Desarrollo de una Interfaz de Usuario para una Máquina de Hemodiálisis

Olmos, K.[1], Gómez, M.[2], Rascón, L.[3]
Instituto de Ingeniería y Tecnología
Universidad Autónoma de Ciudad Juárez
[1]kolmos@uacj.mx, [2]mgomez@uacj.mx, [3]lrascon@uacj.mx

Introducción

Este trabajo muestra el desarrollo de un software cuya funcionalidad principal es servir de interfaz de usuario de una máquina de hemodiálisis. Este proyecto surge de la idea de construir una máquina de hemodiálisis que utilice una computadora como interfaz de usuario y que permita almacenar la información de los pacientes, médicos y enfermeras involucrados en el proceso. Un trabajo similar desarrollado por (Bengtsoon & Bosch, 1999) presenta las experiencias en el diseño del software para una máquina de hemodiálisis y discute la metodología aplicada; sin embargo, hace una separación entre hardware y software y sólo muestra el modelado del software.

La máquina de hemodiálisis propuesta se compone de una computadora personal, un sistema de control y un sistema físico. La metodología utilizada para desarrollar el software se basó en el proceso de desarrollo presentado por Larman (2003). Esta metodología es iterativa e incremental y divide el proyecto en ciclos de desarrollo. En cada uno de éstos se realizan las etapas de análisis, diseño, codificación y pruebas. En este trabajo, el análisis se realizó bajo un enfoque integral, en el que se considera la máquina de hemodiálisis (hardware y software) como un todo. Para facilitar el mantenimiento y los cambios en un futuro, el diseño del software fue realizado bajo el paradigma de orientación a objetos (Booch, Lovelle, & Cernuda,

1996), con una arquitectura basada en capas (Weitzenfeld, 2005) y utilizando los patrones GRASP, controlador, experto, alta cohesión y bajo acoplamiento, para la asignación de responsabilidades (Larman, 2003).

Como resultado se obtuvo un software cuya funcionalidad principal es actuar como interfaz de usuario para la máquina de hemodiálisis propuesta. La verificación del software se llevó a cabo al sustituir el sistema de control por una computadora personal que ejecuta un software que emula las funcionalidades de este último. Además, al haber sido diseñado bajo el paradigma de orientación a objetos y tomando en cuenta los patrones de diseño y la arquitectura de tres capas, se considera que el software cumple con los factores externos de corrección, extensibilidad, reutilización y facilidad de uso, cualidades que de acuerdo a Meyer (1999) determinan la calidad de un sistema de software.

1. Máquina de Hemodiálisis

La función principal de una máquina de hemodiálisis es eliminar las toxinas de la sangre en personas con insuficiencia renal. En una sesión de hemodiálisis se le extrae la sangre al paciente por medio de un catéter para que circule a través de un circuito externo en el que se filtra para eliminar toxinas y exceso de agua. Una vez libre de toxinas la sangre es regresada al paciente.

En la figura 1 (Graves, 2001) se muestran los principios básicos del proceso de una máquina de hemodiálisis la cual utiliza una bomba pulsátil para mantener el flujo sanguíneo simulando el bombeo que hace el corazón. Mientras la sangre circula fuera del paciente es necesario inyectar heparina para evitar su coagulación. Para ello se incluye un mecanismo que presiona el émbolo de una jeringa con un flujo definido para toda la sesión. El proceso de eliminación de toxinas se realiza mediante

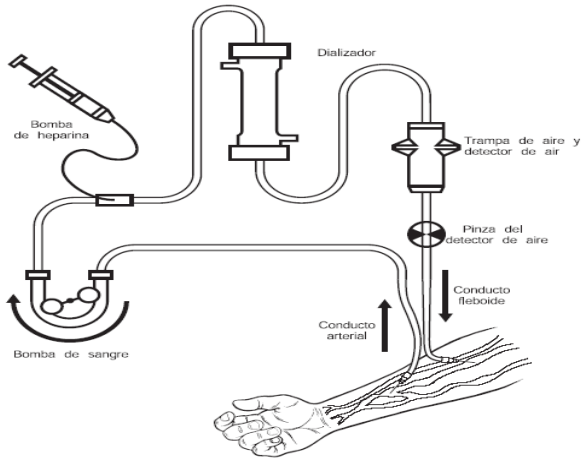

Figura 1. Principios básicos del proceso de una maquina de hemodiálisis (Graves, 2001).

un filtro al cual le llega la sangre y una solución salina. Las toxinas de sangre son liberadas por diferencia de presiones. Para evitar que se altere la temperatura de la sangre que se le regresará al paciente es importante que la solución salina se mantenga a una temperatura similar a la corporal.

Este proceso incluye un sistema de alarmas para cuidar la seguridad del paciente. Es necesario monitorear el funcionamiento de la bomba de sangre y la bomba de heparina así como monitorear presiones arterial y venosa, temperatura de la solución salina y detectar burbujas en la sangre antes de que ésta se regrese al paciente. En caso de que alguna variable esté fuera de su intervalo preestablecido se activa un oclusor que presiona la línea que regresa la sangre al paciente.

En la figura 2 se puede observar la máquina de hemodiálisis propuesta la cual consta de una computadora personal, un sistema de control y un sistema físico. La computadora personal albergará un sistema de software cuya función principal es actuar como interfaz de usuario. La

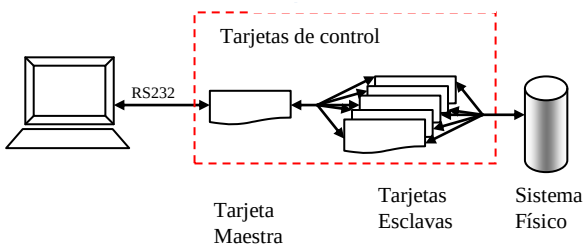

Figura 2. Esquema general de la máquina de hemodiálisis (Vazquez, Ortega, Ochoa, Gómez, Rascón, & Olmos, 2001).

computadora a su vez se comunica con el sistema de control a través del puerto serie y este último sistema se conecta a los componentes físicos (Vazquez, Ortega, Ochoa, Gómez, Rascón, & Olmos, 2001).

El sistema de software permite configurar las especificaciones de la sesión de hemodiálisis y sirve como interfaz entre el operador y el sistema físico para realizar la sesión de diálisis. El sistema de control se divide en dos secciones: una tarjeta maestra y cuatro tarjetas esclavas. La tarjeta maestra realiza la comunicación entre la computadora y las tarjetas esclavas. Esta comunicación se lleva a cabo al recibir los códigos enviados por la computadora personal, analizarlos y reenviarlos a la tarjeta esclava correspondiente. El monitoreo y control del sistema físico se divide en cuatro secciones: bomba de sangre, bomba de heparina, control de temperatura y alarmas. Por cuestiones de seguridad se utiliza un esquema de control distribuido en el que se asigna una tarjeta esclava a cada una de estas secciones. (Gómez & Ortega, 2002).

2. Metodología

El desarrollo del sistema de software de la máquina de hemodiálisis se realizó siguiendo una metodología basada en el proceso presentado por Larman (2003). Esta metodología utiliza UML (Booch, Rumbaugh, & Jacobson, 2005) como lenguaje de modelado, es iterativa e

incremental y divide el proyecto en ciclos de desarrollo, en cada uno de los cuales se realizan las etapas de análisis, diseño, codificación y pruebas. Los resultados del análisis y diseño del primer ciclo de desarrollo fueron reportados en (Olmos, Gómez, Bravo, & Rascón, 2005).

El análisis se realizó bajo un enfoque integral en el que se considera la máquina de hemodiálisis (hardware y software) como un todo. Los artefactos que se obtuvieron en esta etapa fueron el modelo de casos de uso, que consta del diagrama de casos de uso y de los flujos de eventos de cada uno de ellos, el modelo conceptual y el diagrama de estados.

En la etapa de diseño se elaboraron los diagramas de actividades y los diagramas de secuencia para cada caso de uso, así como el diagrama de clases. Con el fin de que el sistema fuera mantenible, reutilizable, legible y flexible, el diseño se realizó con un enfoque de arquitectura de tres capas, además de utilizar los patrones GRASP para asignar las responsabilidades de las clases que constituyen el sistema de software (Larman, 2003). El diagrama de clases se obtuvo a partir de los diagramas de secuencia.

La funcionalidad del software se probó al remplazar el sistema de control por una computadora personal que ejecuta un software que emula las funcionalidades de este último. El software de emulación permite seleccionar los códigos que enviaría el sistema de control al comunicarse con la interfaz de usuario. Así mismo, recibe y despliega los códigos que esta última le enviaría al sistema de control.

3. Análisis del sistema

El desarrollo del sistema de software de la máquina de hemodiálisis requirió, en primer lugar, entender el funcionamiento de la máquina como un todo. Por lo tanto, el análisis de requerimientos abarcó

diferentes estrategias como 1) entrevistas a nefrólogos y a enfermeras especializadas, 2) asistencia a sesiones de hemodiálisis, 3) análisis de sesiones grabadas en video y 4) utilización de ingeniería inversa a una máquina de hemodiálisis. Los artefactos generados en la etapa de análisis fueron el modelo de casos de uso, el modelo conceptual y el diagrama de estados, los cuales se explican a continuación.

3.1. Modelo de Casos de Uso

El modelo de casos de uso consta del diagrama de casos de uso y el flujo de eventos asociado a cada uno. Para la generación de los casos de uso se siguieron las recomendaciones proporcionadas en (Armour & Miller, 2001). Un total de dieciocho casos de uso fueron encontrados, dentro de los más significativos se encuentran: inicializar sistema, normalizar sistema y realizar sesión (Olmos, Ortega, Gómez, Fernández, & Rascón, 2002).

3.2. Modelo Conceptual

El modelo conceptual, reportado en (Olmos, Ortega, Gómez, Fernández, & Rascón, 2002) se obtuvo a partir de los casos de uso. El modelo representa a la máquina propuesta en dos niveles de abstracción; en el primer nivel, la máquina de hemodiálisis se divide en el sistema de software, el sistema de control y el sistema físico, de acuerdo a la arquitectura mostrada en la figura 2. En el segundo nivel estos subsistemas se dividen en componentes y se representan a un nivel de detalle suficiente. Lo que permite conocer las interacciones entre los componentes y entre los sistemas. En este nivel se puede observar que el sistema de software será responsable de la administración de los pacientes, de los usuarios y de las sesiones de diálisis. Además, este sistema tendrá un módulo de comunicación que permitirá intercambiar mensajes con el sistema de control. El modelo conceptual del sistema

de software se puede observar en la figura 3.

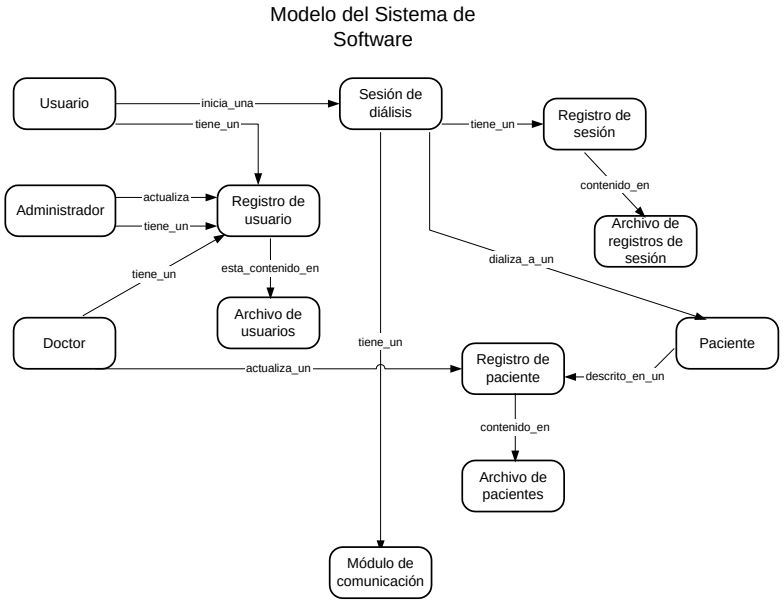

Figura 3. Modelo conceptual del sistema de software (Olmos, Ortega, Gómez, Fernández, & Rascón, 2002).

3.3. Modelo de estados

Las máquinas de estado son especialmente útiles para comprender el comportamiento del sistema al modelar los aspectos dinámicos de éste. Una máquina de estados especifica las secuencias de estados por las que pasa un objeto, un caso de uso o un sistema completo, a lo largo de su vida. Esta secuencia de determina de acuerdo a los eventos recibidos y a la respuesta del sistema hacia estos eventos (Booch, Rumbaugh, & Jacobson, 2005).

En la figura 4 se observa el diagrama de estados del caso de uso de "Realizar sesión de hemodiálisis". El programa espera que el usuario introduzca sus datos y después de validarlos muestra el menú principal. Si el operador selecciona la opción *Realizar sesión de hemodiálisis*, el programa envía un código de inicio al sistema de control y espera a que éste le regrese el código que

representa el estado actual del sistema físico. El estado se determina al encender el sistema de control cuando cada módulo del sistema físico realiza un auto diagnóstico de sus componentes.

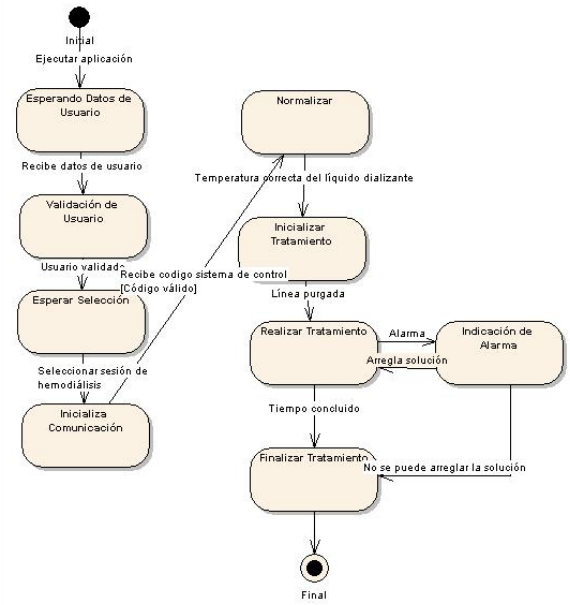

Figura 4. Diagrama de estados de la sesión de hemodiálisis.

Si el código que envía el sistema de control al sistema de software es válido el operador debe elegir la temperatura a la cual se mantendrá el líquido dializante y seleccionar la opción *Normalizar*. Cuando se alcanza la temperatura establecida, el sistema de software le indica al operador que puede introducir las especificaciones requeridas e iniciar el tratamiento de diálisis. En caso de que se presente una alarma durante el tratamiento el sistema de control manda un código al sistema de software, el cual indicará al operador el tipo de alarma a través de la interfaz de usuario. Es importante mencionar que la acción ejecutada en respuesta a la alarma se realiza automáticamente en el sistema físico o por el operador. Si el problema que causa la alarma se resuelve el sistema de software continúa con el tratamiento, si no,

éste se interrumpe y el operador finaliza la sesión.

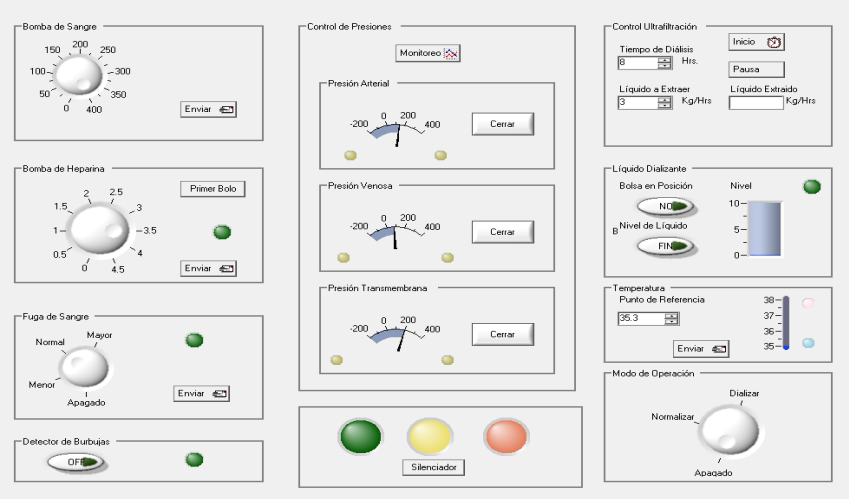

Figura 5. Interfaz de usuario de la máquina de hemodiálisis.

4. Diseño

4.1 Diseño Interfaz de Usuario

El diseño de la interfaz de usuario se realizó tomando como base la máquina de hemodiálisis Baxter 550, la cual dispone de indicadores analógicos y se opera de una forma manual mediante perillas y selectores. Sin embargo, fue necesario agregar controles que estuvieran en concordancia con la arquitectura de la máquina de hemodiálisis propuesta y que permitieran y facilitaran la operación de la máquina mediante la computadora personal. El diseño de la interfaz de usuario se muestra en la figura 5.

4.2. Diseño dinámico

El diseño dinámico consta de los diagramas de actividades y los diagramas de colaboración de cada uno de los casos de uso. Para facilitar el mantenimiento y los cambios en un futuro, el diseño del software fue realizado bajo el paradigma de orientación a objetos, con una arquitectura basada en capas y utilizando los patrones GRASP de controlador, experto, alta cohesión y bajo acoplamiento,

para la asignación de responsabilidades. Algunas de las clases se obtuvieron del modelo conceptual, sin embargo se tuvieron que agregar clases que permitieran la funcionalidad esperada del software.

4.3. Diseño estático

El diagrama de clases representa el diseño estático del sistema de software. Este diagrama se obtuvo al concentrar la información de cada uno de los diagramas de secuencia generados en el diseño dinámico. La figura 6 representa el diagrama de clases del sistema de software. Como puede observarse el usuario y el sistema de control se consideran actores. La clase *ControladorSesionDialisis* se generó de acuerdo al patrón controlador y es una de las clases más importantes ya que es responsable de la secuencia de eventos en una sesión de diálisis. Una de las estrategias que se siguió fue generar una clase por cada sección de la interfaz de usuario que agrupara a los indicadores y controles. Por ejemplo, en la figura 7 se puede observar el diagrama de clases de la sección bomba de heparina. La clase *ControladorBombaHeparina* agrupa a una perilla, un indicador y los botones de *Primer bolo* y *Enviar*, que están en correspondencia con esta sección en la interfaz de usuario. Así mismo, cada sección de la interfaz de usuario tendrá su clase correspondiente. Debido a sus funciones similares estas clases se derivarán de la clase abstracta *ControladorSecciones* como se muestra en la figura 8.

Para lograr una arquitectura basada en capas, la estrategia a seguir fue separar las clases del dominio de las clases de interfaz, como se menciona en (Weitzenfeld, 2005). Los eventos de la interfaz de usuario son capturados por una clase denominada *GUI*, la cual únicamente tiene la responsabilidad

del manejo de eventos del operador. Esta clase manda un mensaje con el evento solicitado a la clase *ControladorSesionDialisis*. Ésta determina las acciones a seguir: modificar los controles de la interfaz o comunicarse con el sistema de control. En el primer caso la clase es responsable de determinar qué secciones están involucradas en el evento solicitado por el usuario y enviar la información necesaria para que realicen su tarea. En el segundo caso se envía un mensaje a la clase *Comunicador* para que ésta sea la responsable de mandar el mensaje al sistema de control. La clase *Comunicador* es la que contiene toda la información relacionada con la comunicación a través del puerto serie con el sistema de control. El hecho de utilizar una clase especialmente diseñada para el manejo de estos eventos permitirá en un futuro modificar el esquema de comunicación sin afectar las clases relacionadas con el dominio de la aplicación.

En el caso de que el sistema de control requiera comunicarse con la PC la clase comunicador recibirá el mensaje y lo pasará a la clase *AnalizadorDeCódigo*, la cual es responsable de descifrar el código y enviarlo al o a los controladores de sección para que estos a su vez manden mensajes a sus elementos y se visualice en la pantalla los efectos del código enviado por el sistema de control. Cabe mencionar que según la arquitectura de la máquina, el sistema de software únicamente es responsable de indicar al usuario los eventos que están sucediendo en la sesión de diálisis. La creación de una clase responsable de analizar los códigos enviados por el sistema de control permite delimitar las responsabilidades y que el diseño se considere de bajo acoplamiento. Esto permitiría en un futuro hacer cambios

en los códigos de envío, sin afectar las clases del dominio de la aplicación.

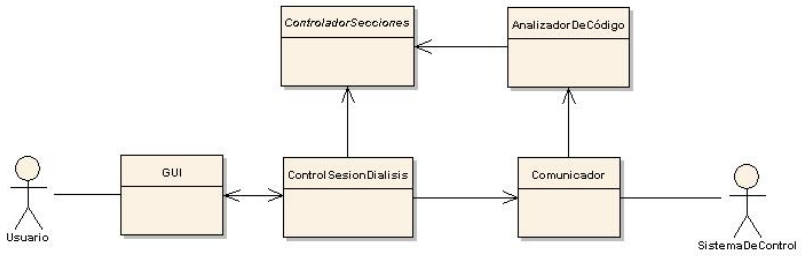

Figura 6. Diagrama de clases del sistema de software.

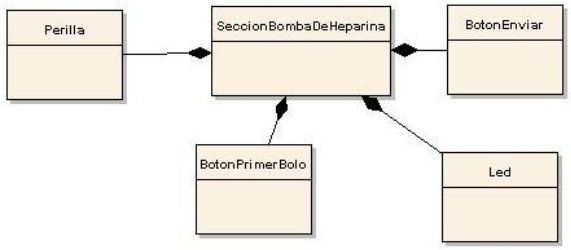

Figura 7. Diagrama de clases de la sección bomba de heparina.

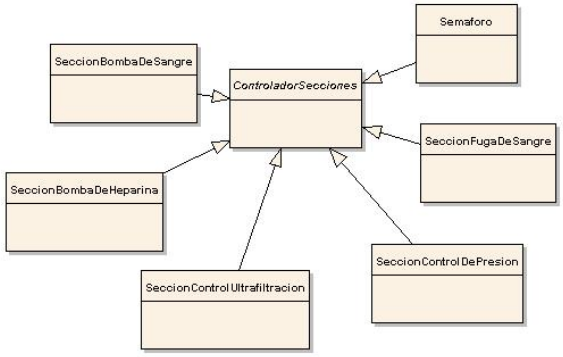

Figura 8. Clases abstracta *ControladorSecciones* y sus clases derivadas.

5. Resultados

El código de la interfaz de usuario se desarrolló tomando como base el diseño estático y dinámico. El programa se desarrolló utilizando el entorno de Visual Studio 2005 con el lenguaje de

programación C# y utilizando los controles e indicadores proporcionados por *Measurement Studio* de *National Instruments*. El punto crítico en la codificación fue la configuración del puerto serie, la cual fue resuelta en el trabajo de Hernández y Herrera (2006).

Se considera que el uso de patrones GRASP y el diseño basado en la arquitectura de tres capas proporciona al sistema atributos de calidad como mantenibilidad, reutilizabilidad, legibilidad y flexibilidad.

Como se mencionó al inicio del artículo, para realizar el sistema de software y probarlo se sustituyó el sistema de control por una computadora personal. Las pruebas se llevaron a cabo utilizando los casos de uso como casos de prueba. Durante las pruebas el sistema se comportó de acuerdo a los requerimientos establecidos. En el diseño de la máquina de hemodiálisis el sistema de control (hardware) es el responsable del sistema físico y la funcionalidad del sistema de software es servir de interfaz de usuario. Dependiendo del tipo de alarma el usuario o el sistema de control deberán encargarse de estabilizar el sistema. Por lo tanto, en el caso de las alarmas, el sistema de software únicamente activa los controles correspondientes en la interfaz pero no realiza ninguna acción para reponerse de la alarma.

6. Conclusiones

Una de las decisiones más relevantes de este proyecto fue iniciar con el análisis del funcionamiento de la máquina de hemodiálisis de forma integral en lugar de trabajar únicamente con el sistema de software. Esto permitió determinar los requerimientos del sistema de software de una forma más precisa y contextualizada.

Por otro lado, el seguir un proceso de desarrollo permitió que el sistema de software se construyera de una manera

sistemática y que se pudiera llevar un control durante el desarrollo del mismo. La principal ventaja del proceso fue que permitió desarrollar un software que cumpliera con los requerimientos de funcionalidad.

El proceso implicó contar con una documentación en UML lo que facilitó la tarea de integración de nuevos miembros al proyecto. En particular los casos de uso mostraron su efectividad al ser una adecuada fuente de información para entender el sistema, ya que los nuevos miembros del equipo de desarrollo sólo requirieron del estudio de éstos para involucrarse en el dominio de la aplicación.

Por lo anterior, consideramos que esta metodología sería de gran utilidad para los ingenieros de desarrollo de hardware principalmente en aplicaciones complejas basadas en microcontroladores. Lo anterior debido a que desde el inicio del diseño se deben considerar todos los escenarios del sistema además de que la documentación se generaría conforme se avance en el proyecto.

Con esta aplicación se da por terminado el desarrollo del sistema de software, como trabajo futuro se esperaría finalizar con el sistema de control y realizar las pruebas del sistema en forma integral.

Como aplicación adicional creemos que este sistema de software puede servir como ayuda en la capacitación de médicos y enfermeras que requieran aprender el funcionamiento de una máquina de hemodiálisis.

Trabajos citados

Armour, F., & Miller, G. (2001). *Advanced Use Case Modeling.* Addison-Wesley.

Bengtsoon, P., & Bosch, J. (1999). Haemo Dialysis Arquitecture Design Experiences. *International Conference on Software Enginiering*, (págs. 516-526).

Booch, G., Lovelle, C., & Cernuda, J. M. (1996). *Análisis y Diseño Orientado a Objetos.* Addison Wesley.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified Modeling Language User Guide* (2 edition ed.). Addison-Wesley.

Gómez, M., & Ortega, L. (2002). Diseño de un Sistema de Tarjetas Maestra-Esclava basada en la Vista de Casos de Uso para una Máquina de Hemodiálisis. *XXIV Congreso Internacional de Electrónica. 24.* Chihuahua, México: Instituto Tecnológico de Chihuahua.

Graves, G. (2001). Arterial and Venous Pressure Monitoring During Hemodialysis. *Nephrology Nursing Journal , 28* (1), 23-30.

Hernández, G., & Herrera, I. (2006). *Desarrollo de un Sistema de Software para la manipulación de una Máquina de Hemodiálisis utilizando el Paradigma de Orientación a Objetos.* Tesis de Licenciatura, IIT-UACJ, Juárez, México.

Larman, C. (2003). *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Madrid: Pearson Education.

Meyer, B. (1999). *Construcción de software Orientado a Objetos.* Madrid: Prentice Hall.

Olmos, K., Gómez, M., Bravo, G., & Rascón, L. (2005). Utilización de un Proceso de Desarrollo de Software para la Construcción de una Máquina de Hemodiálisis: Etapas de Análisis y Diseño. *CONIELECOMP*. Puebla.

Olmos, K., Ortega, L., Gómez, M., Fernández, L. F., & Rascón, L. (2002). Utilización de UML en el Modelado de una Máquina de Hemodiálisis desde el punto de vista de Casos de Uso. *SOMI (Sociedad Mexicana de Instrumentación).* Mérida.

Vazquez, R., Ortega, L., Ochoa, H., Gómez, M., Rascón, L., & Olmos, K. (2001). Diseño y Construcción de una

Máquina que no Utiliza el Sistema de Preparación de Líquido Dializante. *Foro Estatal SIVILLA-CHIHUAHUA.* Chihuahua.

Weitzenfeld, A. (2005). *Ingeniería de Software orientada a objetos con UML, Java e Internet.* México: Thompson.