigencias de los diseños de clases actualmente. La metodología de desarrollo utilizada, RUP que entre sus principales ventajas se encuentra el ser iterativo e incremental, se propone que para una segunda versión de ésta aplicación se tengan en cuenta estos elementos del diseño a la hora de realizarlo o refinar el mismo.

Conclusiones

Como resultado de este capítulo se obtuvo el diagrama de componente, se explicaron algunas soluciones a nivel de código. Se propone además un patrón para guardar los archivos de árboles genealógicos y que posteriormente puedan ser cargados por cualquier aplicación de las que se dedican a representar árboles genealógicos. También se hace referencia al acta que valida la liberación del producto por la Dirección de Calidad en la UCI y se realiza un análisis de la solución obtenida donde se recomiendan algunos elementos a tener en cuenta para una segunda versión.

CONCLUSIONES

La especificación de los requisitos funcionales posibilitó la definición de las funcionalidades de la aplicación informática para la representación de árboles genealógicos.

Se realizó el diseño de las clases de la aplicación a partir del uso de los patrones de diseño y arquitectura.

Se implementaron las clases del diseño definidas y se obtuvo una aplicación informática que permite la representación de los árboles genealógicos. Además de la aplicación, se elaboró el manual de instalación y el manual de usuario de la misma, elementos que fueron liberados por la Dirección de Calidad en la UCI.

Como resultado de esta investigación se obtuvo alasARBOGEN. La aplicación se encuentra desplegada en la red nacional de centros de genética médica de Cuba y en el CNGM.

RECOMENDACIONES

Se recomienda realizar una segunda versión de la aplicación teniendo en cuentas los aspectos señalados en esta versión, que sea multiplataforma y pueda hacer uso de una base de dato para gestionar la información asociada a los árboles genealógicos.

Se recomienda integrar esta aplicación al Sistema Informático de Genética Médica (SIGMédica) en proceso de desarrollo.

Se recomienda hacer uso de la propuesta de estandarizar del formato de los archivos para almacenar los datos correspondientes a un árbol genealógico.

REFERENCIAS BIBLIOGRÁFICAS

1. babylon. Árbol Genealógico. [En línea] [Citado el: 8 de Diciembre de 2007.] Disponible en: http://www.babylon.com/definition/%C3%A1rbol_geneal%C3%B3gico/Spanish

2. Atención Primaria. [En línea] [Citado el: 10 de Diciembre de 2007.] Disponible en: http://atencionprimaria.wordpress.com/2007/11/11/genogramas.

3. Salud Familiar. [En línea] 11 de noviembre de 2007 [Citado el: 8 de Diciembre de 2007.] Disponible en: http://cambiodemodelo.blogspot.com/2007/11/genograma.html.

4. ASHEGUI. [En línea] [Citado el: 9 de Diciembre de 2007.] Disponible en: http://www.hemofiliaguipuzcoa.org/genetica/listar.asp

5. CubaGenWeb. Centro de la Genealogía Cubana . [En línea] [Citado el: 9 de Diciembre de 2007.] Disponible en: http://www.cubagenweb.org/e-dna.htm.

6. PC WORLD digital. BitGen II. Árboles genealógicos y heráldica desde el PC. [En línea] [Citado el: 11 de Diciembre de 2007.] Disponible en: http://www.idg.es/pcworld/BitGen_II._Árboles_genealógicos_y_heráldica_desde_/art131220.htm.

7. GDS. Sistema General de Documentación familiar. [En línea] [Citado el: 11 de Diciembre de 2007.] Disponible en: http://www.gdsystem.net/v5/es/es_gds_quees.asp.

8. OnlyMacSoftware. GenealogíaMac. [En línea] [Citado el: 11 de Diciembre de 2007.] Disponible en: http://www.onlymac.de/html/stammbaum4es.html.

9. Softwarecientifico. Cyrillic. [En línea] [Citado el: 12 de Diciembre de 2007.] Disponible en: http://www.softwarecientifico.com/paginas/cyrillic.htm.

10. eduangi telecom [En línea] [Citado el: 15 de Enero de 2008.] Disponible en: http://web.madritel.es/personales3/edcollado/ingsw/tema2/2-4.htm

11. Sánchez Mendoza María A.. Informatizate. Metodologías De Desarrollo De Software. [En línea] [Citado el: 15 de 1 de 2008.] Disponible en: http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

12. Guerrero Luis A. Universidad de Chile [En línea] [Citado el: 15 de 1 de 2008.] Disponible en: http://www.eici.ucm.cl/Academicos/R_Villarroel/descargas/ing_sw_1/RUP.pdf

13. Introducción Ingeniería de Software [En línea] Noviembre de 2007[Citado el: 15 de 1 de 2008.] Disponible en: http://eproano334.blogspot.es/tags/Vida/

14. . http://web.usal.es/~mlperez/programacion_archivos/trabajos_2006_2007/Tema2_GR2.pdf

15. Marcianos.com [En línea] [Citado el: 17 de enero de 2008.] Disponible en: http://www.marcianos.com/enc/lenguaje_de_programacion_pascal.html

16. Almendras Vargas Wilfredo Ajayu [En línea] septiembre de 2007 [Citado el: 20 de Enero de 2008.] Disponible en: http://ajayu.memi.umss.edu.bo/wilfredo/weblog/patrones-de-diseno

17. Centro Nacional de Genética Médica. [En línea] [Citado el: 8 de Febrero de 2008.] Disponible en: http://www.sld.cu/sitios/genetica/

18. Vilas Fernández Ana [En línea] marzo de 2001 [Citado el: 20 de Abril de 2008.] Disponible en: http://www-gris.det.uvigo.es/~avilas/UML/node42.html

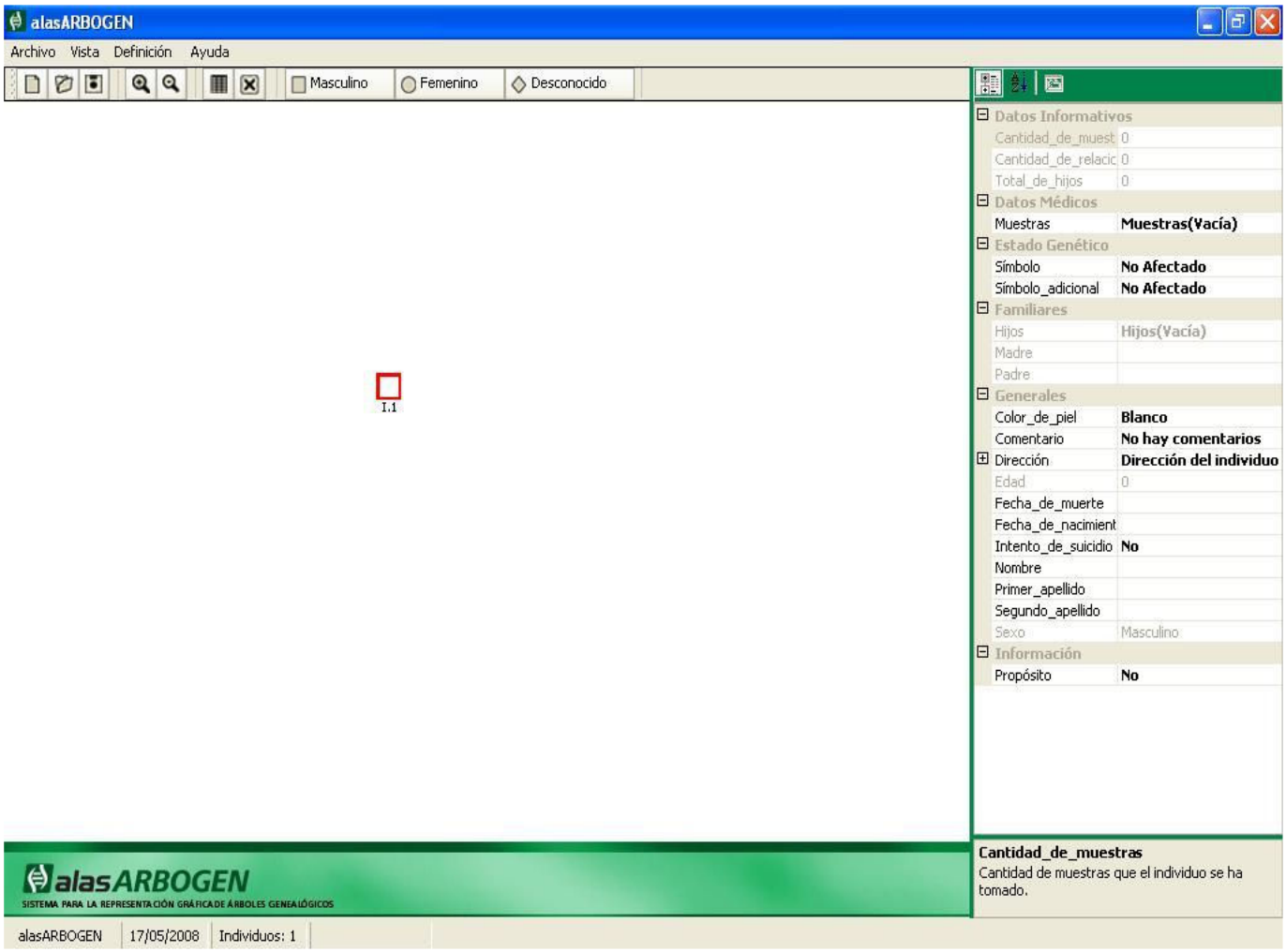
BIBLIOGRAFÍA

1. Jacobson, Ivar; Booch, Grady y Rumbaugh, James: El Proceso Unificado de Desarrollo Volumen I,The Addison Wesley Longman Inc., 1999.
2. Larman, Craig: UML y patrones, introducción al análisis y diseño orientado a objetos, Félix Varela,
   2004.
3. El Arte de Modelar [En línea] [Citado el: 12 de Febrero de 2008.] Disponible en: http://dc.exa.unrc.edu.ar/nuevodc/m baterias/sistemas/2007/TEORICOS/TEORIA_2_UML_Intro_DC_2007.pdf
4. R. Villarroel, E. Fernández-Medina, J. Trujillo, M. Piattini. Un Profile de UML para Diseñar Almacenes de Datos Seguros [En línea] [Citado el: 13 de Febrero de 2008.] Disponible en: http://ieeexplore.ieee.org/iel5/9907/31504/01468661.pdf?tp=&isnumber=&arnumber=1468661
5. Seco González José Antonio. PROGRAMMATIUM [En línea] [Citado el: 4 de Marzo de 2008.] Disponible en: http://programmatium.blogspot.com/2008/01/el-lenguaje-de-programacin-c_23.html
6. La Revista Informática.com [En línea] [Citado el: 6 de Marzo de 2008.] Disponible en: http://www.larevistainformatica.com/C1.htm
7. Gracia Joaquín IngenieroSoftware [En línea] Mayo de 2005 [Citado el: 9 de Abril de 2008.] Disponible en:http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php
8. Larman. UML y Patrones: Introducción al análisis y programación orientada a objetos [En línea] C. / México, Prentice Hall, 1999 [Citado el: 11 de Marzo de 2008.] Disponible en: http://bibliodoc.uci.cu/pdf/reg00062.pdf
9. Pressman, Roger S. / Madrid, McGraw-Hill. Ingeniería del Software: un enfoque práctico. Parte I y II [En línea] 2002 [Citado el: 11 de Marzo de 2008.] Disponible en: http://bibliodoc.uci.cu/pdf/reg02689.pdf
10. http://www.info-ab.uclm.es/asignaturas/42530/pdf/M2tema12.pdf
11. Ayuda extendida del Rational Rose Enterprise Edition 2003.

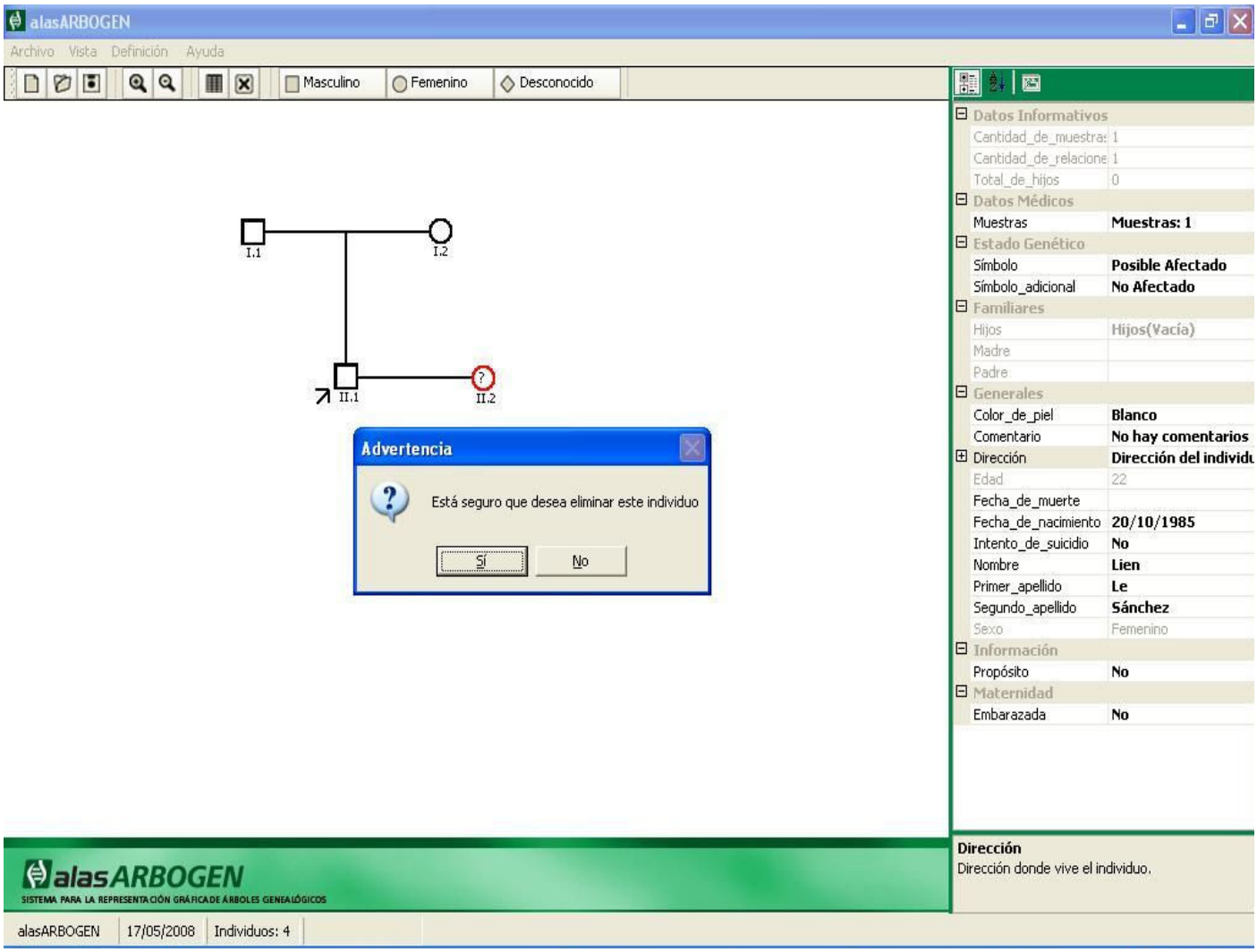
ANEXOS

Anexo 1. Prototipos no funcionales del CUS Gestionar gráficamente un individuo.

Anexo 1.1. Insertar un individuo, Modificar datos de un individuo, Mostrar datos de un individuo.



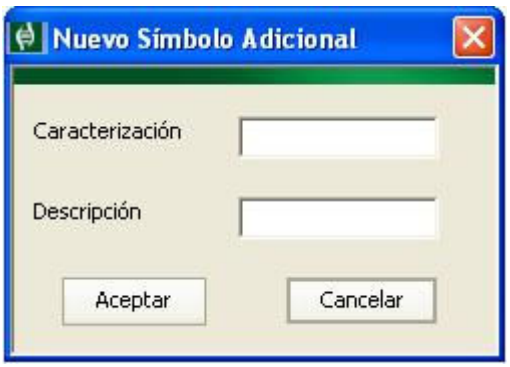
Anexo 1.2. Eliminar un individuo



Anexo 2. Prototipos no funcionales del CUS Gestionar relaciones entre individuos

Anexo 3. Prototipos no funcionales del CUS Gestionar símbolo

Anexo 3.1 Crear nuevo símbolo

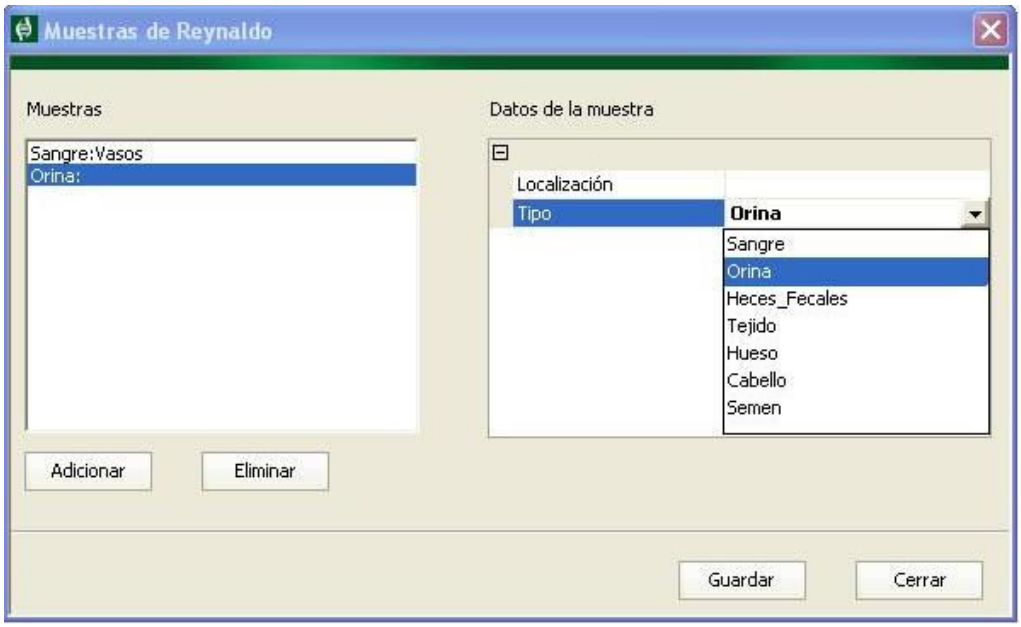


Anexo 3.2 Modificar símbolo

Anexo 3.3 Eliminar un símbolo

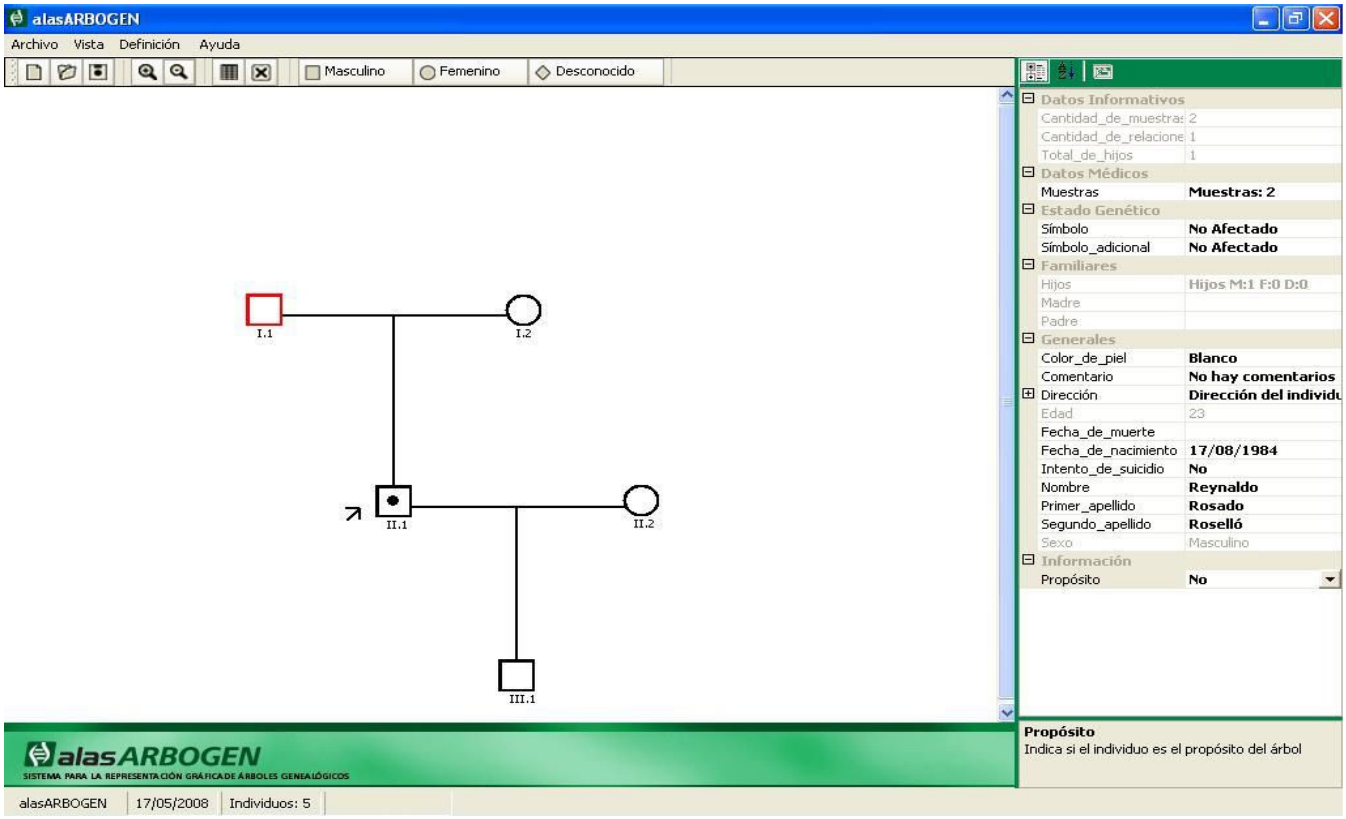
Anexo 4. Prototipos no funcionales del CUS Gestionar enfermedad

Anexo 5. Prototipos no funcionales del CUS Gestionar muestra

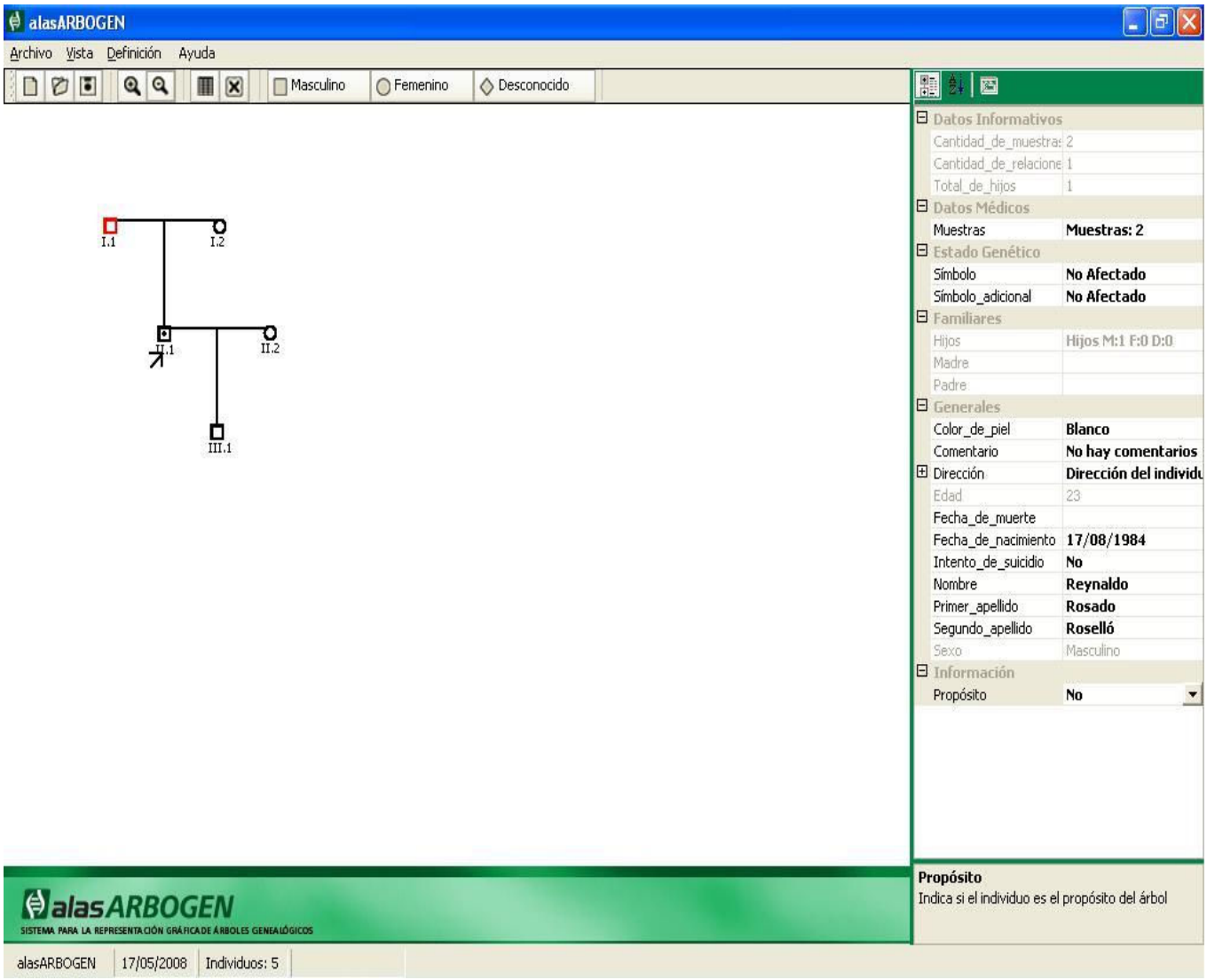


Anexo 6. Prototipos no funcionales del CUS Gestionar dimensiones del árbol

Anexo 6.1. Aumentar tamaño de los individuos

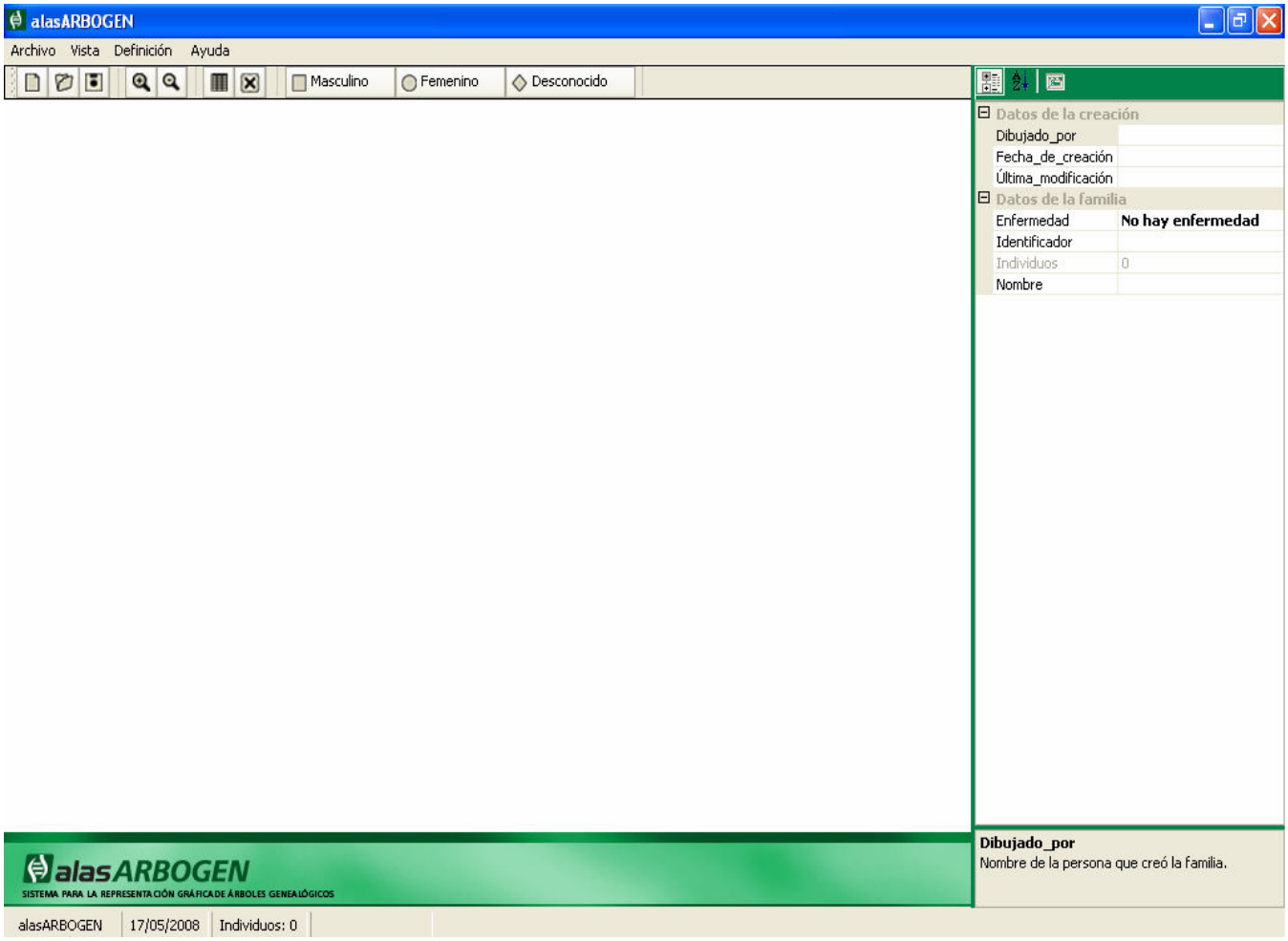


Anexo 6.2. Disminuir tamaño de los individuos

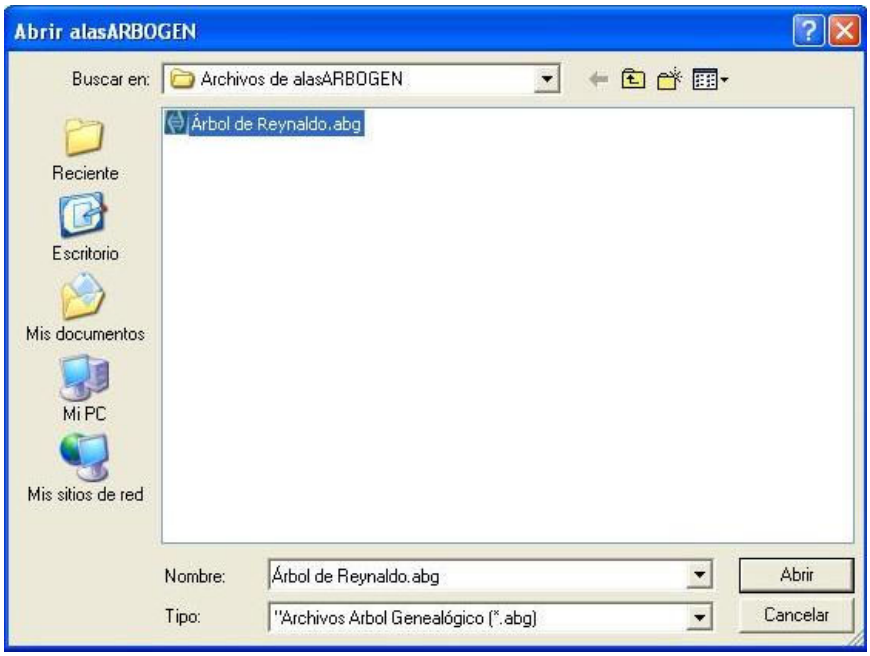


Anexo 7. Prototipos no funcionales del CUS Gestionar familia

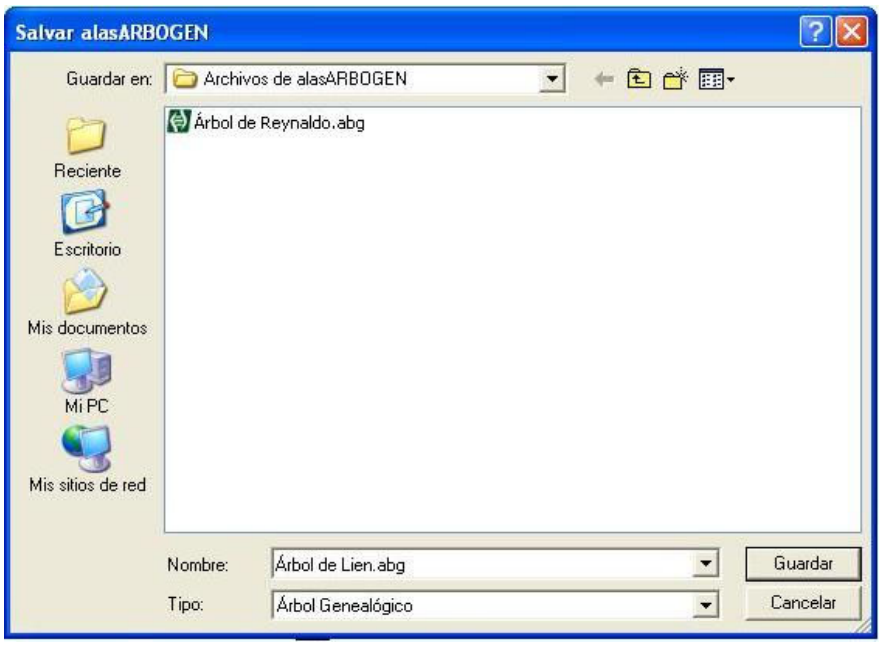
Anexo 7.1 Crear una nueva familia, Modificar datos de una familia



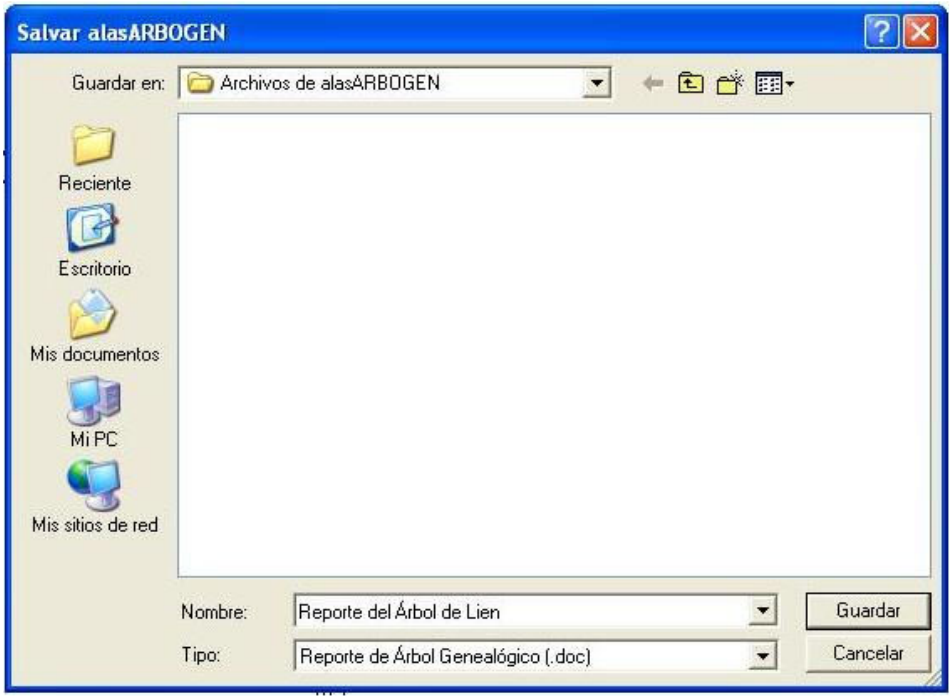
Anexo 7.2 Buscar una familia



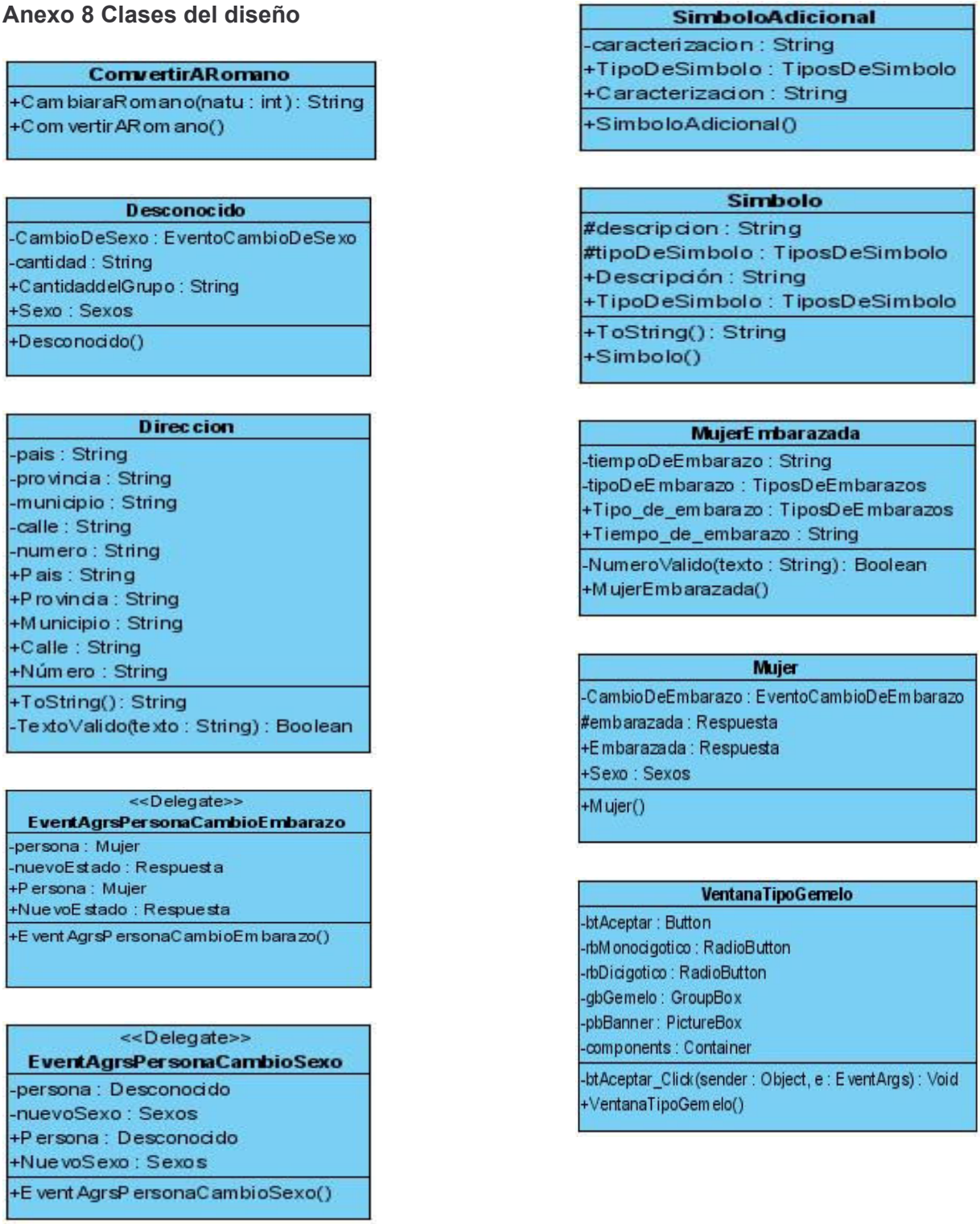
Anexo 7.3 Guardar familia

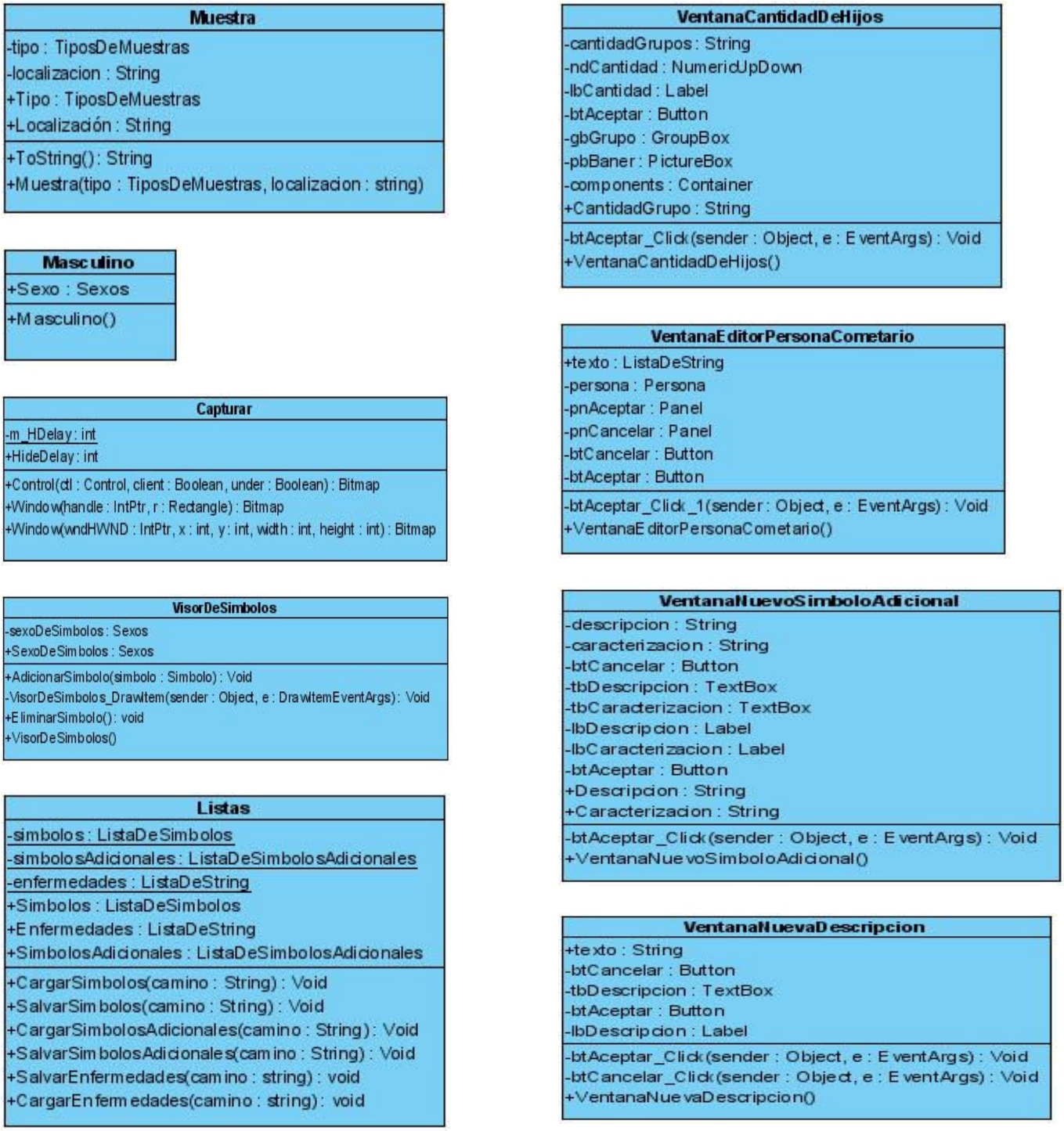


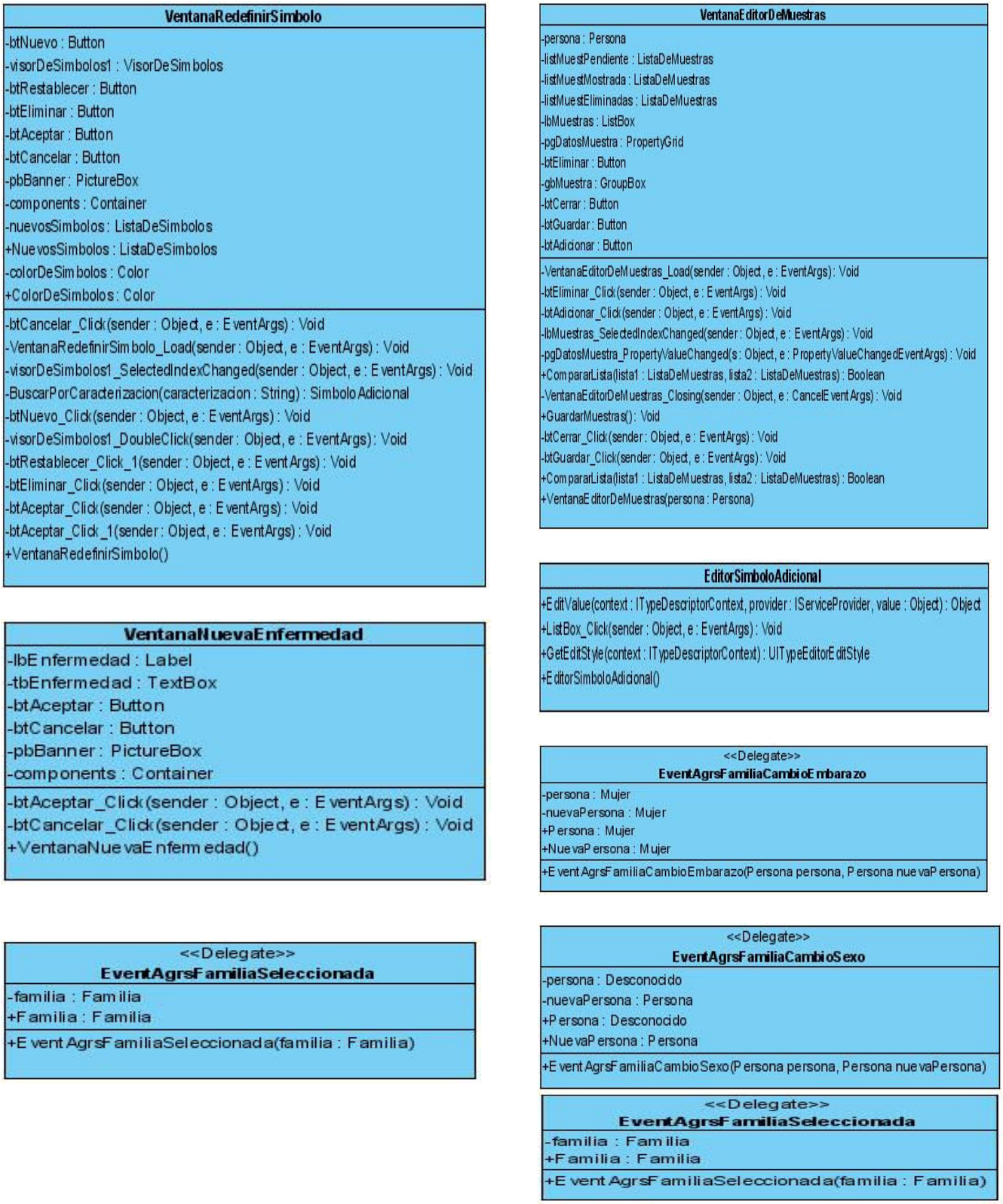
Anexo 7.4 Exportar familia

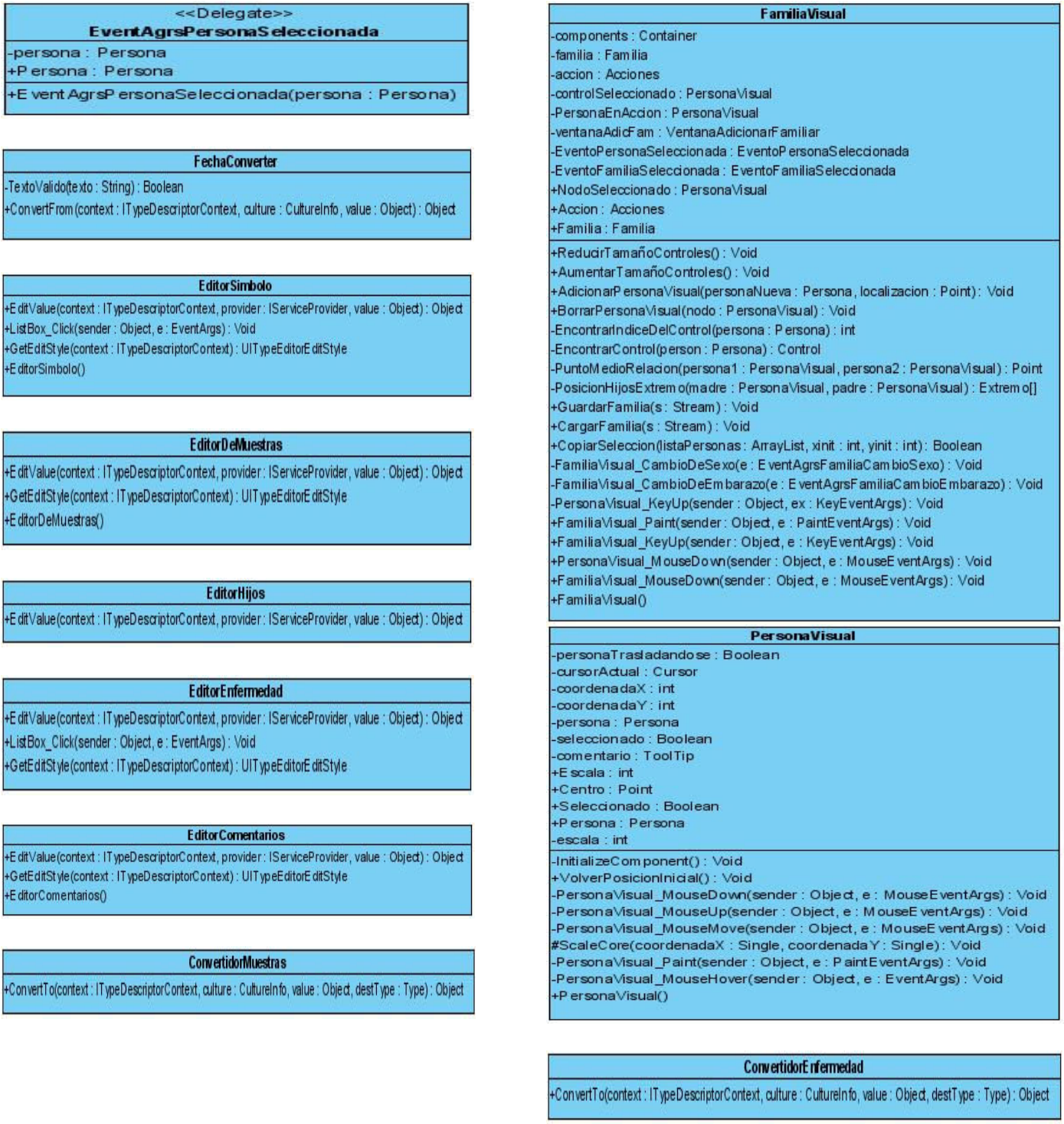


Anexo 8 Clases del diseño

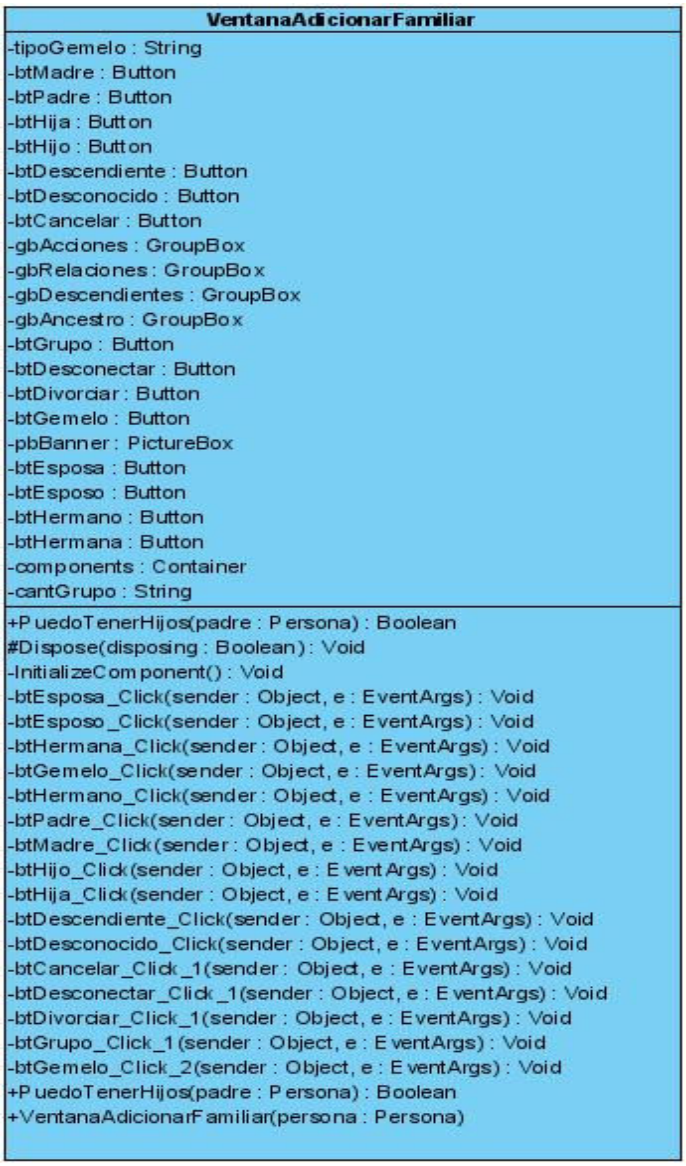



Muestra
-tipo : TiposDeMuestras
-localizacion : String
+Tipo : TiposDeMuestras
+Localización : String
+ToString() : String
+Muestra(tipo : TiposDeMuestras, localizacion : string)
Masculino
+Sexo : Sexos
+Masculino()
Capturar
-m_HDelay : int
+HideDelay : int
+Control(ctl : Control, client : Boolean, under : Boolean) : Bitmap
+Window(handle : IntPtr, r : Rectangle) : Bitmap
+Window(wndHWND : IntPtr, x : int, y : int, width : int, height : int) : Bitmap
VisorDeSimbolos
-sexoDeSimbolos : Sexos
+SexoDeSimbolos : Sexos
+AdicionarSimbolo(simbolo : Simbolo) : Void
-VisorDeSimbolos_DrawItem(sender : Object, e : DrawItemEventArgs) : Void
+EliminarSimbolo() : void
+VisorDeSimbolos()
Listas
-simbolos : ListaDeSimbolos
-simbolosAdicionales : ListaDeSimbolosAdicionales
-enfermedades : ListaDeString
+Simbolos : ListaDeSimbolos
+Enfermedades : ListaDeString
+SimbolosAdicionales : ListaDeSimbolosAdicionales
+CargarSimbolos(camino : String) : Void
+SalvarSimbolos(camino : String) : Void
+CargarSimbolosAdicionales(camino : String) : Void
+SalvarSimbolosAdicionales(camino : String) : Void
+SalvarEnfermedades(camino : string) : void
+CargarEnfermedades(camino : string) : void
VentanaCantidadDeHijos
-cantidadGrupos : String
-ndCantidad : NumericUpDown
-lbCantidad : Label
-btAceptar : Button
-gbGrupo : GroupBox
-pbBaner : PictureBox
-components : Container
+CantidadGrupo : String
-btAceptar_Click(sender : Object, e : EventArgs) : Void
+VentanaCantidadDeHijos()
VentanaEditorPersonaCometario
+texto : ListaDeString
-persona : Persona
-pnAceptar : Panel
-pnCancelar : Panel
-btCancelar : Button
-btAceptar : Button
-btAceptar_Click_1(sender : Object, e : EventArgs) : Void
+VentanaEditorPersonaCometario()
VentanaNuevoSimboloAdicional
-descripcion : String
-caracterizacion : String
-btCancelar : Button
-tbDescripcion : TextBox
-tbCaracterizacion : TextBox
-lbDescripcion : Label
-lbCaracterizacion : Label
-btAceptar : Button
+Descripcion : String
+Caracterizacion : String
-btAceptar_Click(sender : Object, e : EventArgs) : Void
+VentanaNuevoSimboloAdicional()
VentanaNuevaDescripcion
+texto : String
-btCancelar : Button
-tbDescripcion : TextBox
-btAceptar : Button
-lbDescripcion : Label
-btAceptar_Click(sender : Object, e : EventArgs) : Void
-btCancelar_Click(sender : Object, e : EventArgs) : Void
+VentanaNuevaDescripcion()


VentanaRedefinirSimbolo
-btNuevo : Button
-visorDeSimbolos1 : VisorDeSimbolos
-btRestablecer : Button
-btEliminar : Button
-btAceptar : Button
-btCancelar : Button
-pbBanner : PictureBox
-components : Container
-nuevosSimbolos : ListaDeSimbolos
+NuevosSimbolos : ListaDeSimbolos
-colorDeSimbolos : Color
+ColorDeSimbolos : Color
-btCancelar_Click(sender : Object, e : EventArgs) : Void
-VentanaRedefinirSimbolo_Load(sender : Object, e : EventArgs) : Void
-visorDeSimbolos1_SelectedIndexChanged(sender : Object, e : EventArgs) : Void
-BuscarPorCaracterizacion(caracterizacion : String) : SimboloAdicional
-btNuevo_Click(sender : Object, e : EventArgs) : Void
-visorDeSimbolos1_DoubleClick(sender : Object, e : EventArgs) : Void
-btRestablecer_Click_1(sender : Object, e : EventArgs) : Void
-btEliminar_Click(sender : Object, e : EventArgs) : Void
-btAceptar_Click(sender : Object, e : EventArgs) : Void
-btAceptar_Click_1(sender : Object, e : EventArgs) : Void
+VentanaRedefinirSimbolo()
VentanaEditorDeMuestras
-persona : Persona
-listMuestPendiente : ListaDeMuestras
-listMuestMostrada : ListaDeMuestras
-listMuestEliminadas : ListaDeMuestras
-lbMuestras : ListBox
-pgDatosMuestra : PropertyGrid
-btEliminar : Button
-gbMuestra : GroupBox
-btCerrar : Button
-btGuardar : Button
-btAdicionar : Button
-VentanaEditorDeMuestras_Load(sender : Object, e : EventArgs) : Void
-btEliminar_Click(sender : Object, e : EventArgs) : Void
-btAdicionar_Click(sender : Object, e : EventArgs) : Void
-lbMuestras_SelectedIndexChanged(sender : Object, e : EventArgs) : Void
-pgDatosMuestra_PropertyValueChanged(s : Object, e : PropertyValueChangedEventArgs) : Void
+CompararLista(lista1 : ListaDeMuestras, lista2 : ListaDeMuestras) : Boolean
-VentanaEditorDeMuestras_Closing(sender : Object, e : CancelEventArgs) : Void
+GuardarMuestras() : Void
-btCerrar_Click(sender : Object, e : EventArgs) : Void
-btGuardar_Click(sender : Object, e : EventArgs) : Void
+CompararLista(lista1 : ListaDeMuestras, lista2 : ListaDeMuestras) : Boolean
+VentanaEditorDeMuestras(persona : Persona)
VentanaNuevaEnfermedad
-lbEnfermedad : Label
-tbEnfermedad : TextBox
-btAceptar : Button
-btCancelar : Button
-pbBanner : PictureBox
-components : Container
-btAceptar_Click(sender : Object, e : EventArgs) : Void
-btCancelar_Click(sender : Object, e : EventArgs) : Void
+VentanaNuevaEnfermedad()
EditorSimboloAdicional
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
+ListBox_Click(sender : Object, e : EventArgs) : Void
+GetEditStyle(context : ITypeDescriptorContext) : UITypeEditorEditStyle
+EditorSimboloAdicional()
<<Delegate>>
EventAgrsFamiliaCambioEmbarazo
-persona : Mujer
-nuevaPersona : Mujer
+Persona : Mujer
+NuevaPersona : Mujer
+EventAgrsFamiliaCambioEmbarazo(Persona persona, Persona nuevaPersona)
<<Delegate>>
EventAgrsFamiliaSeleccionada
-familia : Familia
+Familia : Familia
+EventAgrsFamiliaSeleccionada(familia : Familia)
<<Delegate>>
EventAgrsFamiliaCambioSexo
-persona : Desconocido
-nuevaPersona : Persona
+Persona : Desconocido
+NuevaPersona : Persona
+EventAgrsFamiliaCambioSexo(Persona persona, Persona nuevaPersona)
<<Delegate>>
EventAgrsFamiliaSeleccionada
-familia : Familia
+Familia : Familia
+EventAgrsFamiliaSeleccionada(familia : Familia)


<<Delegate>>
EventAgrsPersonaSeleccionada
-persona : Persona
+Persona : Persona
+EventAgrsPersonaSeleccionada(persona : Persona)
FechaConverter
-TextoValido(texto : String) : Boolean
+ConvertFrom(context : ITypeDescriptorContext, culture : CultureInfo, value : Object) : Object
EditorSimbolo
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
+ListBox_Click(sender : Object, e : EventArgs) : Void
+GetEditStyle(context : ITypeDescriptorContext) : UITypeEditorEditStyle
+EditorSimbolo()
EditorDeMuestras
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
+GetEditStyle(context : ITypeDescriptorContext) : UITypeEditorEditStyle
+EditorDeMuestras()
EditorHijos
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
EditorEnfermedad
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
+ListBox_Click(sender : Object, e : EventArgs) : Void
+GetEditStyle(context : ITypeDescriptorContext) : UITypeEditorEditStyle
EditorComentarios
+EditValue(context : ITypeDescriptorContext, provider : IServiceProvider, value : Object) : Object
+GetEditStyle(context : ITypeDescriptorContext) : UITypeEditorEditStyle
+EditorComentarios()
ConvertidorMuestras
+ConvertTo(context : ITypeDescriptorContext, culture : CultureInfo, value : Object, destType : Type) : Object
FamiliaVisual
-components : Container
-familia : Familia
-accion : Acciones
-controlSeleccionado : PersonaVisual
-PersonaEnAccion : PersonaVisual
-ventanaAdicFam : VentanaAdicionarFamiliar
-EventoPersonaSeleccionada : EventoPersonaSeleccionada
-EventoFamiliaSeleccionada : EventoFamiliaSeleccionada
+NodoSeleccionado : PersonaVisual
+Accion : Acciones
+Familia : Familia
+ReducirTamañoControles() : Void
+AumentarTamañoControles() : Void
+AdicionarPersonaVisual(personaNueva : Persona, localizacion : Point) : Void
+BorrarPersonaVisual(nodo : PersonaVisual) : Void
-EncontrarIndiceDelControl(persona : Persona) : int
-EncontrarControl(person : Persona) : Control
-PuntoMedioRelacion(persona1 : PersonaVisual, persona2 : PersonaVisual) : Point
-PosicionHijosExtremo(madre : PersonaVisual, padre : PersonaVisual) : Extremo[]
+GuardarFamilia(s : Stream) : Void
+CargarFamilia(s : Stream) : Void
+CopiarSeleccion(listaPersonas : ArrayList, xinit : int, yinit : int) : Boolean
-FamiliaVisual_CambioDeSexo(e : EventAgrsFamiliaCambioSexo) : Void
-FamiliaVisual_CambioDeEmbarazo(e : EventAgrsFamiliaCambioEmbarazo) : Void
-PersonaVisual_KeyUp(sender : Object, ex : KeyEventArgs) : Void
+FamiliaVisual_Paint(sender : Object, e : PaintEventArgs) : Void
+FamiliaVisual_KeyUp(sender : Object, e : KeyEventArgs) : Void
+PersonaVisual_MouseDown(sender : Object, e : MouseEventArgs) : Void
+FamiliaVisual_MouseDown(sender : Object, e : MouseEventArgs) : Void
+FamiliaVisual()
PersonaVisual
-personaTrasladandose : Boolean
-cursorActual : Cursor
-coordenadaX : int
-coordenadaY : int
-persona : Persona
-seleccionado : Boolean
-comentario : ToolTip
+Escala : int
+Centro : Point
+Seleccionado : Boolean
+Persona : Persona
-escala : int
-InitializeComponent() : Void
+VolverPosicionInicial() : Void
-PersonaVisual_MouseDown(sender : Object, e : MouseEventArgs) : Void
-PersonaVisual_MouseUp(sender : Object, e : MouseEventArgs) : Void
-PersonaVisual_MouseMove(sender : Object, e : MouseEventArgs) : Void
#ScaleCore(coordenadaX : Single, coordenadaY : Single) : Void
-PersonaVisual_Paint(sender : Object, e : PaintEventArgs) : Void
-PersonaVisual_MouseHover(sender : Object, e : EventArgs) : Void
+PersonaVisual()
ConvertidorEnfermedad
+ConvertTo(context : ITypeDescriptorContext, culture : CultureInfo, value : Object, destType : Type) : Object


Persona
#intentoDeSuicidio : Respuesta
#identificador : String
#generacion : int
#numero : int
#nombre : String
#primerApellido : String
#segundoApellido : String
#direccion : Direccion
#comentario : ListaDeString
#simbolo : Simbolo
#simboloAdicional : SimboloAdicional
#fechaDeNacimiento : DateTime
#fechaDeMuerte : DateTime
#madre : Persona
#padre : Persona
#familia : Familia
#hijos : ListaDePersonas
#gemelos : ListaDePersonas
#sexo : Sexos
#colorDePiel : ColoresDePiel
#tipoGemelo : TypeGemelos
#relaciones : ListaDePersonas
#muestras : ListaDeMuestras
#proposito : Proposito
#listaDeDivorcios : ListaDePersonas
#grupo : Boolean
+Tipo_Gem : TipoGemelos
+Comentario : ListaDeString
+Intento_de_suicidio : Respuesta
+Color_de_piel : ColoresDePiel
+Sexo : Sexos
+Proposito : Respuesta
+Símbolo : Simbolo
+Símbolo_adicional : SimboloAdicional
+Dirección : Direccion
+Relaciones : ListaDePersonas
+Muerto : Boolean
+Numero : int
+Cantidad_de_muestras : int
+Muestras : ListaDeMuestras
+Cantidad_de_hijos : int
+Total_de_hijos : int
+Hijos : ListaDePersonas
+Divorcios : ListaDePersonas
+Familia : Familia
+Fecha_de_nacimiento : DateTime
+Fecha_de_muerte : DateTime
+Padre : Persona
+Madre : Persona
+Identificador : String
+Generacion : int
+Primer_apellido : String
+Segundo_apellido : String
+Edad : int
+Nombre : String
+Grupo : Boolean
+Gemelos : ListaDePersonas
+Cantidad_de_relaciones : int
+Persona()
+ValidarTexto(texto : String) : Boolean
+ToString() : String
+AdicionarHijo(hijo : Persona) : int
+AdicionarRelacion(person : Persona) : Void
+AdicionarGemelo(brother : Persona) : int
+BorrarHijo(hijo : Persona) : Void
+BorrarRelacion(person : Persona) : Void
+BorrarGemelo(gemelo : Persona) : Void
+BorrarPadre() : Void
+BorrarMadre() : Void
+CalcularEdad(inicial : DateTime, final : DateTime) : int
+PuedoTenerHijos() : boolean


VentanaAdicionarFamiliar
-tipoGemelo : String
-btMadre : Button
-btPadre : Button
-btHija : Button
-btHijo : Button
-btDescendiente : Button
-btDesconocido : Button
-btCancelar : Button
-gbAcciones : GroupBox
-gbRelaciones : GroupBox
-gbDescendientes : GroupBox
-gbAncestro : GroupBox
-btGrupo : Button
-btDesconectar : Button
-btDivorciar : Button
-btGemelo : Button
-pbBanner : PictureBox
-btEsposa : Button
-btEsposo : Button
-btHermano : Button
-btHermana : Button
-components : Container
-cantGrupo : String
+PuedoTenerHijos(padre : Persona) : Boolean
#Dispose(disposing : Boolean) : Void
-InitializeComponent() : Void
-btEsposa_Click(sender : Object, e : EventArgs) : Void
-btEsposo_Click(sender : Object, e : EventArgs) : Void
-btHermana_Click(sender : Object, e : EventArgs) : Void
-btGemelo_Click(sender : Object, e : EventArgs) : Void
-btHermano_Click(sender : Object, e : EventArgs) : Void
-btPadre_Click(sender : Object, e : EventArgs) : Void
-btMadre_Click(sender : Object, e : EventArgs) : Void
-btHijo_Click(sender : Object, e : EventArgs) : Void
-btHija_Click(sender : Object, e : EventArgs) : Void
-btDescendiente_Click(sender : Object, e : EventArgs) : Void
-btDesconocido_Click(sender : Object, e : EventArgs) : Void
-btCancelar_Click_1(sender : Object, e : EventArgs) : Void
-btDesconectar_Click_1(sender : Object, e : EventArgs) : Void
-btDivorciar_Click_1(sender : Object, e : EventArgs) : Void
-btGrupo_Click_1(sender : Object, e : EventArgs) : Void
-btGemelo_Click_2(sender : Object, e : EventArgs) : Void
+PuedoTenerHijos(padre : Persona) : Boolean
+VentanaAdicionarFamiliar(persona : Persona)


ConvertidorComentarios
+ConvertTo(context : ITypeDescriptorContext, culture : CultureInfo, value : Object, destType : Type) : Object


ConvertidorHijos
+ConvertTo(context : ITypeDescriptorContext, culture : CultureInfo, value : Object, destType : Type) : Object


Familia
-identificador : String
-nombre : String
-enfermedad : String
-fechaDeCreacion : DateTime
-ultimaModificacion : DateTime
-dibujadoPor : String
+personas : ListaDePersonas
-listaDeGeneraciones : ArrayList
-CambioDeEmbarazo : EventoPersonaCambioDeEmbarazo
-CambioDeSexo : EventoPersonaCambioDeSexo
+Personas : ListaDePersonas
+Dibujado_por : String
+Última_modificación : DateTime
+Nombre : String
+Enfermedad : String
+Individuos : int
+Identificador : String
+ListaDeGeneraciones : ArrayList
+Fecha_de_creación : DateTime
-ValidarTexto(texto : String) : Boolean
-ValidarNumero(texto : String) : Boolean
+EsAntecesorDe(antecesor : Persona, descendiente : Persona) : Boolean
+PuedeSerEsposoDe(esposa : Persona, esposo : Persona) : Boolean
+PuedeSerEsposaDe(esposo : Persona, esposa : Persona) : Boolean
+PuedeSerDescendienteDe(descendiente : Persona, antecesor : Persona) : Boolean
+PuedeSerHermanoDe(nuevoHermano : Persona, persona : Persona) : Boolean
+EsDescendienteDe(descendiente : Persona, antecesor : Persona) : Boolean
+AdicionarPersona(nuevaPersona : Persona) : Void
+BuscarPersona(identificador : String) : Persona
-BuscarPersona(person : Persona) : int
+AdicionarPadre(padre : Persona, hijo : Persona) : Void
+AdicionarMadre(madre : Persona, hijo : Persona) : Void
+PuedeSerMadreDe(madre : Persona, hijo : Persona) : Boolean
+PuedeSerPadreDe(padre : Persona, hijo : Persona) : Boolean
+EsHijoDe(hijo : Persona, padre : Persona, madre : Persona) : Boolean
+EsHijoDe(hijo : Persona, padres : Persona) : Boolean
+DesconectarDeLosPadres(persona : Persona) : Void
+RemoverPersona(persona : Persona) : Void
+TienenHijosComun(persona1 : Persona, persona2 : Persona) : Boolean
+SonHermanos(persona1 : Persona, persona2 : Persona) : String
+AdicionarHermano(nuevoHermano : Persona, persona : Persona) : Void
+AdicionarEsposo(mujer : Persona, esposo : Persona) : Void
+ConvertirDesconocidoA(persona : Desconocido, nuevaPersona : Persona) : Void
+CambiarSexo(e : EventAgrsPersonaCambioSexo) : Void
-PasarDatosMujerAMujer(destino : Mujer, fuente : Mujer) : Void
+CambiarEmbarazo(e : EventAgrsPersonaCambioEmbarazo) : Void
+ExportarReporteTexto(pfile : String) : Void
+BuscarSuGeneracion(Persona persona)
+GenerarNuevoIdentificador() : String
+Divorciar(persona1 : Persona, persona2 : Persona) : Void
+PuedeSerHijaDe(hija : Persona, progenitor : Persona) : Boolean
+PuedeSerHijoDe(hija : Persona, progenitor : Persona) : Boolean
-VerificarCasados(persona1 : Persona, persona2 : Persona) : Boolean
+EliminarRelacion(persona1 : Persona, persona2 : Persona) : Void
+GestinarRelacionesPersona(accion : Acciones, personaEnAccion : PersonaVisual, personaTemp : Perso...
+GestionarRelacionesFamilia(accion : Acciones, PersonaEnAccion : PersonaVisual) : Persona
+BorrarReferencias(controlSeleccionado : PersonaVisual) : void
+AnalizarFechasCompatibles(progenitor : Persona, descendiente : Persona) : boolean
+CalcularEdad(fechaInicial,DateTime fechaFinal) : DateTime, fechaFinal : DateTime) : int
+EstanDivorciados(persona1 : Persona, persona2 : Persona) : Boolean
+EliminarPersona(persona : Persona) : void
+Familia()


VentanaProposito
-coordenadaX1 : int
-coordenadaY1 : int
-coordenadaX2 : int
-coordenadaY2 : int
-miBitmap : Bitmap
-cambioPendiente : Boolean
-totalIndividuos : int
-areaSelecionada : Rectangle
-familiaVisual1 : FamiliaVisual
-VisorDePropiedades : PropertyGrid
-btNuevoArbol : Button
-btBuscarArbol : Button
-btGuardarArbol : Button
-btAumentarDimensiones : Button
-btDisminuirDimenciones : Button
-btVistaTabular : Button
-btEliminarIndividuo : Button
-btInsertarMasculino : Button
-btInsertarFemenino : Button
-btInsertarDesconocido : Button
-mnNuevoArbol : MenuItem
-mnBuscarArbol : MenuItem
-mnGuardarArbol : MenuItem
-mnExportarArbol : MenuItem
-mnExportarReporte : MenuItem
-mnExportarImagen : MenuItem
-mnAumentarDimensiones : MenuItem
-mnReducirDimensiones : MenuItem
-mnDefinicion : MenuItem
-mnDefinirEnfermedad : MenuItem
-mnDefinirSimbolo : MenuItem
-ucFamiliaVisual : FamiliaVisual
-pbSeparador1 : PictureBox
-pbSeparador2 : PictureBox
-pbSeparador3 : PictureBox
-pbSeparador4 : PictureBox
-sdGuardarArbol : SaveFileDialog
-sdGuardarImagen : SaveFileDialog
-sdGuardarReporte : SaveFileDialog
-odCargarArbol : OpenFileDialog
-mnCopiarImagen : MenuItem
-mnGuardarSeleccion : MenuItem
-CVentanaProposito_EventoPersonaSeleccionada(sender : Object, e : EventAgrsPersonaSeleccionada) : Void
-CVentanaProposito_EventoFamiliaSeleccionada(sender : Object, e : EventAgrsFamiliaSeleccionada) : Void
-MainWindow_Load(sender : Object, e : EventArgs) : Void
-VisorDePropiedades_PropertyValueChanged(s : Object, e : PropertyValueChangedEventArgs) : Void
-mnDefinirEnfermedad_Click(sender : Object, e : EventArgs) : Void
-mnDefinirSimbolo_Click(sender : Object, e : EventArgs) : Void
-mnExportarReporte_Click(sender : Object, e : EventArgs) : Void
-mnAumentarDimensiones_Click(sender : Object, e : EventArgs) : Void
-mnReducirDimensiones_Click(sender : Object, e : EventArgs) : Void
-mnExportarImagen_Click(sender : Object, e : EventArgs) : Void
-mnGuardarArbol_Click(sender : Object, e : EventArgs) : Void
-mnBuscarArbol_Click(sender : Object, e : EventArgs) : Void
-mnNuevoArbol_Click(sender : Object, e : EventArgs) : Void
-VentanaProposito_Closing(sender : Object, e : CancelEventArgs) : Void
-ucFamiliaVisual_MouseDown(sender : Object, e : MouseEventArgs) : Void
-ucFamiliaVisual_MouseMove(sender : Object, e : MouseEventArgs) : Void
-ucFamiliaVisual_MouseUp(sender : Object, e : MouseEventArgs) : Void
-mnCopiarImagen_Click_1(sender : Object, e : EventArgs) : Void
-mnGuardarSeleccion_Click(sender : Object, e : EventArgs) : Void
-sdGuardarArbol_FileOk(sender : Object, e : CancelEventArgs) : Void
-btNuevoArbol_Click(sender : Object, e : EventArgs) : Void
-btBuscarArbol_Click(sender : Object, e : EventArgs) : Void
-btGuardarArbol_Click(sender : Object, e : EventArgs) : Void
-btAumentarDimensiones_Click(sender : Object, e : EventArgs) : Void
-btDisminuirDimenciones_Click(sender : Object, e : EventArgs) : Void
-btEliminarIndividuo_Click(sender : Object, e : EventArgs) : Void
-btInsertarMasculino_Click(sender : Object, e : EventArgs) : Void
-btInsertarFemenino_Click(sender : Object, e : EventArgs) : Void
-btInsertarDesconocido_Click(sender : Object, e : EventArgs) : Void

Anexo 9 Acta de Liberación de Productos Software

Acta de Liberación de Productos Software

Acta de Liberación de Productos Software

Árbol Genealógico.

Versión 1.0

Acta de Liberación de Productos Software

Control de versiones

Fecha	Versión	Descripción	Autor
08/04/2008	1.0	Se emite el acta de liberación de la aplicación y Manual de Usuario de ArboGen.	Ing. Roig Calzadilla Díaz.

Emitida a favor de: Árbol Genealógico.
Estructura: Dirección del Proyecto.
Responsable Ing. Alfonso Claro Arceo.

Proyecto: ArboGen **Producto:** Árbol Genealógico

Acta de Liberación de Productos Software

1. Datos del producto

1.1. Clasificado como:

Aplicación Desktop.

1.2. Detalle de los elementos probados y su estado final:

Artefacto	Estado final
Aplicación.	0 NC.

1.3. Versión:

Proyecto: ArboGen **Producto:** Árbol Genealógico

Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

2. Elementos revisados o probados y herramientas utilizadas

Elemento	Herramienta
Diseño y Ortografía de las interfaces.	
Funcionalidad del Sistema.	

2.1. Cantidad total de horas empleadas y rango de fechas:

Se empleo 120 horas a la revisión desde el día 04/03/08 hasta el 04/04/08.

2.1.1. Estructura del equipo de prueba empleado y turnos de trabajo

En la revisión participaron 2 Especialistas de Calidad, un Asesor Calidad, 12 estudiantes.

Proyecto: ArboGen **Producto:** Árbol Genealógico

Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

3. Evaluado por:

3.1. Especialista principal Asignado:

Ing. Tayché Capote García.

Ing. Roig Calzadilla Díaz.

3.2. Asesores y especialistas involucrados:

Nombre y apellidos	Rol ocupado o tarea
Ing. Nadia Porro Lugo.	Asesor de Calidad.

3.3. Otro personal especializado participante:

Nombre y apellidos	Rol ocupado o tarea

Proyecto: ArboGen **Producto:** Árbol Genealógico

Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

4. Aprobado por:

4.1. Laboratorio de pruebas:

Ing. Tayché Capote García.

4.2. Dirección General de la IP

Ing. Alejandro Gabriel Machado Cento.

4.3. Fecha de Liberación:

8 de abril del 2008.

_____	_____
Laboratorio de Pruebas	Equipo de Desarrollo
Ing. Roig Calzadilla Díaz.	Ing. Alfonso Claro Arceo.

Proyecto: ArboGen **Producto:** Árbol Genealógico

GLOSARIO DE TÉRMINOS

ADN mitocondrial: Es el material genético de las mitocondrias.

ADNmt: ADN Mitocondrial, es una molécula circular de una macromolécula compleja (DNA)

Árboles genealógicos: Es una representación gráfica que expone los datos genealógicos de un individuo en una forma organizada y sistemática.

BRCA1: Gen en el cromosoma 17.

BRCA2: Gen en el cromosoma 13.

RUP: Metodología Proceso Unificado de Desarrollo.

XP: Programación extrema.

CD: Disco compacto, es un soporte digital óptico utilizado para almacenar cualquier tipo de información.

CIL: Es un lenguaje ensamblador orientado a objetos

CNGM: Nacional de Genética Médica.

Compilador: Programa que traduce el código fuente a lenguaje de máquina.

CU: Caso de uso.

CUS: Caso de uso sistema.

Depurador: Programa que corrige errores en el código fuente para que pueda ser bien compilado.

Editor de código: Programa donde se escribe el código fuente.

Gen: es el conjunto de una secuencia determinada de nucleótidos de uno de los lados de la escalera del cromosoma referenciado.

Genealogía: Conjunto de antepasados de una persona.

Genealogistas: Persona entendida en genealogías y linajes, y que escribe sobre ellos.

Genética Médica: Es la rama de la medicina que se dedica a la prevención, diagnóstico y tratamiento de la patología y alteraciones de origen genético.

Genograma: Es un formato para dibujar un árbol genealógico que registra información sobre los miembros de una familia y sus relaciones sobre por lo menos tres generaciones.

GOF: Patrón de diseño Gang of Four.

GRASP: Es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades).

GUI: Es una forma de programar en la que no es necesario escribir el código para la parte gráfica del programa, sino que se puede hacer de forma visual.

Hematología: Parte de la biología o de la medicina que realiza el estudio histológico, funcional y patológico de la sangre.

Hemofilia: Enfermedad hereditaria ligada al cromosoma X, caracterizada por la dificultad en la coagulación de la sangre.

Hemofílicos: Portadores de la hemofilia.

Heráldica: Es la ciencia y arte auxiliar de la Historia que estudia la composición y significado de los escudos de armas o blasones.

Homocigótico: Individuo con alelos idénticos en uno o más loci de cromosomas homólogos.

Homosexual: Siente atracción sexual por individuos de su mismo sexo.

HTML: Lenguaje de Marcado de Hipertexto, es el lenguaje de marcado predominante para la construcción de páginas web.

IDE: Entorno de desarrollo integrado.

Metafile: Es un código escrito específicamente para intentar explotar una vulnerabilidad crítica.

Mortinatos: Nacimiento de un niño muerto.

Muestra: Examen tomado al individuo.

MVC: Patrón arquitectónico Modelo-Vista-Controlador.

PDA: Personal Digital Assistant (Asistente Digital Personal), es un computador de mano originalmente diseñado como agenda electrónica.

Prenatal: Anterior al nacimiento

AND: Ácido desoxirribonucleico.

PropertyGrid: Es un editor de propiedades de objetos. Es exactamente el mismo control que utiliza el IDE para editar las propiedades de los controles y demás elementos que constituyen una aplicación.

Propósito: Persona a partir de la que se traza el árbol genealógico.

RTTI: RunTime Type Identification (Tipo de identificación en tiempo de ejecución).

Serialización: Es el proceso en el que se toman objetos y se convierte su información de estado en un formato que permita su transporte o su almacenamiento.

UCI: Universidad de las Ciencias Informática.

UML: Lenguaje Unificado de Modelado.