SISTEMA DE INFORMACIÓN PARA OPTIMIZAR EL PROCESO DEL SISTEMA DE COSTOS UNIVERSITARIOS DE LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

MOISÉS GONZÁLEZ CARREÑO
GIOVANNI ALFONZO LÓPEZ CASTILLO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

SISTEMA DE INFORMACIÓN PARA OPTIMIZAR EL PROCESO DEL SISTEMA DE COSTOS UNIVERSITARIOS DE LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

MOISÉS GONZÁLEZ CARREÑO
GIOVANNI ALFONZO LÓPEZ CASTILLO

Proyecto de grado para optar por el título: Ingeniero de Sistemas

Directora
MARIELA RIVERO RIVERA
Ingeniera de Sistemas e Informática

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

TABLA DE CONTENIDO

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

TITULO: SISTEMA DE INFORMACIÓN PARA OPTIMIZAR EL PROCESO DEL SISTEMA DE COSTOS UNIVERSITARIOS DE LA UNIVERSIDAD INDUSTRIAL DE SANTANDER.

AUTORES: LÓPEZ CASTILLO GIOVANNI ALFONZO**

GONZÁLEZ CARREÑO MOISÉS**

PALABRAS CLAVE: Sistema de Información Web, Costos Universitarios, Aplicaciones Cliente Servidor, Java, Modelo de Construcción de Prototipos, Implementación, EISI (Escuela de Ingeniería de Sistemas e Informática).

DESCRIPCION: En la actualidad Planeación cuenta con un sistema de información de costos universitarios el cual se desarrolló buscando dotar a la UIS de herramientas confiables en la obtención de la información sobre los costos asociados a sus actividades básicas de docencia, investigación, extensión y asesorías y servicios, para así brindar la información necesaria para la planeación institucional, dotando a planeación con indicadores y datos estadísticos para poder darles seguimiento y hacer inteligencia de negocios, pero la herramienta actual a pesar de cumplir con sus objetivos no está implementada con las herramientas tecnológicas de la época haciendo de esta una herramienta poco amigable para el usuario, complicada en su operación y de difícil mantenimiento al carecer en la actualidad de personal con la capacidad de darle mantenimiento; Es por esto que se hace necesario la realización de un proyecto cuyo objetivo general sea: "Desarrollar un sistema de información en ambiente web, para optimizar el proceso del sistema de costos universitarios de la Universidad Industrial de Santander."

Los requerimientos fueron definidos por el cliente, que en éste caso es la unidad académico administrativa 'PLANEACION' de la Universidad Industrial de Santander, con quienes se formuló el diseño y la posterior evaluación del producto final entregado desarrollado en este proyecto con el modelo de construcción de prototipos como metodología.

Este nuevo producto se realizó con los estándares de calidad y eficiencia de la División de Servicios de Información de la Universidad Industrial de Santander en los desarrollos de los nuevos sistemas de información.

---

* Trabajo de grado. Modalidad: Trabajo de investigación.

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática.

Director: Ingeniera. Mariela Rivero Rivera

ABSTRACT

TITLE: INFORMATION SYSTEM TO OPTIMIZE THE PROCESS OF THE INFORMATION SYSTEM OF UNIVERSITY COSTS AT THE INDUSTRIAL UNIVERSITY OF SANTANDER. *

AUTHORS: LOPEZ CASTILLO GIOVANNI ALFONZO**

GONZALEZ CARREÑO MOISES**

KEYWORDS: Web Information System, University Costs, Client Serving Applications, Java, Prototype Constructions Model, Implementation, EISI (School of Systems Engineering and Computer Science).

DESCRIPTION: Currently Planning has a university information system of university costs which was developed looking for to provide the UIS reliable tools in obtaining information about the costs associated with its basic activities of teaching, research, extension and consulting and services, thus to provide the information necessary for institutional planning, endowing to Planning with indicators and data statistics to track them and to do business intelligence, but the current tool despite meeting its objectives is not implemented with the technological tools of the time making this an unfriendly tool for the user, complicated in its operation and difficult to maintain because currently lacking staff with the ability to maintain it; that is why the performance is necessary a project whose overall objective is: “Development a web information system environment to optimize the process of the information systems of university costs at the Universidad Industrial de Santander.”

The requirements were defined by the customer, which in this case is the academic administrative unit 'PLANNING' of the Industrial University of Santander, with whom he made the design and subsequent evaluation of the final product delivered, developed the building model prototyping as a methodology.

This new product came about with the standards of quality and efficiency of the Division of Information Services of the Industrial University of Santander during the ongoing development of the new information systems.

---

* Work Degree. Modality: Investigation Work.

** Physical mechanical Engineering Faculty. School of Systems Engineering and Computer Science.

Director: Engineer. Mariela Rivero Rivera

INTRODUCCIÓN

Hoy en día, contar con un software, que permita manejar la información de manera eficiente, eficaz y efectiva, es el deseo de cualquier organización. Desarrollar una solución software en base a los problemas, necesidades y oportunidades del cliente es un gran reto.

La Universidad Industrial de Santander actualmente cuenta con un software para llevar a cabo el proceso del sistema de costos universitarios, desarrollado en modo carácter bajo la plataforma 4GL, el cual presenta dificultades en cuanto al mantenimiento, debido a que hay poco personal capacitado en la plataforma que realice estas actividades, su interfaz es poco amigable, su acceso solo se hace desde el interior de la institución, hay poca documentación con respecto al diseño del software y aún hay procesos que se realizan manualmente y que deben automatizarse; debido a estas razones se requiere dar una solución inmediata, confiable, ágil y amigable.

Este proyecto, plantea desarrollar una nueva versión para el sistema de información de costos universitarios (SICU) en ambiente web, para apoyar los procesos de Planeación de la Universidad Industrial de Santander, satisfaciendo las necesidades presentadas por los usuarios interesados y cumpliendo con los estándares de desarrollo que contempla la universidad.

La primera metodología de Costos Universitarios fue presentada en el año de 1973 por el Instituto Colombiano para el Fomento de la Educación Superior (ICFES), orientada a ser aplicada por las Instituciones de educación superior, especialmente las de índole oficial, con el fin de poder conocer los costos unitarios y totales basados en las actividades y programas que desarrollan dichas instituciones y a su vez brindar la información necesaria para la planeación institucional. La Universidad Industrial de Santander adecuó, de acuerdo a sus requerimientos y necesidades, esta metodología produciendo un primer modelo de Costos Universitarios en 1977, el cual se ha venido modificando y perfeccionando a través del tiempo.

El Sistema de Información de Costos Universitarios se desarrolló buscando dotar a la UIS de herramientas confiables en la obtención de la información sobre los costos asociados a sus actividades básicas de docencia, investigación, extensión y asesorías y servicios.

El SICU se integra fundamentalmente con los subsistemas de información Académico, Financiero y de Recursos Humanos, generando no sólo la información necesaria para el análisis de costos sino que también posibilita la generación de indicadores y datos estadísticos.

Con la herramienta tecnológica actual, la información sólo puede ser accedida desde equipos de cómputo que estén dentro de la red de la UIS y no desde cualquier equipo o dispositivo que esté conectado a internet por fuera del claustro universitario, además su lenguaje nativo de desarrollo es el lenguaje de programación Informix-4GL, el cuál día a día se hace de difícil mantenimiento debido al escaso recurso humano que da soporte a este lenguaje además que la División de Servicios de Información (DSI) de la Universidad Industrial de Santander busca en su día a día proporcionar servicios informáticos dentro del proceso de modernización institucional, por estas razones se hace necesario el desarrollo de una nueva versión del sistema SICU en ambiente web que aproveche las ventajas tecnológicas de esta época y de la infraestructura informática que posee la UIS.

1 JUSTIFICACIÓN

En la actualidad el manejo y administración de la información es una parte fundamental de toda entidad u organización, ya que ésta es una pieza clave en la toma de decisiones, seguimiento y control de actividades, con el fin de tomar medidas correctivas cuando los resultados se desvían significativamente de los objetivos y propósitos. Por esta razón, cada día se ve la necesidad de buscar herramientas que permitan manejar dicha información.

De acuerdo a los avances tecnológicos, la mayoría de las entidades han optado en utilizar sistemas de información, los cuales proporcionan soluciones prácticas en el procesamiento de ésta, en forma rápida y confiable, teniendo en cuenta un medio fundamental que es el internet, el cual permite que haya una conexión permanente entre los miembros de la organización, el usuario final y el proceso que se lleva a cabo.

La Universidad Industrial de Santander maneja un Sistema de Información de Costos Universitarios en modo carácter, herramienta poco amigable y de restringido acceso, puesto que solo se puede acceder a este sistema desde el interior de la universidad, además aún se procesa manualmente información que se debe sistematizar para que los procesos sean más integrados y actualizados.

El propósito del proyecto es ofrecer soluciones a las falencias ya mencionadas, desarrollando una nueva versión del SICU en un ambiente web, de tal manera que los diferentes usuarios interesados y todo aquel que esté debidamente autorizado podrá desde cualquier lugar consultar y obtener información sobre los diferentes elementos de costos universitarios en forma oportuna, confiable e inmediata, cumpliendo con los estándares de desarrollo que contempla la universidad.

2 OBJETIVOS

2.1 Objetivo General

Desarrollar un sistema de información en ambiente web, para optimizar el proceso del sistema de costos universitarios de la Universidad Industrial de Santander.

2.2 Objetivos Específicos

- Crear el módulo para actualizar constantes del sistema, que se utiliza como información básica para el proceso de costos universitarios.
- Generar las interfaces del sistema de información de planta física, recursos humanos, académico y financiero, con el fin de obtener la información necesaria para generar los cuadros de estudio del sistema de costos universitarios.
- Construir el módulo para generar los cuadros de costos universitarios, que son:

    1. Gastos anuales de las unidades de costo (Función docencia, función investigación, función extensión y función de apoyo).
    2. Distribución de recursos de las unidades de costo de función docencia.
    3. Distribución del tiempo semanal según actividad y valor de las colaboraciones para las Unidades Académico -Administrativas.
    4. Gastos semanales de las unidades de costo.
    5. Actividad laboral en las unidades de costo de función docencia, distribuido en comités y administración.
    6. Gasto semanal de apoyo de función docencia, investigación y extensión.
    7. Determinación de puntos por factor para cada unidad de costo de función docencia.

8. Distribución porcentual de los puntos por descriptor entre las unidades de costo de función docencia.
9. Distribución del gasto semanal de apoyo entre las unidades de costo de función docencia.
10. Costo total de las unidades de costo de función docencia.
11. Costo semanal por hora contacto y por EHSS (Estudiante hora/semana servido) de las unidades de costo de función docencia.
12. Costo semanal de cada programa académico.
13. Costo por estudiante según programa académico.
14. Costos anuales de la función docencia, función investigación y función extensión.

- Desarrollar e implementar la interfaz gráfica para la navegación web del sistema de costos universitarios, en la cual los usuarios debidamente autorizados podrán:

1. Ingresar, modificar, consultar o eliminar la información requerida por el Sistema de Información de Costos Universitarios.
2. Consultar la información de los 14 cuadros de estudio mencionados anteriormente.

3 FUNDAMENTOS TEÓRICOS

3.1 SISTEMAS DE INFORMACIÓN

Definición de sistemas de información: El Diccionario de la Real Academia de la Lengua Española en su edición vigésimo segunda define la palabra sistema como: “Conjunto de cosas que relacionadas entre sí ordenadamente contribuyen a determinado objeto”.

Otras de las definiciones que comúnmente encontramos es: Conjunto de componentes interrelacionados que recolectan (o recuperan), procesan, almacenan y distribuyen información para apoyar los procesos de toma de decisiones y de control en una organización.

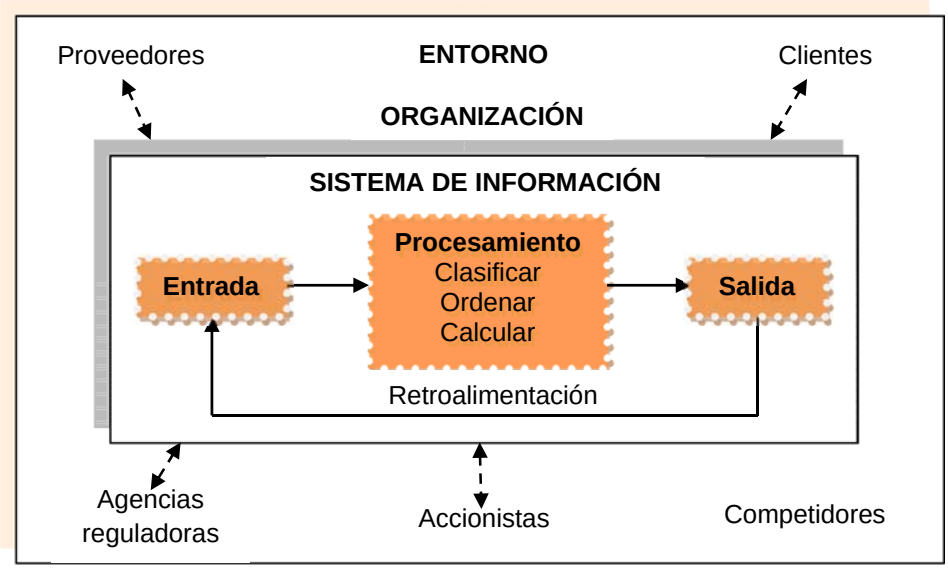
Los sistemas de información realizan cuatro actividades básicas: entrada, almacenamiento, procesamiento y salida de información.

- *Entrada de Información:* Es el proceso mediante el cual el Sistema de Información toma los datos que requiere para procesar la información. Las entradas pueden ser manuales o automáticas. Las manuales son aquellas que se proporcionan en forma directa por el usuario, mientras que las automáticas son datos tomados de otros sistemas o módulos.
- *Almacenamiento de información:* El almacenamiento de la información en bases de datos es una de las características o capacidades más importantes de los sistemas hoy en día, ya que a través de éstas, el sistema puede manipular la información según como sea solicitada.
- *Procesamiento de Información:* Es la capacidad del Sistema de Información para efectuar operaciones de acuerdo con una secuencia de acciones preestablecidas. Estas operaciones pueden efectuarse con datos introducidos recientemente en el sistema o bien con datos que estén almacenados. Esta característica de los sistemas permite la transformación

de datos fuente, en información que puede ser utilizada para la toma de decisiones.

- *Salida de Información:* Es importante aclarar que la salida de un Sistema de Información puede constituir la entrada a otro Sistema de Información o módulo.

Figura 1. Funciones de un sistema de información.



3.1.1 Tipos y Usos de los Sistemas de Información: Los objetivos básicos de un Sistema de Información dentro de la organización son:

- Automatización de procesos operativos.
- Proporcionar información que sirva de apoyo al proceso de toma de decisiones.
- Lograr ventajas competitivas a través de su implantación y uso.

Los Sistemas de Información que logran la automatización de procesos operativos dentro de una organización, son llamados frecuentemente *Sistemas*

Transaccionales, ya que su función primordial consiste en el procesamiento de transacciones tales como pagos, cobros, pólizas, entradas, salidas.

Por otra parte, los Sistemas de Información que apoyan el proceso de toma de decisiones son los *Sistemas de Soporte a la toma de decisiones.* El tercer tipo de sistema, de acuerdo con su uso u objetivos que cumplen, es el de los *Sistemas Estratégicos*, los cuales se desarrollan en las organizaciones con el fin de lograr ventajas competitivas, a través del uso de la tecnología de información.

3.2 ARQUITECTURA DE INFORMACIÓN

3.2.1 Modelo Cliente/Servidor: El modelo arquitectónico cliente-servidor es un modelo de sistema en el que dicho sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que acceden y usan los servicios. Los principales componentes de este modelo son:

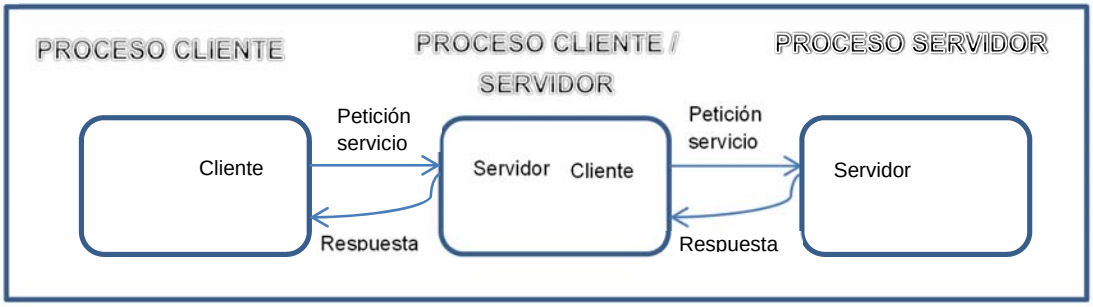
- Un conjunto de servidores que ofrecen servicios a otros subsistemas. Ejemplos de servidores son servidores de impresoras que ofrecen servicios de impresión, servidores de ficheros que ofrecen servicios de gestión de ficheros y servidores de compilación, que ofrecen servicios de compilación de lenguajes de programación.
- Un conjunto de clientes que llaman a los servicios ofrecidos por los servidores. Estos son normalmente subsistemas en sí mismos. Puede haber varias instancias de un programa cliente ejecutándose concurrentemente.
- Una red que permite a los clientes acceder a estos servicios. Esto no es estrictamente necesario ya que los clientes y los servidores podrían ejecutarse sobre una única máquina.

La mayoría de los sistemas cliente-servidor se implementan como sistemas distribuidos. Los clientes pueden conocer los nombres de los servidores disponibles y los servicios que éstos proporcionan. Sin embargo, los servidores no necesitan conocer la identidad de los clientes o cuántos clientes tienen. Los clientes acceden a los servicios proporcionados por un servidor a través de llamadas a procedimientos remotos usando un protocolo de petición-respuesta tal como el

protocolo http usado en la WWW (Por sus siglas en Inglés, World Wide Web). Básicamente, un cliente realiza una petición a un servidor y espera hasta que recibe una respuesta.

La ventaja más importante del modelo cliente-servidor es que es una arquitectura distribuida. Se puede hacer un uso efectivo de los sistemas en red con muchos procesadores distribuidos. Es fácil añadir un nuevo servidor e integrarlo con el resto del sistema o actualizar los servidores de forma transparente sin afectar al resto del sistema. Arquitectura para Aplicaciones Distribuidas en Internet.

Figura 2. Ejemplo de comunicación de procesos en el modelo Cliente/Servidor.

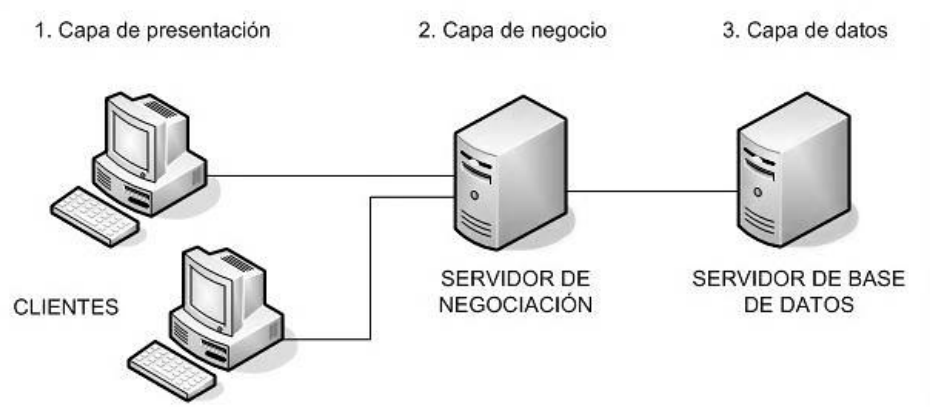


3.2.2 Arquitectura de Tres Capas: Es una arquitectura cliente/servidor en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado dando así gran escalabilidad. Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la Interfaz de Programación de Aplicaciones (API) que existe entre niveles.

- *Capa de presentación:* Es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.

Figura 3. Modelo de arquitectura de tres capas.



- *Capa de negocio:* Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

- *Capa de datos:* es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

3.3 GENERALIDADES DEL ENTORNO DE DESARROLLO

3.3.1 Aplicaciones basadas a la Web: Los sistemas y aplicaciones basados a la Web ofrecen un complejo arreglo de contenido y funcionalidad a una amplia población de usuarios finales. La ingeniería Web es el proceso con el que se crean aplicaciones Web de alta calidad.

Conforme las aplicaciones Web se integran cada vez más en las estrategias de negocios y servicios, crece en importancia la necesidad de construir sistemas confiables, prácticos y adaptables.

Al igual que cualquier disciplina, la ingeniería Web aplica un enfoque genérico que se suaviza mediante estrategias, tácticas y métodos especializados. El sistema se construye con tecnologías y herramientas especializadas asociadas a la Web. Entonces se entrega a los usuarios finales y se evalúa mediante criterios tanto técnicos como empresariales. Dado que las aplicaciones Web evolucionan continuamente, se deben establecer mecanismos para el control de configuraciones, el aseguramiento de la calidad y el soporte continuo.

En ocasiones es difícil estar seguro de que lo que se ha hecho esta correcto, hasta que los usuarios finales ejecutan la aplicación Web. Sin embargo, se aplican prácticas de aseguramiento de la calidad del software para valorar la calidad de los modelos, la facilidad de uso, el desempeño y la seguridad.

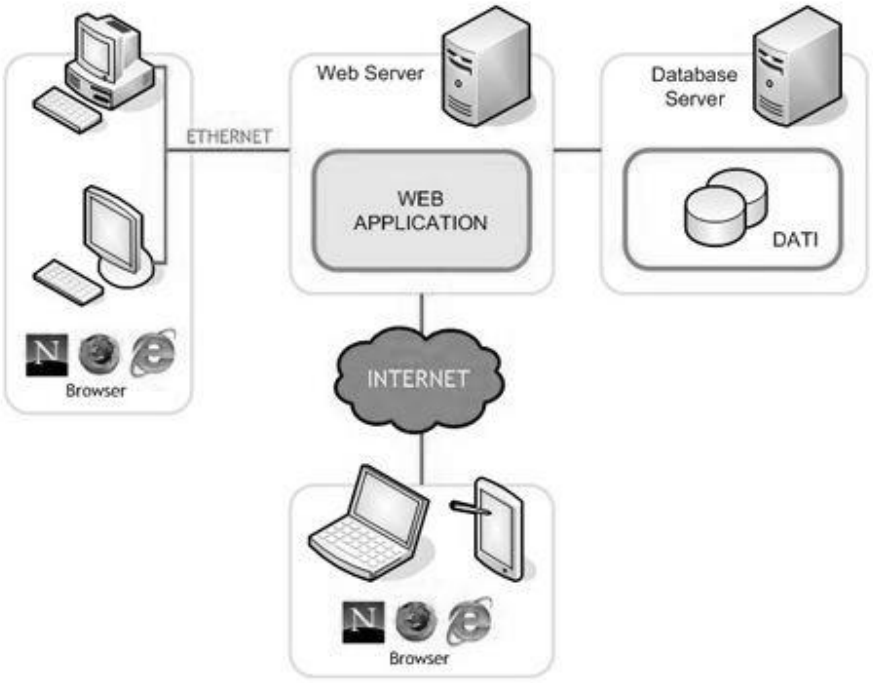
3.3.1.1 ***La web como sistema de información:*** El alcance de las aplicaciones basadas en la Web ha crecido enormemente, ahora existen cuatro tipos generales de sistemas basados en la Web: Intranets, para el apoyo a la red de trabajo interno, sitios Web-presence que están comercializando herramientas diseñadas para llegar a los consumidores fuera de la empresa, sistemas de comercio electrónico que soportan la interacción de los clientes, como las compras en línea, y una mezcla de los sistemas internos y externos para apoyar la comunicación de business-to-business, comúnmente llamadas extranets.

Por lo tanto, una plataforma Web se ha transformado en los últimos años desde una mera presencia en el mercado a una plataforma que puede soportar todas las facetas del trabajo organizacional. Como resultado de ello, importantes esfuerzos de los sistemas de información se orientan cada vez más hacia la explotación de los beneficios de esta plataforma, lo que lleva al desarrollo de sistemas de información basados en tecnología Web, lo que llamamos "sistemas de información basados en web" (WIS).

Hay una clara diferencia entre un conjunto de páginas web y un WIS. Este último apoya el trabajo, y por lo general se integra perfectamente con otros no-WIS, tales como bases de datos y sistemas de procesamiento de transacciones. Los WIS también son diferentes de los sistemas de información tradicionales. Se requieren nuevos enfoques para el diseño y desarrollo, tienen el potencial de llegar a un público mucho más amplio, y son generalmente el resultado de los esfuerzos de base. Estas diferencias introducen retos gerenciales y técnicos.

Turoff y Hiltz acuñan un nuevo término "superconectividad" que destaca el potencial de la Web como un conducto para la "transmisión casi perfecta de la comunicación y la información" y para definir la eventual interconexión global de todas las personas y organizaciones. Económica, la superconectividad transparente está dando lugar a nuevos tipos de organizaciones y nuevas formas de interactuar con las organizaciones existentes.

Figura 4. Arquitectura de un WIS.



Las WIS son exitosas, sin embargo, el éxito depende del desarrollo. Como afirma Dennis: " WIS son los primeros sistemas de información y los sistemas Web segundos... en el desarrollo de WIS se deben usar los mismos principios disciplinados... necesarios para construir sistemas de información no-web de éxito" Dennis destaca cuatro lecciones importantes para los administradores y desarrolladores a lo largo de este tema. Desarrollo web, sin embargo, es lo suficientemente diferente del desarrollo tradicional de Sistemas de Información (IS) ya que requiere nuevos enfoques de la ingeniería de software.

3.3.1.2 ***Tecnologías web:*** Inicialmente, era difícil la construcción de aplicaciones sofisticadas. La primera generación de aplicaciones Web era primitiva, en general basada en formularios con información y aplicaciones de búsqueda. Incluso estas aplicaciones básicas requerían de un alto seniority para su construcción.

A través del tiempo, el conocimiento necesario para construir aplicaciones ha sido reducido. Hoy en día, es relativamente sencillo construir aplicaciones sofisticadas utilizando las modernas plataformas y lenguajes, como pueden ser PHP, .NET o Java.

Primera generación – CGI: Common Gateway Interface (CGI) fue la tecnología reinante desde aproximadamente 1993 hasta fines de los '90 cuando los lenguajes de scripting comenzaron a ganar importancia.

CGI trabaja encapsulando la información provista por el usuario en variables de ambiente. Estas luego son accedidas por scripts o programas desarrollados comúnmente en Perl o C. Estos programas procesan la información provista por los usuarios, y luego envían código HTML con la información procesada a la salida estándar, que a su vez es capturada por el servidor Web y pasada al usuario.

Scripting: La falta de manejos de sesiones y control de autorización por parte de CGI impidió el desarrollo de aplicaciones Web comerciales con esa tecnología.

Los desarrolladores Web comenzaron entonces a utilizar lenguajes de script, como JavaScript o PHP para resolver esos problemas. Básicamente los lenguajes de script son ejecutados en el servidor Web y como son no compilados son desarrollados e implementados más fácilmente.

Los lenguajes de script tienen algunas desventajas:

- La mayoría de los lenguajes no son tipados y no promueven buenas prácticas de programación.
- Son más lentos en comparación con los lenguajes compilados (a veces hasta 100 veces más lentos).

- Es difícil (no imposible) escribir aplicaciones de múltiples capas porque en general las capas de presentación, aplicación y datos residen en la misma máquina, limitando de esta forma la escalabilidad y seguridad.
- La mayoría no soporta nativamente métodos remotos o llamadas a Web services, lo que hace difícil la comunicación entre servidores de aplicación y con Web services externos.

De cualquier manera a pesar de las desventajas aplicaciones grandes y frecuentemente accedidas han sido desarrolladas utilizando lenguajes de script, como eGroupWare (egroupware.org), que está escrita en PHP. Además muchas aplicaciones de Internet banking han sido desarrolladas en ASP.

Los lenguajes de script incluyen, ASP, Perl, Cold Fusion y PHP. De cualquier manera, muchos de esos podrían ser considerados como lenguajes interpretados híbridos, en particular las últimas versiones de PHP y Cold Fusion.

3.3.1.3 ***Plataformas de desarrollo web:*** El impacto, transversalidad y popularidad de los sistemas basados en Web, han sido acompañados de una multiplicidad de tecnologías que soportan el desarrollo de aplicaciones empresariales orientadas a Web. Ahora se trataran dos de las plataformas de desarrollo de aplicaciones empresariales orientadas a la web más conocidas y utilizadas, como lo son J2EE y .NET.

J2EE

J2EE, es una plataforma para soluciones empresariales que define el estándar para el desarrollo de aplicaciones empresariales de varios niveles haciéndolas más simples, basándolas en componentes modulares y estandarizados, ofrece cantidad de servicios para estos componentes, y facilita la tarea del programador automatizando detalles de comportamiento de la aplicación.

Este modelo divide las aplicaciones empresariales en tres partes fundamentales:

- *Componentes:* Núcleo principal para los desarrolladores.
- *Conectores:* Están bajo la plataforma definiendo un servicio portable API que se comunica con las ofertas existentes en los proveedores, Promueven la flexibilidad con variedad de implementaciones de servicios específicos, En especial implementan contratos de servicios de mensajería que permiten la comunicación bidireccional entre los componentes de la plataforma y los sistemas empresariales.
- *Contenedores:* Interceden ante los clientes y componentes, prestando servicios transparentes a ambos incluyendo soporte de transacción y agrupación de recursos, esto permite implementar el comportamiento de estos en la implementación.

Principales Características:

- Utiliza muchas características de la plataforma de Java 2, Edición Estándar (J2SE), como “Escríbelo una vez, córrelo en cualquier parte”, la portabilidad, el API JDBC para la base de datos de Access, la tecnología CORBA para la interacción con los recursos existentes, y un modelo de seguridad que protege los datos incluso en las aplicaciones de internet.
- Agrega soporte completo para las aplicaciones con componentes de JavaBeans, Java Servlets API, Java Server pages y tecnología XML.
- Incluye especificaciones completas y test de cumplimiento para asegurar la portabilidad de la aplicación en todos los sistemas empresariales existentes que soportan esta plataforma.
- Asegura la interoperabilidad de servicios web por medio del la inclusión del perfil WS-I Basic, es decir que se puede construir aplicaciones sobre esta plataforma como servicios web que interactúan con los servicios web desde entornos no compatibles con J2EE.
- Integra la simplicidad, portabilidad, escalabilidad, e integración con sistemas heredados.

- El modelo de la aplicación encapsula las capas de funcionalidad en funcionalidad en tipos específicos de componentes. La lógica de negocio se encapsula en componentes Enterprise Java Beans (EJB). La Interacción con el cliente puede darse a través de simples páginas web HTML, a través de las páginas web impulsadas por los applets, Java Servlets, tecnología JavaServer Pages, o a través de aplicaciones independientes de Java. Los componentes se comunican de forma transparente usando varios estándares: HTML, XML, HTTP, SSL, RMI, IIOP, entre otros.

.NET

Es una plataforma de desarrollo y ejecución que ofrece todas las herramientas y servicios para la creación de aplicación empresariales con una ejecución óptima. Es la evolución de COM (Component Object Model) basada en desarrollo de aplicación de visual basic.

Componentes:

- Runtime o entorno de ejecución, que es el componente que permite ejecutar la aplicación (iniciarla y detenerla) e interactuar con el sistema operativo para proveer a la aplicación servicios y recursos.

- Bibliotecas que ofrecen controles y funciones con componentes listos para usar con otras aplicaciones.

- Lenguajes de programación de alto nivel, junto con compiladores para permitir el desarrollo de aplicaciones en la plataforma.

- Utilitarios y herramientas de desarrollo encargados de simplificar las tareas comunes al programador.

- Documentación de las mejores prácticas de diseño, organización, desarrollo, prueba e instalación de las aplicaciones.

Principales Características:

- Basada en el paradigma de orientación a objetos.
- Ejecución intermedia, Es decir que la ejecución de las aplicaciones se hace por medio del runtime no del sistema operativo.
- Es multilenguaje, ofrece una lista de lenguajes para codificar los programas no exige un lenguaje especifico.
- Permite la creación y ejecución de aplicaciones empresariales grandes y complejas para la operación de distintos tipos de organizaciones.
- .Net fue diseñado de manera tal de poder proveer un único modelo de programación, uniforme y consistente, para todo tipo de aplicaciones (ya sean de formularios Windows, de consola, aplicaciones Web, aplicaciones móviles, etc.) y para cualquier dispositivo de hardware (PC's, Pocket PC's, Teléfonos Celulares Inteligentes, también llamados "SmartPhones", Tablet PC's, etc.). Esto representa un gran cambio con respecto a las plataformas anteriores a .NET, las cuales tenían modelos de programación, bibliotecas, lenguajes y herramientas distintas según el tipo de aplicación y el dispositivo de hardware.
- Hace posible integrar las aplicaciones con otras creadas en la plataforma anterior COM, en plataformas de Microsoft, en otras plataformas de software, sistemas operativos o lenguajes de programación, haciendo uso de estándares globales como XML, HTTP, SOAP, WSDL y UDDI.

3.3.1.4 ***Web 2.0:*** Hay tres principios que pueden definir la Web 2.0:

- *Comunidad:* el usuario aporta contenidos, interactúa con otros usuarios, crea redes de conocimiento, etc.
- *Tecnología:* un mayor ancho de banda permite transferir información a una velocidad antes inimaginable. En lugar de paquetes de software, podemos tener servicios web y nuestro terminal puede ser cliente y servidor al mismo tiempo y en cualquier lugar del mundo.

- *Arquitectura modular:* favorece la creación de aplicaciones complejas de forma más rápida y a un menor coste.

Una novedad, y clave del éxito de la filosofía Web 2.0, es que, finalmente, no es el editor del website el que aporta los contenidos y decide cuáles son interesantes y cuáles no, sino que es la propia comunidad la que proporciona y promociona determinados contenidos en detrimento de otros.

Para tener una noción de la participación de los usuarios en la Web 2.0 podemos mencionar la regla del 1%, que asegura que, de cada 100 personas que usan este tipo de servicios, aproximadamente 90 únicamente consultan, 9 participan y sólo 1 crea contenidos. En el caso de los blogs, la regla es mucho más desalentadora. Según UseIt, los blogs tienen la peor desigualdad de participación, puesto que es evidente que la 90-9-1, regla que caracteriza la mayoría de las comunidades on-line, se transforma en 95-4,9-0,1 en los blogs.

3.4 JAVA

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable.

Java fue diseñado con unos pocos principios clave en mente:

- *Fácil de usar:* Los fundamentos de Java provenían de un lenguaje de programación llamado c ++. Aunque c ++ es un lenguaje poderoso, se consideró demasiado complejo en su sintaxis, e insuficiente para todos los requisitos de Java. Java se basa, y mejora las ideas de c ++, para proporcionar un lenguaje de programación poderoso y fácil de usar.
- *Confiabilidad:* Java necesitó reducir la probabilidad de errores fatales desde los errores de programación. Con esto en mente, la programación orientada

a objetos se introdujo. Una vez los datos y su manipulación se empaquetaron en un solo lugar, aumentó la robustez de Java.

- *Seguro:* Como Java apuntaba originalmente a dispositivos móviles que intercambian datos a través de redes, fue construido para incluir un alto nivel de seguridad. Java es probablemente el lenguaje de programación más segura hasta la fecha.

- *Plataforma Independiente:* Los programas necesitaban trabajar independientemente de la máquina en que estaban siendo ejecutados. Java fue escrito para ser un lenguaje portable que no se preocupe por el sistema operativo o el hardware de la computadora.

***3.4.1 Java Enterprise Edition 5.0 (Jee5) Jee5*:** JEE5 se centra en hacer el desarrollo más fácil, sin embargo, conserva la riqueza de la plataforma J2EE 1.4. Ofreciendo características tales como JavaServer Faces (JSF) API de servicios tecnológicos y de web, Java EE 5 hace la codificación más simple y sencilla, pero mantiene el poder que ha establecido Java EE como la primera plataforma para servicios web y desarrollo de aplicaciones empresariales.

3.4.2 Java Server Faces (JSF): El objetivo de la tecnología Java Server Faces es desarrollar aplicaciones web. Tradicionalmente, las aplicaciones web se han codificado mediante páginas JSP (Java Server Pages) que recibían peticiones a través de formularios y construían como respuesta páginas HTML (Hiper Text Markup Language) mediante ejecución directa o indirecta -a través de bibliotecas de etiquetas- de código Java, lo que permitía, por ejemplo, acceder a bases de datos para obtener los resultados a mostrar, amén de realizar operaciones marginales como insertar o modificar registros en tablas relacionales, actualizar un carrito de la compra, etc.

JSF pretende facilitar la construcción de estas aplicaciones proporcionando un entorno de trabajo (framework) vía web que gestiona las acciones producidas por el usuario en su página HTML y las traduce a eventos que son enviados al servidor con el objetivo de regenerar la página original y reflejar los cambios pertinentes provocados por dichas acciones. En definitivas cuentas, se trata de hacer

aplicaciones Java en las que el cliente no es una ventana de la clase JFrame o similar, sino una página HTML, entonces cualquier evento realizado sobre una página JSF incurre en una carga sobre la red, ya que el evento debe enviarse a través de ésta al servidor, y la respuesta de éste debe devolverse al cliente; por ello, el diseño de aplicaciones Java Server Faces debe hacerse con cuidado cuando se pretenda poner las aplicaciones a disposición del mundo entero a través de internet. Aquellas aplicaciones que vayan a ser utilizadas en una intranet podrán aprovecharse de un mayor ancho de banda y producirán una respuesta mucho más rápida.

Características principales

La tecnología JSF constituye un marco de trabajo (framework) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC.

Los principales componentes de la tecnología JSF son:

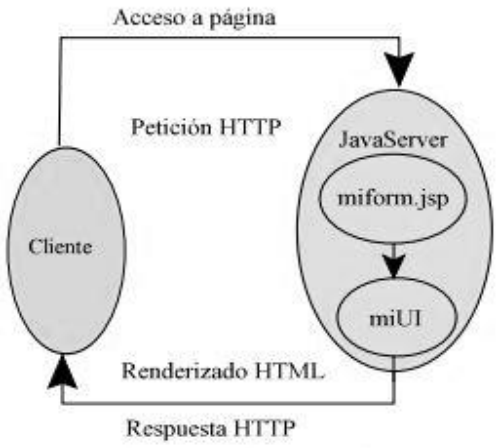
- Una API y una implementación de referencia para:
    1. Representar componentes de interfaz de usuario (UI-User Interface) y manejar su estado.
    2. Manejar eventos, validar en el lado del servidor y convertir datos.
    3. Definir la navegación entre páginas.
    4. Soportar internacionalización y accesibilidad, y proporcionar extensibilidad para todas estas características.
- Una librería de etiquetas Java Server Pages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido y la librería de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y

mantenimiento de aplicaciones web con UI's en el lado servidor. Con un mínimo esfuerzo, es posible:

- Conectar eventos generados en el cliente a código de la aplicación en el lado servidor.
- Mapear componentes UI a una página de datos en el lado servidor.
- Construir una interfaz de usuario con componentes reutilizables y extensibles.

Figura 5. Diagrama de un aplicación JSF.



En la Figura 5 se muestra la interfaz de usuario creada con la tecnología JSF representada por "miUI" que se ejecuta en el servidor y se renderiza en el cliente. La página JSP, "miform.jsp", especifica los componentes de la interfaz de usuario mediante etiquetas personalizadas definidas por JSF. "miUI" maneja los objetos referenciados por la JSP, que pueden ser de los siguientes tipos:

- Objetos componentes que mapean las etiquetas sobre la página JSP.
- Oyentes de eventos, validadores y conversores registrados y asociados a los componentes.

- Objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación (lógica de negocio).

3.4.3 Mapeo Relacional de Objetos (ORM) / API de Persistencia en Java (JPA):

JPA es una abstracción sobre JDBC (Java Database Connectivity) que nos permite realizar una correlación entre las bases de datos relacionales que almacenan la información mediante tablas, filas, y columnas, y el sistema orientado a objetos de Java de una forma sencilla, realizando por nosotros toda la conversión entre nuestros objetos y las tablas de una base de datos. Esta conversión se llama Mapeo Relacional de Objetos, y puede configurarse a través de metadatos (mediante xml o anotaciones). JPA también nos permite seguir el sentido inverso, creando objetos a partir de las tablas de una base de datos, y también de forma transparente. A estos objetos les llaman entidades (entities).

JPA establece una interface común que es implementada por un proveedor de persistencia de nuestra elección (como Hibernate, Eclipse Link, etc), de manera que podemos elegir en cualquier momento el proveedor que más se adecue a nuestras necesidades. Así, es el proveedor quién realiza el trabajo, pero siempre funcionando bajo la API de JPA.

3.4.4 EJB: Alrededor de 1996, cómo Java se estaba reforzando por la aparición de tecnologías tales cómo (RMI y JTA) que abordan las necesidades de las aplicaciones a gran escala, surgió la necesidad de un framework de componentes de negocio que pudiera unificar esas tecnologías e incorporarlas bajo un modelo de desarrollo estándar. EJB nació para cubrir esta necesidad. En los últimos 10 años, ha evolucionado hasta abarcar numerosas características (mientras expulsa otras) y ha madurado en un framework robusto y estándar para el despliegue y ejecución de los componentes de negocio en un entorno multiusuario y distribuido.

EJB 3 es definida por JSR 220: Enterprise JavaBeans 3. Mientras que la versión anterior abarcaba un solo documento, este JSR abarca ahora tres documentos (enumerados a continuación). El primer documento ofrece una síntesis de las características de alto nivel de la nueva versión, y se centra en el nuevo modelo de desarrollo simplificado para la construcción de componentes EJB. Los últimos

documentos abordan los detalles técnicos del núcleo del framework EJB y el modelo de persistencia respectivamente.

- El API simplificado de EJB 3 da una visión general de alto nivel del nuevo modelo de desarrollo EJB 3.
- El núcleo de los contratos y requerimientos de EJB se centra en la sesión y mensajes de beans controlados.
- Java Persistence API aborda las entidades y el framework de persistencia.

Lo que surge de estos tres documentos que componen las especificaciones de EJB 3 es a la vez un modelo de componentes y un framework.

Servicios proporcionados por el contenedor de EJB

- *Manejo de transacciones:* Apertura y cierre de transacciones asociadas a las llamadas a los métodos del bean.
- *Seguridad:* Comprobación de permisos de acceso a los métodos del bean.
- *Concurrencia:* Llamada simultánea a un mismo bean desde múltiples clientes.
- *Servicios de red:* Comunicación entre el cliente y el bean en máquinas distintas.
- *Gestión de recursos:* Gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.
- *Persistencia:* Sincronización entre los datos del bean y tablas de una base de datos.
- *Gestión de mensajes:* Manejo de Java Message Service (JMS).

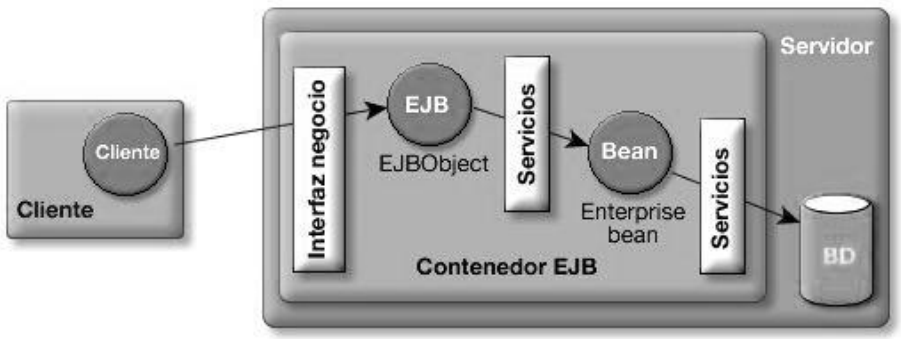
- *Escalabilidad:* Posibilidad de constituir clústers de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.
- *Adaptación en tiempo de despliegue:* Posibilidad de modificación de todas estas características en el momento del despliegue del bean.

Funcionamiento de los componentes EJB.

El funcionamiento de los componentes EJB se basa fundamentalmente en el trabajo del contenedor EJB. El contenedor EJB es un programa Java que corre en el servidor y que contiene todas las clases y objetos necesarios para el correcto funcionamiento de los enterprise beans.

En la Figura 6 se puede ver una representación de muy alto nivel del funcionamiento básico de los enterprise beans. En primer lugar, se observa que el cliente que realiza peticiones al bean y el servidor que contiene el bean están ejecutándose en máquinas virtuales Java distintas. Incluso pueden estar en distintos hosts. Otra cosa a resaltar es que el cliente nunca se comunica directamente con el enterprise bean, sino que el contenedor EJB proporciona un EJBObject que hace de interfaz. Cualquier petición del cliente (una llamada a un *método de negocio* del enterprise bean) se debe hacer a través del objeto EJB, el cual solicita al contenedor EJB una serie de servicios y se comunica con el enterprise bean. Por último, el bean realiza las peticiones correspondientes a la base de datos.

Figura 6. Ejemplo del Funcionamiento de los Enterprises Beans.



El contenedor EJB se preocupa de cuestiones como:

- ¿Tiene el cliente permiso para llamar al método?
- Hay que abrir la transacción al comienzo de la llamada y cerrarla al terminar.
- ¿Es necesario refrescar el bean con los datos de la base de datos?

Tipos de Beans

La tecnología EJB define tres tipos de beans: beans de sesión, beans de entidad y beans dirigidos por mensajes.

- *Beans de entidad:* Representan un objeto concreto que tiene existencia en alguna base de datos de la empresa. Una instancia de un bean de entidad representa una fila en una tabla de la base de datos. El uso de beans de entidad nos da una perspectiva orientada a objetos de los datos y proporciona a los programadores un mecanismo más simple para acceder y modificar los datos. Es mucho más fácil, por ejemplo, cambiar el nombre de un estudiante llamando a student.setName() que ejecutando un comando SQL contra la base de datos. Además, el uso de objetos favorece la reutilización del software. Una vez que un bean de entidad se ha definido, su definición puede usarse a lo largo de todo el sistema de forma consistente. Un bean Estudiante proporciona una forma completa de acceder a la información del estudiante y eso asegura que el acceso a la información es consistente y simple.
- *Beans dirigidos por mensajes:* Pueden escuchar mensajes de un servicio de mensajes JMS. Los clientes de estos beans nunca los llaman directamente, sino que es necesario enviar un mensaje JMS para comunicarse con ellos. Los beans dirigidos por mensajes no necesitan objetos EJBObject porque los clientes no se comunican nunca con ellos directamente.
- *Beans de sesión:* Representa un proceso o una acción de negocio. Normalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un bean de sesión. Mientras que un bean de entidad representa una cosa que se puede representar con un nombre, al pensar en

un bean de sesión deberías pensar en un verbo. Existen dos tipos de beans de sesión:

1. *Stateless (Session) EJB's:* Este tipo de EJB como su nombre lo indica no mantiene estado ("Stateless") en el "EJB Container", estos EJB's son utilizados para realizar tareas rutinarias que no requieren identificar o rastrear al cliente, algunos EJB's de este tipo son: operaciones matemáticas complejas, búsquedas generales, entre otros.

2. *Statefull (Session) EJB's:* A diferencia de "Stateless (Session) EJB's" este tipo de EJB's permiten mantener la sesión del cliente en el "EJB Container", de esta manera el cliente puede trabajar con cierto juego de datos específico administrado por el "EJB Container", la aplicación ideal para este tipo de EJB es un componente de compra ("Shopping Cart") el cual puede identificar artículos e información personal del cliente a través de un lapso de tiempo extenso ("Session").

Diferencia entre Beans de entidad y Beans de sesión

Los beans de entidad se diferencian de los beans de sesión, principalmente, en que son persistentes, permiten el acceso compartido, tienen clave primaria y pueden participar en relaciones con otros beans de entidad:

- *Persistencia*: Debido a que un bean de entidad se guarda en un mecanismo de almacenamiento se dice que es persistente. Persistente significa que el estado del bean de entidad existe más tiempo que la duración de la aplicación o del proceso del servidor J2EE. Un ejemplo de datos persistentes son los datos que se almacenan en una base de datos.

  Los beans de entidad tienen dos tipos de persistencia: *Persistencia Gestionada por el Bean* (BMP, *Bean-Managed Persistence*) y *Persistencia Gestionada por el Contenedor* (CMP, *Container-Managed Persistence*). En el primer caso (BMP) el bean de entidad contiene el código que accede a la base de datos. En el segundo caso (CMP) la relación entre las columnas de la base de datos y el bean se describe en el fichero de propiedades del bean, y el contenedor EJB se ocupa de la implementación.

- *Acceso compartido:* Los clientes pueden compartir beans de entidad, con lo que el contenedor EJB debe gestionar el acceso concurrente a los mismos y por ello debe usar transacciones. La forma de hacerlo dependerá de la política que se especifique en los descriptores del bean.

- *Clave primaria:* Cada bean de entidad tiene un identificador único. Un bean de entidad alumno, por ejemplo, puede identificarse por su número de expediente. Este identificador único, o *clave primaria*, permite al cliente localizar a un bean de entidad particular.

- *Relaciones:* De la misma forma que una tabla en una base de datos relacional, un bean de entidad puede estar relacionado con otros EJB. Por ejemplo, en una aplicación de gestión administrativa de una universidad, el bean *alumnoEjb* y el bean *actaEjb* estarían relacionados porque un alumno aparece en un acta con una calificación determinada.

  Las relaciones se implementan de forma distinta según se esté usando la persistencia manejada por el bean o por el contenedor. En el primer caso, al igual que la persistencia, el desarrollador debe programar y gestionar las relaciones. En el segundo caso es el contenedor el que se hace cargo de la gestión de las relaciones. Por ello, estas últimas se denominan a veces relaciones gestionadas por el contenedor.

Diferencias entre los Beans dirigidos por mensajes y los Beans de sesión y de entidad: La diferencia más visible es que los clientes no acceden a los beans dirigidos por mensajes mediante interfaces (explicaremos esto con más detalle más adelante), sino que un bean dirigido por mensajes sólo tienen una clase bean.

En muchos aspectos, un bean dirigido por mensajes es parecido a un bean de sesión sin estado.

- Las instancias de un bean dirigido por mensajes no almacenan ningún estado conversacional ni datos de clientes.

- Todas las instancias de los beans dirigidos por mensajes son equivalentes, lo que permite al contenedor EJB asignar un mensaje a cualquier instancia.

El contenedor puede almacenar estas instancias para permitir que los streams de mensajes sean procesados de forma concurrente.

- Un único bean dirigido por mensajes puede procesar mensajes de múltiples clientes.

Las variables de instancia de estos beans pueden contener algún estado referido al manejo de los mensajes de los clientes. Por ejemplo, pueden contener una conexión JMS, una conexión de base de datos o una referencia a un objeto enterprise bean.

Cuando llega un mensaje, el contenedor llama al método *onMessage* del bean. El método *onMessage* suele realizar un casting del mensaje a uno de los cinco tipos de mensajes de JMS y manejarlo de forma acorde con la lógica de negocio de la aplicación. El método *onMessage* puede llamar a métodos auxiliares, o puede invocar a un bean de sesión o de entidad para procesar la información del mensaje o para almacenarlo en una base de datos.

Un mensaje puede enviarse a un bean dirigido por mensajes dentro de un contexto de transacción, por lo que todas las operaciones dentro del método *onMessage* son parten de un única transacción.

Roles EJB

La arquitectura EJB define seis papeles principales. Brevemente, son:

- *Desarrollador de beans:* Desarrolla los componentes enterprise beans.
- *Ensamblador de aplicaciones:* Compone los enterprise beans y las aplicaciones cliente para conformar una aplicación completa
- *Desplegador:* Despliega la aplicación en un entorno operacional particular (servidor de aplicaciones)

- *Administrador del sistema:* configura y administra la infraestructura de computación y de red del negocio
- *Proporcionador del Contenedor EJB y Proporcionador del Servidor EJB:* Un fabricante (o fabricantes) especializado en manejo de transacciones y de aplicaciones y otros servicios de bajo nivel. Desarrollan el servidor de aplicaciones.

Ventajas de la tecnología EJB

Las ventajas que ofrece la arquitectura Enterprise JavaBeans a un desarrollador de aplicaciones se listan a continuación.

- *Simplicidad:* Debido a que el contenedor de aplicaciones libera al programador de realizar las tareas del nivel del sistema, la escritura de un enterprise bean es casi tan sencilla como la escritura de una clase Java. El desarrollador no tiene que preocuparse de temas de nivel de sistema como la seguridad, transacciones, multi-threading o la programación distribuida. Como resultado, el desarrollador de aplicaciones se concentra en la lógica de negocio y en el dominio específico de la aplicación.
- *Portabilidad de la aplicación:* Una aplicación EJB puede ser desplegada en cualquier servidor de aplicaciones que soporte J2EE.
- *Reusabilidad de componentes:* Una aplicación EJB está formada por componentes enterprise beans. Cada enterprise bean es un bloque de construcción reusable. Hay dos formas esenciales de reusar un enterprise bean a nivel de desarrollo y a nivel de aplicación cliente. Un bean desarrollado puede desplegarse en distintas aplicaciones, adaptando sus características a las necesidades de las mismas. También un bean desplegado puede ser usado por múltiples aplicaciones cliente.
- *Posibilidad de construcción de aplicaciones complejas:* La arquitectura EJB simplifica la construcción de aplicaciones complejas. Al estar basada en componentes y en un conjunto claro y bien establecido de interfaces, se facilita el desarrollo en equipo de la aplicación.

- *Separación de la lógica de presentación de la lógica de negocio:* Un enterprise bean encapsula típicamente un proceso o una entidad de negocio. (un objeto que representa datos del negocio), haciéndolo independiente de la lógica de presentación. El programador de gestión no necesita de preocuparse de cómo formatear la salida; será el programador que desarrolle la página Web el que se ocupe de ello usando los datos de salida que proporcionará el bean. Esta separación hace posible desarrollar distintas lógicas de presentación para la misma lógica de negocio o cambiar la lógica de presentación sin modificar el código que implementa el proceso de negocio.

- *Despliegue en muchos entornos operativos:* Entendemos por entornos operativos el conjunto de aplicaciones y sistemas (bases de datos, sistemas operativos, aplicaciones ya en marcha, etc.) que están instaladas en una empresa. Al detallarse claramente todas las posibilidades de despliegue de las aplicaciones, se facilita el desarrollo de herramientas que asistan y automaticen este proceso. La arquitectura permite que los beans de entidad se conecten a distintos tipos de sistemas de bases de datos.

- *Despliegue distribuido:* La arquitectura EJB hace posible que las aplicaciones se desplieguen de forma distribuida entre distintos servidores de una red. El desarrollador de beans no necesita considerar la topología del despliegue. Escribe el mismo código independientemente de si el bean se va a desplegar en una máquina o en otra (cuidado: con la especificación 2.0 esto se modifica ligeramente, al introducirse la posibilidad de los interfaces locales).

- *Interoperabilidad entre aplicaciones:* La arquitectura EJB hace más fácil la integración de múltiples aplicaciones de diferentes vendedores. El interfaz del enterprise bean con el cliente sirve como un punto bien definido de integración entre aplicaciones.

- *Integración con sistemas no-Java:* Las API's relacionadas, como las especificaciones Connector y Java Message Service (JMS), así como los beans manejados por mensajes, hacen posible la integración de los enterprise beans con sistemas no Java, como sistemas ERP o aplicaciones mainframes.

- *Recursos educativos y herramientas de desarrollo:* El hecho de que la especificación EJB sea un estándar hace que exista una creciente oferta de herramientas y formación que facilita el trabajo del desarrollador de aplicaciones EJB.

Entre las ventajas que aporta esta arquitectura al cliente final, destacamos la posibilidad de elección del servidor, la mejora en la gestión de las aplicaciones, la integración con las aplicaciones y datos ya existentes y la seguridad.

- *Elección del servidor:* Debido a que las aplicaciones EJB pueden ser ejecutadas en cualquier servidor J2EE, el cliente no queda ligado a un vendedor de servidores. Antes de que existiera la arquitectura EJB era muy difícil que una aplicación desarrollada para una determinada capa intermedia pudiera portarse a otro servidor. Con la arquitectura EJB, sin embargo, el cliente deja de estar atado a un vendedor y puede cambiar de servidor cuando sus necesidades de escalabilidad, integración, precio, seguridad, etc. lo requieran.

  Existen en el mercado algunos servidores de aplicaciones gratuitos (JBOSS, el servidor de aplicaciones de Sun, etc.) con los que sería posible hacer unas primeras pruebas del sistema, para después pasar a un servidor de aplicaciones con más funcionalidades.

- *Gestión de las aplicaciones:* Las aplicaciones son mucho más sencillas de manejar (arrancar, parar, configurar, etc.) debido a que existen herramientas de control más elaboradas.

- *Integración con aplicaciones y datos ya existentes:* La arquitectura EJB y otras APIs de J2EE simplifican y estandarizan la integración de aplicaciones EJB con aplicaciones no Java y sistemas en el entorno operativo del cliente. Por ejemplo, un cliente no tiene que cambiar un esquema de base de datos para encajar en una aplicación. En lugar de ello, se puede construir una aplicación EJB que encaje en el esquema cuando sea desplegada.

- *Seguridad:* La arquitectura EJB traslada la mayor parte de la responsabilidad de la seguridad de una aplicación del desarrollador de aplicaciones al vendedor del servidor, el Administrador de Sistemas y al Desplegador (papeles de la especificación EJB) La gente que ejecuta esos papeles están más cualificados que el desarrollador de aplicaciones para hacer segura la aplicación. Esto lleva a una mejor seguridad en las aplicaciones operacionales.

Desventajas de la tecnología EJB

- *Tiempo de Desarrollo:* El desarrollar un Sistema con EJB's es sumamente complejo, aunque para ciertas empresas puede presentar una solución ideal. Debido a la complejidad de tiempo (traduciéndose en costo) para muchas corporaciones EJB's resultan una solución sobrada, denominada en Ingles: "overkill".
- *Conocimiento exhausto de Java:* EJB's es uno de los principales componentes de J2EE y por esta razón también depende fuertemente de otras partes de J2EE: Como RMI, JNDI y JDBC.

3.5 JBOSS DEVELOPER ESTUDIO

En 1999, Marc Fleury comenzó un proyecto de código abierto que él nombró "EJBoss" (para "Enterprise Java Beans software de código abierto"). El objetivo del proyecto era proporcionar una implementación del servidor de código abierto de la especificación EJB.

El servidor rápidamente se hizo popular debido a su bajo costo y facilidad de uso. En 2001, el nombre fue cambiado a "JBoss" debido a las preocupaciones legales de Sun Microsystem sobre el uso del término "EJB". El Grupo de Jboss se incorporó por primera vez en 2001. En el 2006, JBoss fue adquirido por la empresa líder de software de código abierto del mundo, Red Hat. JBoss es conocido actualmente como "JBoss por Red Hat".

JBoss Application Server (ó JBoss AS) es un software libre/código abierto basado en el servidor de aplicaciones de Java EE. Una distinción importante para esta clase de software es que este no sólo implementa un servidor que se ejecuta en Java, pero en realidad implementa la parte de Java EE de Java. Debido a que está basado en Java, el servidor de aplicaciones JBoss opera multiplataforma: es utilizable en cualquier sistema operativo que soporte Java.

3.6 LENGUAJE DE MODELADO UNIFICADO

El lenguaje de modelado unificado (UML) proporciona un conjunto estandarizado de herramientas para documentar el análisis y diseño de un sistema de software. Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código.

En otros términos, UML ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales como procesos de negocio, funciones del sistema, expresiones de lenguajes de programación y esquemas de bases de datos en software.

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. Existen varios diagramas utilizados en UML, los cuales a continuación se explicarán brevemente algunos de ellos.

3.6.1 Diagramas de Clases: El diagrama de clases de UML muestra los objetos relevantes del dominio del problema de un sistema específico y los agrupa en clases. Para ello identifica en cada objeto los atributos que le pertenecen, las operaciones que se pueden realizar con él y sus relaciones con otros objetos (asociaciones, generalizaciones, dependencias, agregaciones y composiciones) con sus cardinalidades y el rol que desempeña en cada relación. Desde sus inicios, el diagrama de clases ha permitido el modelamiento de los conceptos del dominio de un problema determinado, pero en una notación que está dirigida a analistas entrenados en su uso, constituyéndose en un diagrama cuyo nivel de abstracción es bajo.

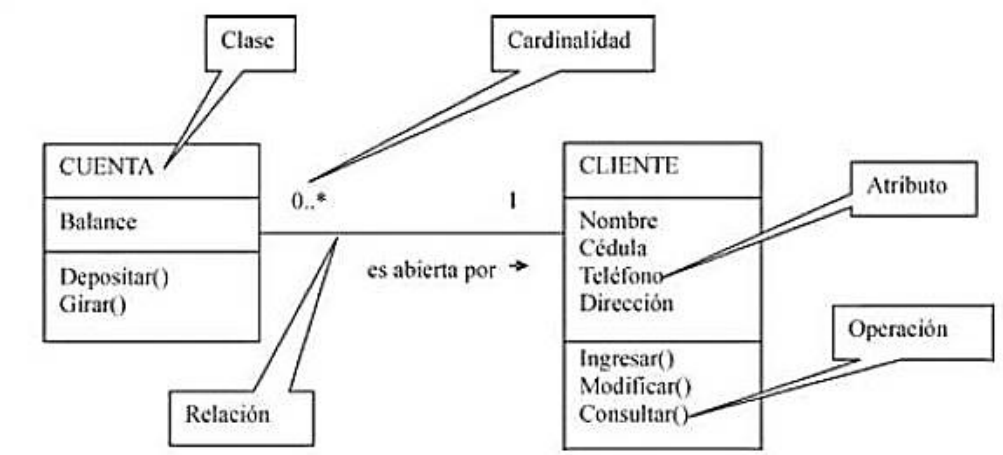
- Propiedades también llamados atributos o características, son valores que corresponden a un objeto, como color, material, cantidad, ubicación. Generalmente se conoce como la información detallada del objeto. Suponiendo que el objeto es una puerta, sus propiedades serían: la marca, tamaño, color y peso.

- Operaciones comúnmente llamados métodos, son aquellas actividades o verbos que se pueden realizar con/para este objeto, como por ejemplo abrir, cerrar, buscar, cancelar, acreditar, cargar. De la misma manera que el nombre de un atributo, el nombre de una operación se escribe con minúsculas si consta de una sola palabra. Si el nombre contiene más de una palabra, cada palabra será unida a la anterior y comenzará con una letra mayúscula, a excepción de la primera palabra que comenzará en minúscula. Por ejemplo: abrirPuerta, cerrarPuerta, buscarPuerta, etc.

- Interfaz es un conjunto de operaciones que permiten a un objeto comportarse de cierta manera, por lo que define los requerimientos mínimos del objeto. Hace referencia a polimorfismo.

- Herencia se define como la reutilización de un objeto padre ya definido para poder extender la funcionalidad en un objeto hijo. Los objetos hijos heredan todas las operaciones y/o propiedades de un objeto padre. Por ejemplo: Una persona puede especializarse en Proveedores, Acreedores, Clientes, Accionistas, Empleados; todos comparten datos básicos como una persona, pero además cada uno tendrá información adicional que depende del tipo de persona, como saldo del cliente, total de inversión del accionista, salario del empleado, etc.

Figura 7. Ejemplo de Diagrama de Clases.



Al diseñar una clase se debe pensar en cómo se puede identificar un objeto real, como una persona, un transporte, un documento o un paquete. Estos ejemplos de clases de objetos reales, es sobre lo que un sistema se diseña. Durante el proceso del diseño de las clases se toman las propiedades que identifican como único al objeto y otras propiedades adicionales como datos que corresponden al objeto.

3.6.2 Diagramas de Casos de Uso:

Los casos de uso modelan la funcionalidad del sistema según lo perciben los usuarios externos, llamados actores. Un caso de uso es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El propósito de los casos de uso es enumerar a los actores y los casos de uso, y demostrar qué actores participan en cada caso de uso.

El estándar de Lenguaje de Modelado Unificado de OMG (Object Management Group) define una notación gráfica para realizar diagramas de casos de uso, pero no el formato para describir casos de uso. Mucha gente sufre la equivocación pensando que un caso de uso es una notación gráfica (o es su descripción). Mientras la notación gráfica y las descripciones son importantes, ellos forman parte de la documentación de un caso de uso -- un propósito para el que el actor puede usar el sistema.

El valor verdadero de un caso de uso reposa en dos áreas:

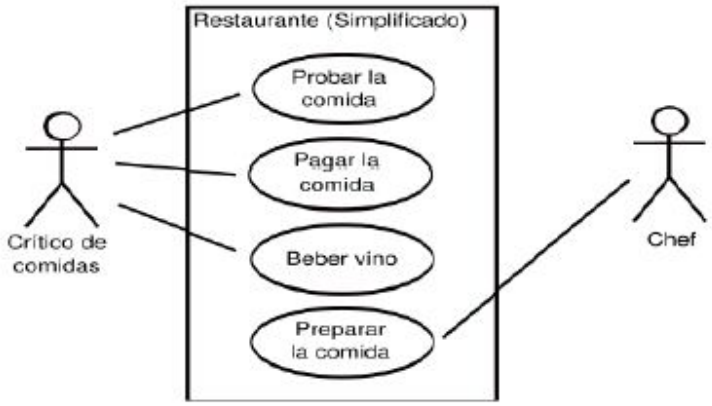
- La descripción escrita del comportamiento del sistema al afrontar una tarea de negocio o un requisito de negocio. Esta descripción se enfoca en el valor suministrado por el sistema a entidades externas tales como usuarios humanos u otros sistemas.
- La posición o contexto del caso de uso entre otros casos de uso, dado que es un mecanismo de organización, un conjunto de casos de uso coherente, consistente promueve una imagen fácil del comportamiento del sistema, un entendimiento común entre el cliente/propietario/usuario y el equipo de desarrollo.

Es práctica común crear especificaciones suplementarias para capturar detalles de requisitos que caen fuera del ámbito de las descripciones de los casos de uso. Ejemplos de esos temas incluyen rendimiento, temas de escalabilidad/gestión, o cumplimiento de estándares.

La Figura 8 describe la funcionalidad de un *Sistema Restaurante* muy simple. Los casos de uso están representados por elipses y los actores están representados por las *figuras humanas*. El actor Crítico de comidas puede Probar la comida, Pagar la comida, o Beber vino. Sólo el actor Chef puede Preparar la comida. Podría ser que ambos Patrón y Cajero estén involucrados en el caso de uso Pagar la comida. El marco define los límites del sistema Restaurante, por ejemplo, los casos de uso se muestran como parte del sistema que está siendo modelado, los actores no.

La interacción entre actores no se ve en el *diagrama de casos de uso*. Si esta interacción es esencial para una descripción coherente del comportamiento deseado, quizás los límites del sistema o del caso de uso deban de ser reexaminados. Alternativamente, la interacción entre actores puede ser parte de suposiciones usadas en el caso de uso. Sin embargo, los actores son una especie de rol, un usuario humano u otra entidad externa pueden jugar varios papeles o roles. Así el Chef y el Cajero podrían ser realmente la misma persona.

Figura 8. Ejemplo de Diagrama de Casos de Uso.



Relaciones de Casos de Uso

Las tres relaciones principales entre los casos de uso son soportadas por el estándar UML, el cual describe notación gráfica para esas relaciones.

- *Inclusión (include o use):* Es una forma de interacción, un caso de uso dado puede "incluir" otro. El primer caso de uso a menudo depende del resultado del caso de uso incluido. Esto es útil para extraer comportamientos verdaderamente comunes desde múltiples casos de uso a una descripción individual, desde el caso de uso que lo incluye hasta el caso de uso incluido, con la etiqueta "«include»". Este uso se asemeja a una expansión de una macro, donde el comportamiento del caso incluido es colocado dentro del comportamiento del caso de uso base. No hay parámetros o valores de retorno.

- *Extensión (Extend):* Es otra forma de interacción, un caso de uso dado, (la extensión) puede extender a otro. Esta relación indica que el comportamiento del caso de uso extensión puede ser insertado en el caso de uso extendido bajo ciertas condiciones. La notación, es una flecha de punta abierta con línea discontinua, desde el caso de uso extensión al caso de uso extendido, con la etiqueta «extend». Esto puede ser útil para lidiar con casos especiales, o para acomodar nuevos requisitos durante el mantenimiento del sistema y su extensión.

  "La extensión, es el conjunto de objetos a los que se aplica un concepto. Los objetos de la extensión son los ejemplos o instancias de los conceptos."

- *Generalización:* En la tercera forma de relaciones entre casos de uso, existe una relación generalización/especialización. Un caso de uso dado puede estar en una forma especializada de un caso de uso existente. La notación es una línea sólida terminada en un triángulo dibujado desde el caso de uso especializado al caso de uso general. Esto se asemeja al concepto orientado a objetos de sub-clases, en la práctica puede ser útil factorizar comportamientos comunes, restricciones al caso de uso general, describirlos

una vez, y enfrentarse a los detalles excepcionales en los casos de uso especializados.

"Entonces la Generalización es la actividad de identificar elementos en común entre conceptos y definir las relaciones de una superclase (concepto general) y subclase (concepto especializado). Es una manera de construir clasificaciones taxonómicas entre conceptos que entonces se representan en jerarquías de clases. Las subclases conceptuales son conformes con las superclases conceptuales en cuanto a la intensión y extensión."

3.6.3 *Diagramas de Secuencias:*

Un diagrama de secuencia muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical que representa el rol durante cierto plazo de tiempo, con la interacción completa. Los mensajes se muestran como flechas entre las líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de una transacción.

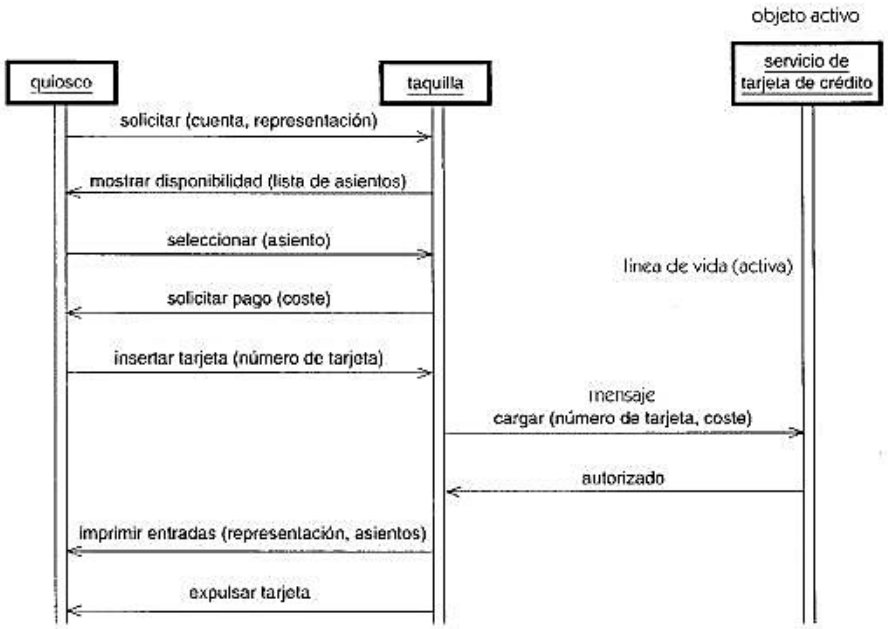
Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso del uso. Cuando está implementado el comportamiento, cada mensaje en un diagrama de secuencia corresponde a una operación en una clase, a un evento disparador, o a una transición en una máquina de estados.

La figura 9 muestra un diagrama de secuencia para el caso de uso de ***comprar entrada***. Este caso de uso lo inicia el cliente en el quisco, comunicándose con la taquilla. Los pasos para el caso de uso ***hacer cargos*** se incluyen en la secuencia, que implica la comunicación con el quiosco y el servicio de la tarjeta de crédito. Este diagrama de secuencia está en una primera etapa de desarrollo y no muestra los detalles completos de la interfaz de usuario.

3.6.4 *Diagramas de Actividades:*

En el Lenguaje de Modelado Unificado, un diagrama de actividades representa los flujos de trabajo paso a paso de negocio y operacionales de los componentes en un sistema. Un Diagrama de Actividades muestra el flujo de control general.

Figura 9. Ejemplo de Diagrama de Secuencia.

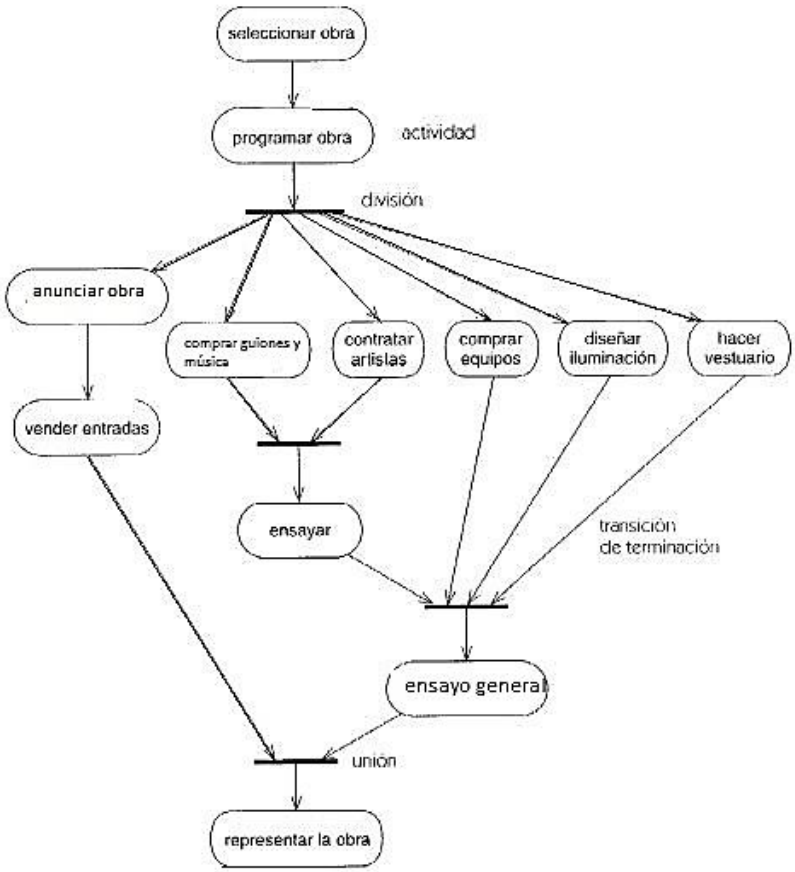


En SysML (Systems Modeling Language) el diagrama de Actividades ha sido extendido para indicar flujos entre pasos que mueven elementos físicos (e.g., gasolina) o energía (e.g., presión). Los cambios adicionales permiten al diagrama soportar mejor flujos de comportamiento y datos continuos.

La especificación del Lenguaje de Modelado Unificado OMG define un diagrama de actividad como: "… una variación de una máquina estados, lo cual los estados representan el rendimiento de las acciones o subactividades y las transiciones se provocan por la realización de las acciones o subactividades."

El propósito del diagrama de actividad es modelar un proceso de flujo de trabajo (workflow) y/o modelar operaciones. Una Operación es un servicio proporcionado por un objeto, que está disponible a través de una interfaz. Una Interfaz es un grupo de operaciones relacionadas con la semántica.

Figura 10. Ejemplo de Diagrama de Actividades.



Esta figura muestra el diagrama de actividades implicadas en montar una obra. Las flechas muestran dependencias secuenciales. Las barras horizontales representan bifurcaciones o uniones de control. En este ejemplo se modelan los procesos reales de una organización humana. El modelado de tales negocios es un propósito importante de los diagramas de actividades, Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes.

Los parámetros de entrada y de salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado del flujo del objeto.

4 MARCO METODOLOGICO

A continuación se hace una descripción y análisis del método que se empleó en el estudio y desarrollo del proyecto.

Las metodologías de desarrollo de software indican como construir técnicamente el software. Estas abarcan tareas que incluyen análisis de requisitos, diseño, construcción del programa, pruebas y mantenimiento.

Para el desarrollo del sistema de información para optimizar el proceso del sistema de costos universitarios de la Universidad Industrial de Santander, se usó cómo metodología de desarrollo el modelo de Prototipado Evolutivo.

4.1 MODELO DE PROTOTIPADO EVOLUTIVO

El término *prototipo* se refiere a un modelo que funciona para una aplicación de sistemas de información. El prototipo no contiene todas las características o lleva a cabo la totalidad de las funciones necesarias del sistema final.

Más bien incluye elementos suficientes para permitir a las personas utilizar el sistema propuesto para determinar qué les gusta, qué no les gusta e identificar aquellas características que deben cambiarse o añadirse. El proceso de desarrollo y empleo de un prototipo tiene cinco características.

- El prototipo es una aplicación que funciona.
- La finalidad del prototipo es probar varias suposiciones formuladas por analistas y usuarios con respecto a las características requeridas del sistema.
- Los prototipos se crean con rapidez.

- Los prototipos evolucionan a través de un proceso iterativo.
- Los prototipos tienen un costo bajo de desarrollo.

Tabla 1 se muestran cinco condiciones para saber la conveniencia de usar el modelo de Prototipado:

Tabla 1. Aplicaciones para Candidatos Modelo de Prototipado Evolutivo.

Aplicaciones para candidatos		
Condiciones:		**Descripción:**
1	**No se conocen los requerimientos.**	La naturaleza de la aplicación es tal que existe poca información disponible con respecto a las características que debe tener el sistema para satisfacer los requerimientos de los usuarios.
2	**Los requerimientos necesitan evaluarse.**	Se conocen los requerimientos aparentes de la información, tanto de los usuarios finales como de la organización, pero es necesario verificarlos y evaluarlos.
3	**Costos altos.**	La inversión de recursos financieros y humanos así como el tiempo necesario para generar la aplicación es sustancial.
4	**Alto riesgo.**	La evaluación inexacta de los requerimientos del sistema o el desarrollo incorrecto de una aplicación ponen en peligro a la organización, a sus empleados y también a sus propios recursos.
5	**Nueva tecnología.**	El deseo de instalar nueva tecnología ya sea en los campos de la computación, de las comunicaciones de datos u otras áreas relacionadas, abre nuevas fronteras para la organización.

Cuando se presente cualquiera de las situaciones descritas en la anterior tabla, siempre debe considerarse el desarrollo de un prototipo como posible método que beneficie a todas las partes interesadas, en nuestro caso los principales motivos de escoger este modelo para el desarrollo del sistema de información para optimizar el proceso del sistema de costos universitarios de la Universidad Industrial de Santander fueron las condiciones dos y cinco, ya que este método permite que el usuario participe directamente en el proceso de desarrollo y reduce el tiempo de desarrollo comparado con otros métodos por otra parte permite la verificación y evaluación de los requerimientos y/o funciones finales que ya están planteadas en el software SICU versión 1.0; un paradigma de construcción por prototipos puede ofrecer el mejor enfoque, ya que la entrega de prototipos, que hacen parte integral del proyecto en su conjunto, permitirán la corrección temprana de errores o la redefinición del sistema en caso de ser necesario, y los prototipos funcionales permitirán la familiarización del usuario con el sistema que se está desarrollando.

4.1.1 Estructura del Prototipado Evolutivo:

El desarrollo de un prototipo para una aplicación se lleva a cabo en una forma ordenada tal como se muestra en la Figura 11.

Identificación de requerimientos conocidos: La determinación de los requerimientos de una aplicación es muy importante, por consiguiente antes de crear el prototipo, los analistas y usuarios deben trabajar juntos para identificar los requerimientos conocidos que tienen que satisfacerse. Para hacerlo determinan los fines para los que servirá el sistema y el alcance de sus capacidades. En general, sólo un analista de sistemas es el que coordina este paso.

Desarrollo de un modelo de trabajo: La construcción de un prototipo es un proceso iterativo de desarrollo. Antes de la primera iteración, los analistas de sistemas explican el método a los usuarios, las actividades a realizar, la secuencia en que se llevarán a cabo y también discuten las responsabilidades de cada participante.

Para comenzar la primera iteración, usuarios y analistas identifican de manera conjunta los datos que son necesarios para el sistema y especifican la salida que

debe producir la aplicación. Las decisiones de diseño necesarias para desarrollar la salida del sistema cambian muy poco en relación con las tomadas en otros métodos de desarrollo. Sin embargo, con un prototipo, se espera que las especificaciones iniciales estén incompletas.

En el desarrollo de un prototipo se preparan los siguientes componentes:

- El lenguaje para el diálogo o conversación entre el usuario y el sistema.
- Pantallas y formatos para la entrada de datos.
- Módulos esenciales de procesamiento.
- Salida del sistema.

El prototipo y el usuario

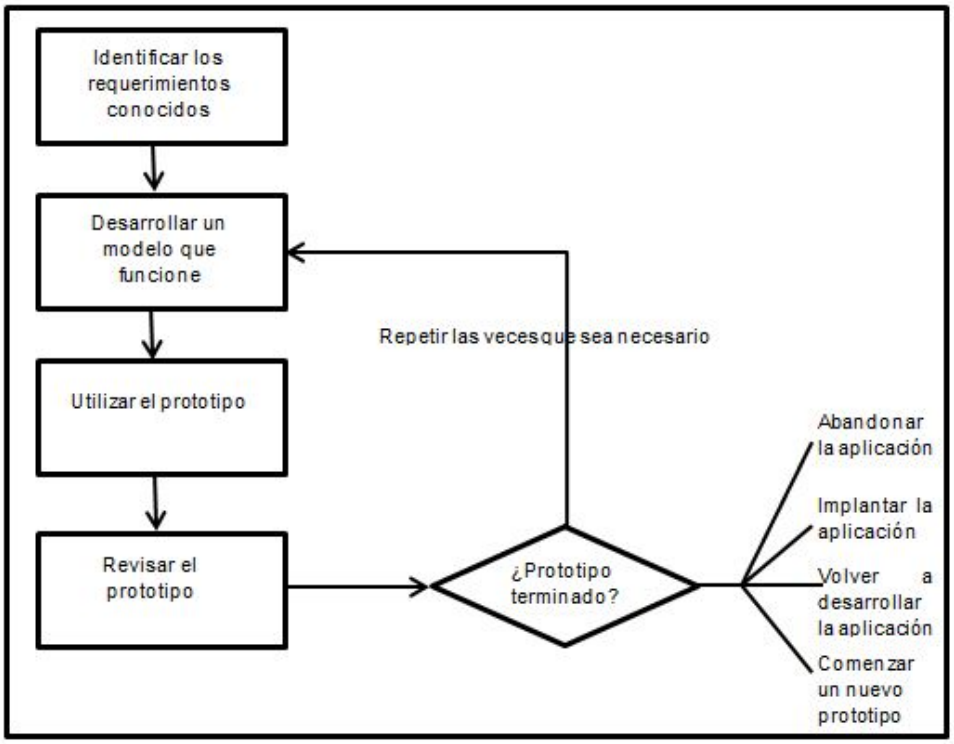
Es responsabilidad del usuario trabajar con el prototipo y evaluar sus características y operación. La experiencia con el sistema bajo condiciones reales permite obtener la familiaridad indispensable para determinar los cambios o mejoras que sean necesarios así como la eliminación de características inadecuadas o innecesarias.

Revisión del Prototipo: Durante la evaluación los analistas de sistemas desean capturar información sobre lo que les gusta y lo que les desagrada a los usuarios; al mismo tiempo ponen atención al por qué reaccionan los usuarios en la forma en que lo hacen. La información obtenida tendrá influencia sobre las características de la siguiente versión de la aplicación. Los cambios al prototipo son planificados con los usuarios antes de llevarlos a cabo, con el analista como responsable de las modificaciones.

Repetición del proceso las veces que sea necesario: Cuando el prototipo está terminado, el siguiente paso es tomar la decisión sobre cómo proceder después de evaluar la información obtenida con el desarrollo y uso del prototipo que pueden ser cuatro caminos: descartar el prototipo y abandonar el proyecto de aplicación,

implantar el prototipo, volver a desarrollar la aplicación o comenzar con otro prototipo.

Figura 11. Diagrama de Flujo de la Estructura del Modelo de Construcción de Prototipos.



4.1.2 Ventajas del Prototipado Evolutivo

- *Aumento en la productividad:* La productividad es importante para los analistas de sistemas y para la organización en la que trabaja. La productividad, cuando se aplica al desarrollo de sistemas, significa llevar a cabo las actividades en forma más eficiente, obteniendo el mayor impacto con la mejor utilización de los recursos.

- *Es eficaz para aclarar los requerimientos de los usuarios:* El desarrollo y uso de un prototipo puede ser un camino muy eficaz para identificar y aclarar los requerimientos que debe satisfacer una aplicación.
- *Verificar la factibilidad del diseño del sistema:* Los analistas pueden experimentar con diferentes características de la aplicación y evaluar la reacción y respuesta por parte del usuario; crear un prototipo y evaluar el diseño por medio de su uso, mostrará la factibilidad del diseño o sugerirá la necesidad de encontrar otras opciones.

4.1.3 Desventajas del Prototipado Evolutivo

- Los usuarios tienen gran dificultad para especificar con anticipación sus necesidades de información, en especial cuando la situación es nueva o cambia con rapidez.
- La especificación completa de los requerimientos de información depende en particular de la forma en que debe utilizarse la tecnología.
- A menudo las descripciones estáticas de sistemas (por ejemplo en documentos o gráficas) no son suficientes para proporcionar detalles sobre situaciones dinámicas.
- La mala comunicación, que siempre es una posibilidad, parece que siempre se presenta en el momento menos oportuno.

4.1.4 Modelo Aplicado para la Construcción de Prototipos: Describiendo brevemente el Modelo de Construcción de Prototipos es:

a) *Escuchar:* Hace alusión a la definición de los objetivos globales para el software entre el desarrollador y los usuarios, identificando los requisitos básicos (procesos básicos, entrada/salida de la información que se maneja, controles a tener en cuenta, tipo del sistema), requisitos transaccionales (los tipos de transacciones y características de los mismos), requisitos de decisión de usuario (Información necesaria para la toma de decisiones) y requisitos de interfaz (estilos de interfaz de acuerdo al Portal Web, Colores

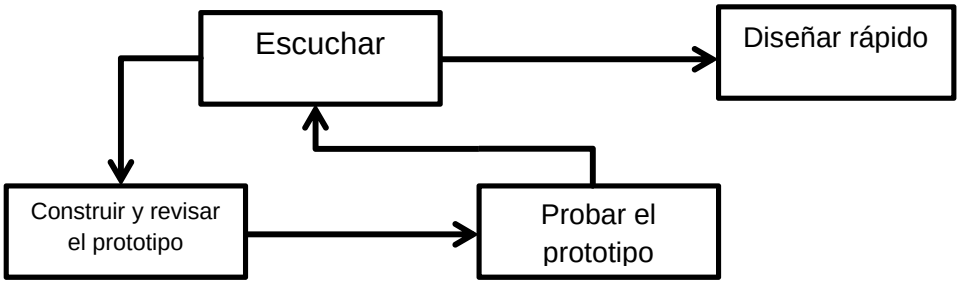
Institucionales, Información a generarse y recibirse), como también las áreas del esquema en donde es necesaria más definición. Posteriormente se plantea con rapidez una iteración de construcción de prototipos y se presenta el modelado (en forma de un diseño rápido).

b) *Diseñar Rápido:* Se centra en una representación de aquellos aspectos del software que serán visibles para el usuario final (por ejemplo, la configuración de la interfaz con el usuario y el formato de los despliegues de salida). El diseño rápido conduce a la construcción del primer prototipo.

c) *Construir y Revisar el Prototipo:* Se refiere a las fases de análisis, diseño e implementación de cada uno de los prototipos.

d) *Probar el Prototipo:* Se hacen las pruebas básicas antes de colocar en marcha en el Servidor, observando si cumple con los requisitos preestablecidos y si es necesario corregir algún defecto.

Figura 12. Modelo Construcción de Prototipos



La iteración ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

4.2 PROCEDIMIENTO PARA LA METODOLOGÍA PLANTEADA

- Se establece una cita con las directivas de Planeación y personal a cargo del proyecto para el comienzo del levantamiento de requisitos.

- En la especificación de requisitos se identifican y verifican los requisitos básicos, transaccionales, de decisión de usuarios e interface de acuerdo a la versión anterior de SICU, al Portal Web y los estándares manejados en la División de Servicios de Información (DSI).

- Luego se produce el Diseño del Prototipo que se enfoca sobre la representación de los aspectos del software visibles al usuario (por ejemplo, métodos de entrada y formatos de salida) y se prosigue a su construcción.

- El prototipo es evaluado por el usuario y se utiliza para refinar los requisitos del software a desarrollar.

- Se produce un proceso iterativo en donde al prototipo se le hace el refinamiento necesario para satisfacer las necesidades del usuario, de tal manera que facilita al que lo desarrolla una mejor comprensión de lo que hay que hacer y poder entregar el producto final requerido.

4.3 PLAN DE TRABAJO

Para el desarrollo del proyecto planteado usando el modelo de Construcción de Prototipos se establece un plan de trabajo el cuál se llevará por fases y se explica a continuación:

4.3.1 Fase de Acoplamiento: Esta fase consiste en la realización de reuniones previas de usuarios con el grupo desarrollador, para dar a conocer los alcances del proyecto y definir los objetivos.

- Se realizará el planteamiento de prioridades sobre las cuales se va a trabajar y establecer la metodología y reglas de trabajo a utilizar dentro de la División de Servicios de Información.

4.3.2 Fase de Recolección y Análisis de Requerimientos: Se procederá con reuniones periódicas con los funcionarios de Planeación de la Universidad Industrial

de Santander, con el objetivo de establecer las necesidades que se quieren satisfacer con el desarrollo del proyecto.

- En esta Fase se debe establecer todo lo que el sistema debe realizar, y todas las restricciones sobre la funcionalidad que se deben tener en cuenta.
- Se especificarán el conjunto completo de resultados a ser obtenidos al utilizar el sistema.
- La importancia de esta fase radica en que se debe mostrar a los desarrolladores y a los usuarios qué se necesita del sistema, en un lenguaje que todos comprendan, por lo tanto es primordial la excelente comunicación entre usuarios y desarrollador.
- Al terminar esta fase se debe tener un modelo completo que represente el sistema total en algún nivel de abstracción mental que puede ser especificada a nivel de detalle en un documento denominado “Especificación de Requerimientos”.

4.3.3 Fase de Diseño del Prototipo: Teniendo la especificación de los requerimientos en un lenguaje natural a nivel de detalle, se procede entonces a abstraer esa información y traducirlo en lenguaje de modelado unificado (UML).

- Dentro de ésta fase se definirá la interfaz del usuario y la interfaz de comunicación entre los programas.
- En esta etapa se deberá evaluar el grado de aceptación y satisfacción del usuario frente a los modelos de entrada y salida de datos y la interfaz de usuario que se ha propuesto y está dentro de los estándares de la División de Servicios de Información.
- Se podrá volver a llegar a esta fase durante el proceso de refinamiento del prototipo en caso de ser necesarias mejoras en los diseños planteados en el primer prototipo.

4.3.4 ***Fase de Desarrollo del Prototipo:*** Durante esta fase se realizará la programación de los requerimientos previamente modelados.

- Se llegará a esta fase cada vez que sea necesario una nueva iteración o un nuevo proceso de refinamiento del sistema, en donde se deberá implementar el nuevo prototipo.

4.3.5 ***Fase de Refinamiento del Prototipo:*** Dentro de esta fase se llevará a cabo la corrección de las fallas que surgieron tras la realización de las pruebas funcionales de las secciones de automatización de los módulos o de la realización de la evaluación por parte del usuario en cuanto al prototipo y su aceptación o de nuevas sugerencias para un mejoramiento o maduración del prototipo.

- Se realizarán los ajustes y refinación del funcionamiento de los servicios, regresando a las fases de diseño y desarrollo hasta obtener el producto final.
- En todo caso, se iniciará una nueva iteración partiendo de este punto, cuando el usuario considere que se deben hacer mejoras, o atender sugerencias para la maduración del prototipo.

4.3.6 ***Fase de Entrega del Producto Final:*** Se entrega al usuario interesado el prototipo de la nueva versión del Sistema de Información de Costos Universitarios plenamente funcional, cumpliendo todas las características inicialmente planteadas, verificadas y aprobadas por el usuario.

5 APLICACIÓN DE LA METODOLOGÍA

5.1 ACOPLAMIENTO, RECOLECCIÓN Y ÁNALISIS DE REQUERIMIENTOS

Se analizó la versión del sistema de costos universitarios usada hasta el momento apoyándonos con el manual operativo y comprendiendo cada una de las funcionalidades de este para así establecer lo que realmente ofrece el sistema.

En esta etapa se llevaron a cabo dos actividades de gran importancia para el desarrollo del software, estas son: La comprensión general del software y la captación de algunos nuevos requerimientos, en cada una de estas etapas fue muy significativa la colaboración de los usuarios.

Para el entendimiento del sistema que se estaba utilizando en Planeación, se organizaron y cumplieron varias visitas en las cuales junto a los usuarios, se analizaron todas las opciones que tenía el sistema comprendiendo sus resultados. Después de lograr entender completamente las funcionalidades del sistema, era de vital importancia, identificar cuál de estas opciones eran utilizadas, cuáles inservibles, cuáles no se sabían que hacían y qué otras necesidades que no estuviesen implementadas se necesitaban tener en cuenta para la nueva versión. Con el fin de cumplir con lo propuesto se definió entre los desarrolladores y los usuarios las opciones que se debían conservar, eliminar, así como, por parte de los desarrolladores su aporte al sistema en ideas en aras de optimizar el ya existente.

5.2 DISEÑO DEL SISTEMA

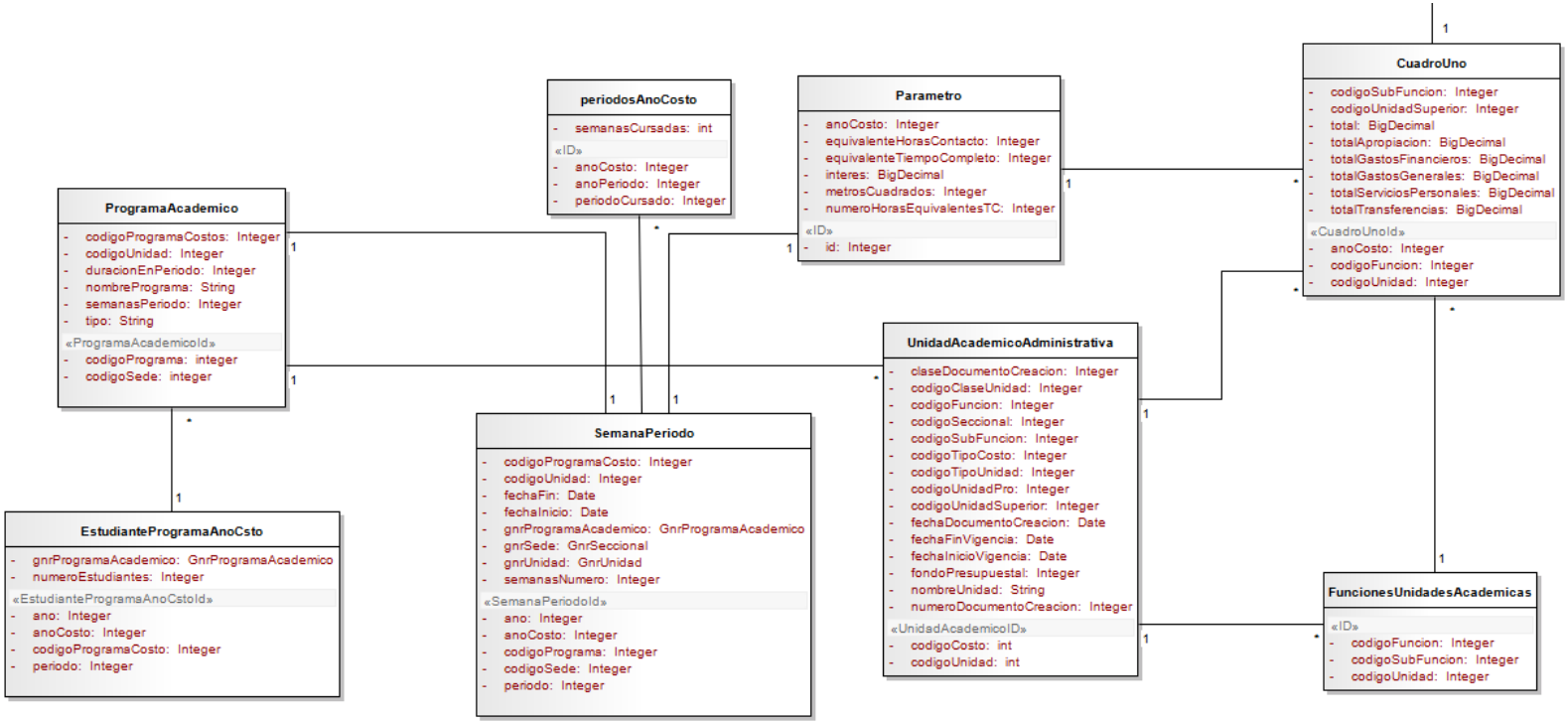
En el proceso de análisis y diseño del sistema de información se construyeron los diagramas UML, durante el transcurso del proyecto, el diseño se fue adaptando a medida que se generaban cambios luego de presentar el prototipo inicial.

5.2.1 Elaboración del diagrama del diagrama de clases:

A continuación se presenta un fragmento importante, aunque simplificado, del modelo de datos del sistema. En la capa de la lógica del negocio, no se utilizan tablas sino

clases. El sistema relaciona las dos a través de un ‘mapeo’ de los datos. Este es un diagrama de clases parcial de la nueva versión del sistema de costos universitarios de la UIS.

Figura 13. Diagrama de clases del SICU.



Clase estudiante programa año costo: Relaciona la cantidad de estudiante por programa y por año costo.

Tabla 2. Clase 'EstudianteProgramaAnoCsto'.

Nombre	Tipo de datos	Descripción
numeroEstudiantes	Integer	Número de estudiantes que están en el programa académico.
ano	Integer	Año en el cual se está cursando el programa académico.
anoCosto	Integer	Año para el cuál se va a calcular el costo.
codigoProgramaCosto	Integer	Código asignado en el sistema de costos al programa académico donde está adscrito el estudiante.
periodo	Integer	Número del periodo, puede ser uno (Enero – Junio) o dos (Julio - Diciembre).

Clase programa académico: Relaciona los programas académicos ofrecidos por la UIS en todas las sedes.

Tabla 3. Clase 'ProgramaAcademico'.

Nombre	Tipo de datos	Descripción
codigoProgramaCostos	Integer	Código asignado en el sistema de costos al programa académico donde está adscrito el estudiante.
códigoUnidad	Integer	Código de la unidad a la que pertenece el programa académico.
duracionEnPeriodo	Integer	La cantidad de periodos que dura el programa académico.
nombrePrograma	String	Nombre del programa académico.
semanasPeriodo	Integer	Cantidad de semanas que dura el periodo académico.
codigoPrograma	Integer	Código del programa académico dónde está adscrito el estudiante.
codigoSede	Integer	Código de la sede del programa académico.

Clase semana periodo: Relaciona la información más detallada, sobre el año en que se cursó el programa y cuáles fueron las semanas reales cursadas.

Tabla 4. Clase 'SemanaPeriodo'.

Nombre	Tipo de datos	Descripción
codigoProgramaCosto	Integer	Código asignado en el sistema de costos al programa académico donde está adscrito el estudiante.
fechaFin	Date	Fecha de finalización del periodo académico.
fechaInicio	Date	Fecha de inicio del periodo académico.
semanasNumero	Integer	Cantidad de semanas que se proyectó en el periodo.
ano	Integer	Año en el cual se está cursando el programa académico.
anoCosto	Integer	Año para el cuál se va a calcular el costo.
codigoPrograma	Integer	Código del programa académico dónde está adscrito el estudiante.
codigoSede	Integer	Código de la sede del programa académico.
periodo	Integer	Número del periodo, puede ser uno (Enero – Junio) o dos (Julio - Diciembre).

Clase parámetro: Relaciona los parámetros para el momento de hacer los cálculos.

Tabla 5. Clase 'Parametro'.

Nombre	Tipo de datos	Descripción
anoCosto	Integer	Año para el cuál se va a calcular el costo.
equivalenteHorasContacto	Date	Punto indicador horas contacto, número para el docente.
equivalenteTiempoCompleto	Date	Punto indicador, tiempo completo.
interes	BigDecimal	Interés que se tendrá en cuenta para aplicar en los cálculos.
metrosCuadrados	Integer	Punto indicador número de metros cuadrados.
numeroHorasEquivalentesTC	Integer	Número de horas semanales para el tiempo completo.
id	Integer	Id asignado por cada una de las tuplas que se han guardado, teniendo en cuenta siempre la última.

Clase cuadro uno: Presenta los gastos causados durante la vigencia, agrupados en rubros agregados, ocasionados por todas y cada una de las unidades de costo de la institución debidamente clasificadas en las tres funciones básicas y las de apoyo a estas funciones.

Clase funciones unidades académicas: Relaciona las funciones con las unidades.

Tabla 6. Clase 'FuncionesUnidadesAcademicas'.

Nombre	Tipo de datos	Descripción
codigoSubFuncion	Integer	Código subfunción de la unidad de costo según actividad.
codigoFuncion	Integer	Código de la función de la unidad de costo según actividad.
codigoUnidad	Integer	Código de la unidad de costo.

Clase periodos año costo: Relaciona las semanas y periodos cursados en el año costo.

Tabla 7. Clase 'PeriodosAnoCosto'.

Nombre	Tipo de datos	Descripción
semanasCursadas	Integer	Cantidad real de semanas que se cursaron en el periodo.
anoPeriodo	Integer	Concatena el año y el periodo académico.
anoCosto	integer	Año para el cuál se va a calcular el costo.
periodoCursado	Integer	Periodo académico que se cursó en el año.

5.2.2 *Elaboración de los diagramas de casos de uso*

Actores:

- *Administrador***:** Profesional de planeación vinculado a la UIS, encargado de la verificación, el control del registro y del procesamiento de la información requerida para la generación y el análisis de resultados anuales de los costos universitarios.

- *Usuario:* Hereda las propiedades del administrador.

Casos de uso

Parámetros del sistema

Figura 14. Caso de uso 'Hacer Mantenimiento Parámetros del Sistema'.

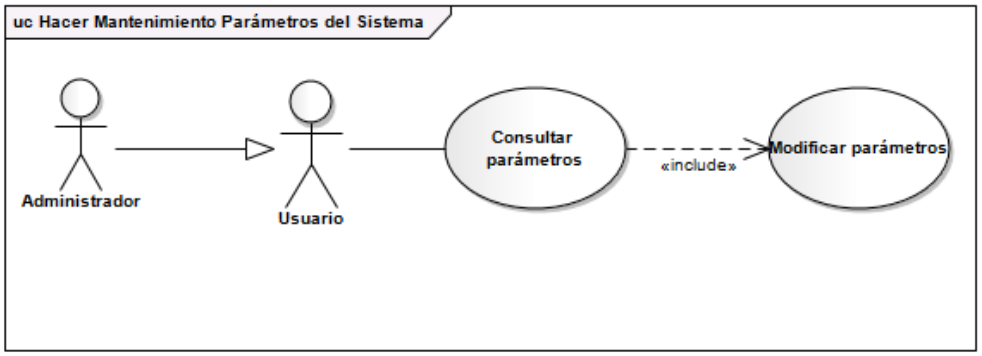


Figura 15. Captura de la interfaz 'Parámetros del sistema'.

Tabla 8. Caso de uso 'Parámetros del sistema'.

Caso de uso	Parámetros del sistema
Actores	Administrador
Propósito	Permite al actor consultar y modificar los parámetros del sistema.
Resumen	El actor consulta los parámetros del sistema antes de generar cuadros o bien puede modificarlos si no son los que requiere para la generación de cuadros, también puede cancelar la modificación.
Precondiciones	Se requiere que el actor esté validado correctamente.
Flujo principal	Se muestra un formulario en el cuál el actor observa los valores actuales de los parámetros del sistema, con opción de modificarlos o cancelar la modificación en el momento que lo esté haciendo.
Subflujos	Ninguno.
Excepciones	Dejar campos vacíos.

Factores prestacionales

Figura 16. Caso de uso 'Hacer Mantenimiento Factores Prestacionales'.

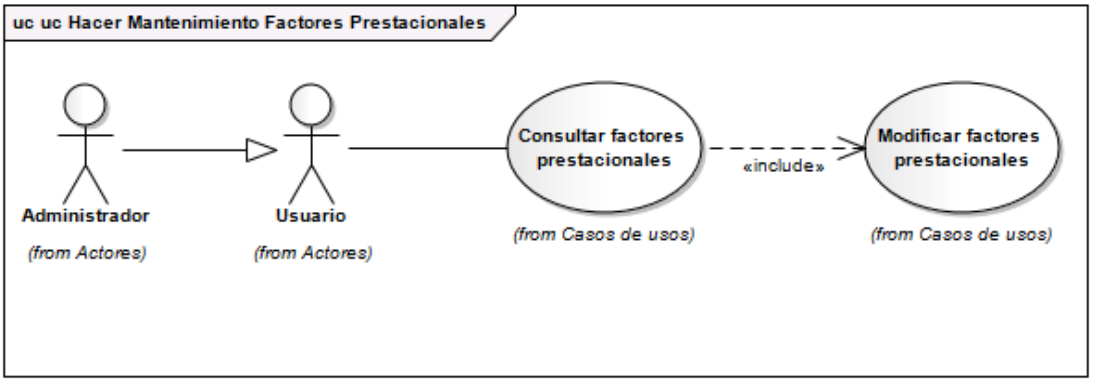


Figura 17. Captura de la interfaz 'Factores prestacionales'.

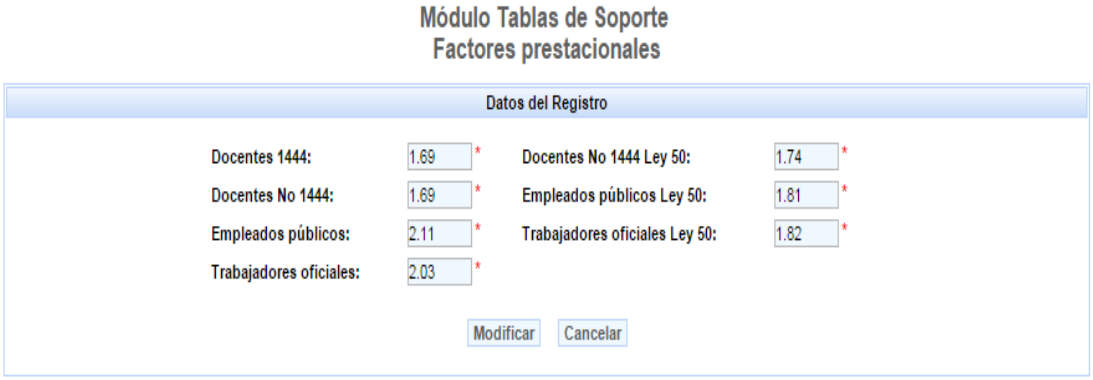


Tabla 9. Caso de uso 'Hacer Mantenimiento Factores Prestacionales'.

Caso de uso	Factores prestacionales
Actores	Administrador
Propósito	Permite al actor puede consultar y modificar los factores prestacionales de los diferentes tipos de docentes, empleados y trabajadores de la UIS.
Resumen	El actor consulta los factores prestacionales antes de generar cuadros o bien puede modificarlos si no son los que requiere para la generación de cuadros, también puede cancelar la modificación.
Precondiciones	Se requiere que el actor esté validado correctamente.
Flujo principal	Se muestra un formulario en el cuál el actor observa los valores actuales de los diferentes tipos de factores prestacionales, con opción de modificarlos o cancelar la modificación en el momento que lo esté haciendo.
Subflujos	Ninguno
Excepciones	Dejar campos vacíos.

Hacer mantenimiento conceptos gasto

Figura 18. Caso de uso 'Mantenimiento Conceptos Gasto'.

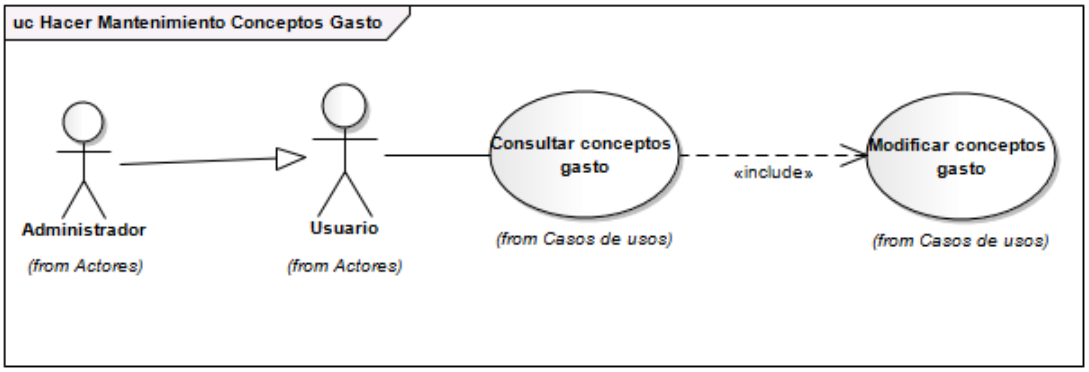


Figura 19. Captura de la interfaz 'Conceptos gasto'.

Tabla 10. Caso de uso 'Hacer mantenimiento conceptos gasto'.

Caso de uso	Hacer mantenimiento conceptos gasto.
Actores	Administrador.
Propósito	Permite al actor consultar y modificar los conceptos de gasto (Rubro agregado).
Resumen	El actor consulta los conceptos de gasto antes de generar cuadros o bien puede modificarlos si no son los que requiere para la generación de cuadros, también puede cancelar la modificación.
Precondiciones	Se requiere que el actor esté validado correctamente.
Flujo principal	Se muestra un formulario en el cuál el actor observa los valores actuales de los diferentes tipos de conceptos de gasto, con opción de modificarlos o cancelar la modificación en el momento que lo esté haciendo.
Subflujos	Ninguno.
Excepciones	Dejar campos vacíos.

Hacer mantenimiento semanas efectivas

Figura 20. Caso de uso 'Hacer Mantenimiento Semanas Efectivas'.

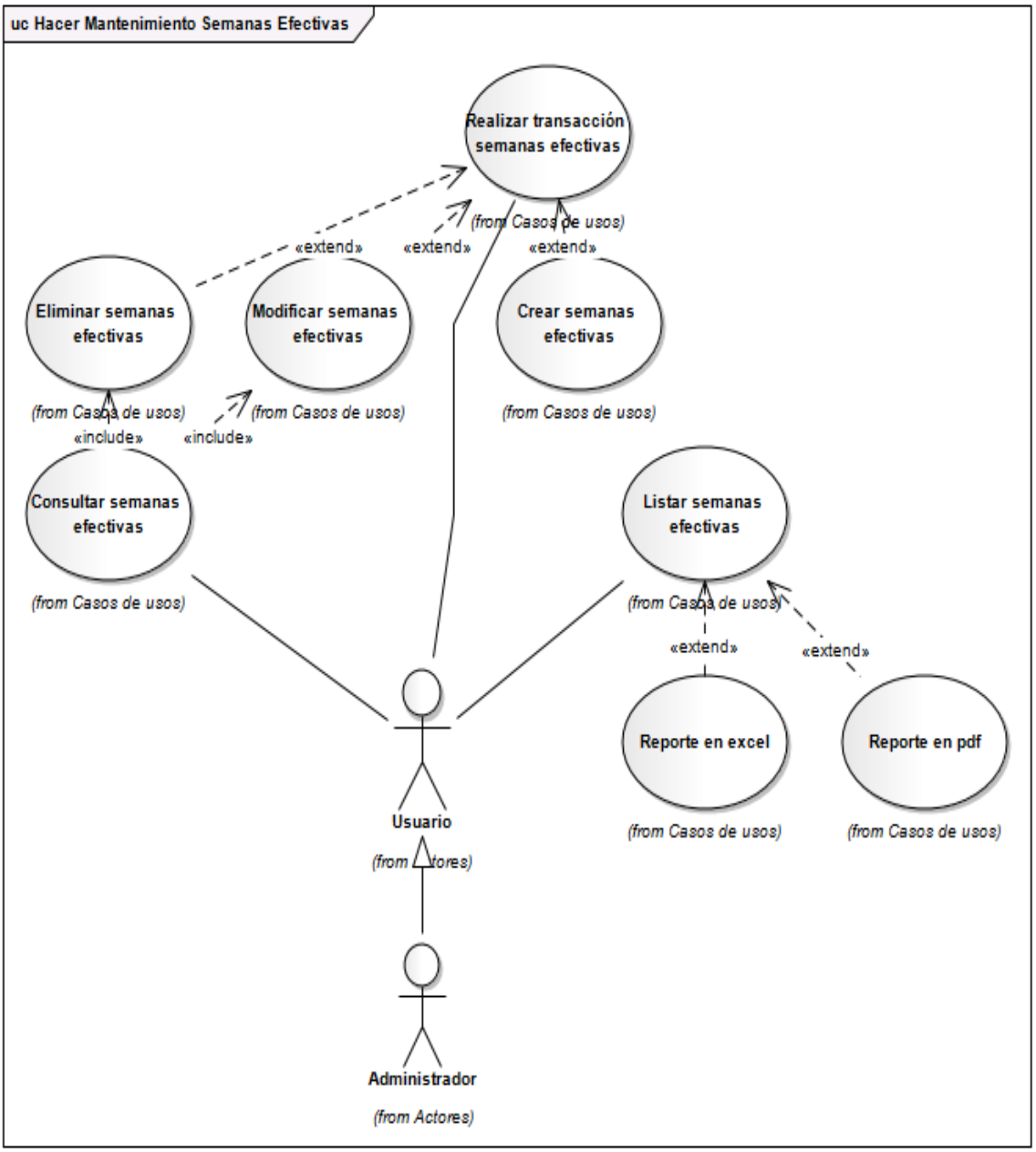


Figura 21. Captura de la interfaz 'Semanas efectivas periodo programa'.

Figura 22. Captura de la interfaz 'Editar semana efectiva'.

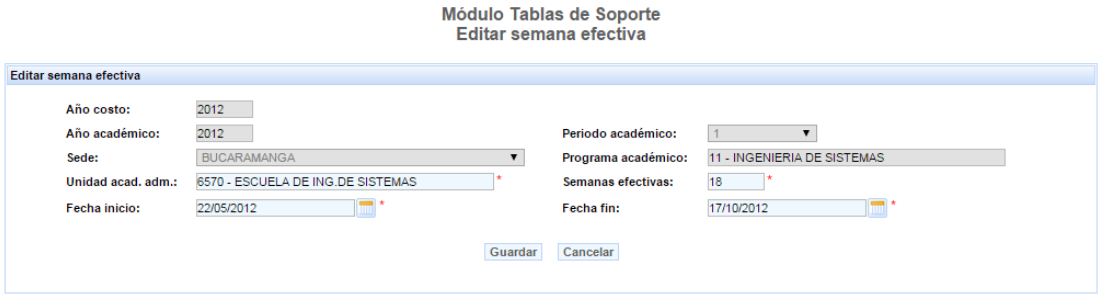


Figura 23. Captura de la interfaz 'Crear semana efectiva'.

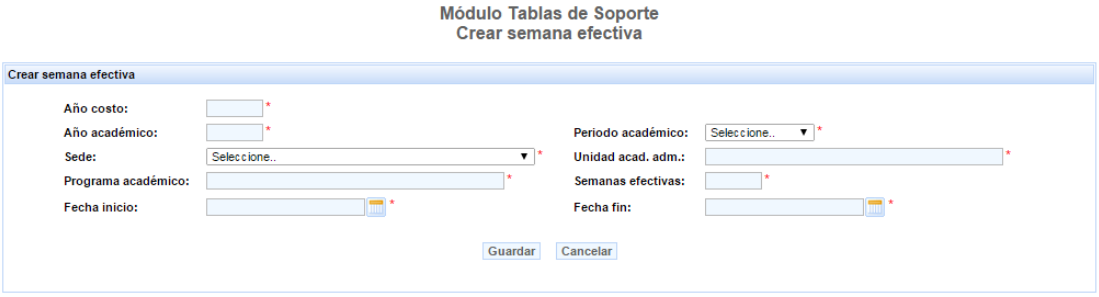


Tabla 11. Caso de uso 'Hacer Mantenimiento Semanas Efectivas'.

Caso de uso	Hacer mantenimiento semanas efectivas
Actores	Administrador
Propósito	Permite al actor consultar, modificar, crear y eliminar las semanas efectivas de los periodos del programa académico para el año costo.
Resumen	El actor consulta las semanas efectivas de cualquier periodo y programa académico para el año costo y bien puede modificarlas si no corresponden, eliminarlas o crearlas.
Precondiciones	Se requiere que el actor esté validado correctamente.
Flujo principal	Se muestra un formulario en el cuál el actor puede consultar las semanas efectivas de cualquier periodo y programa académico para el año costo.
Subflujos	**Subflujo 1**: Una vez realizada la consulta se muestra en la parte inferior de la consulta los resultados con opciones de modificar o eliminar cada uno de los resultados obtenidos, también puede exportar los resultados a Excel o a PDF. **Subflujo 2**: Modificar semanas efectivas de un programa académico para el año costo se muestra una nueva interfaz con las únicas opciones de modificar la unidad académica administrativa, la fecha de inicio y fin del periodo, y la cantidad de semanas efectivas para determinado programa académico en el año costo. **Subflujo 3:** Crear semanas efectivas de un programa académico para el año costo se muestra una nueva interfaz con las opciones de crear las semanas efectivas de un programa académico en el año costo.
Excepciones	**Excepción 1:** Dejar campos vacíos o no correctos al crear semana efectiva.

Generar Cuadros

Figura 24. Caso de uso 'Generar Cuadros'.

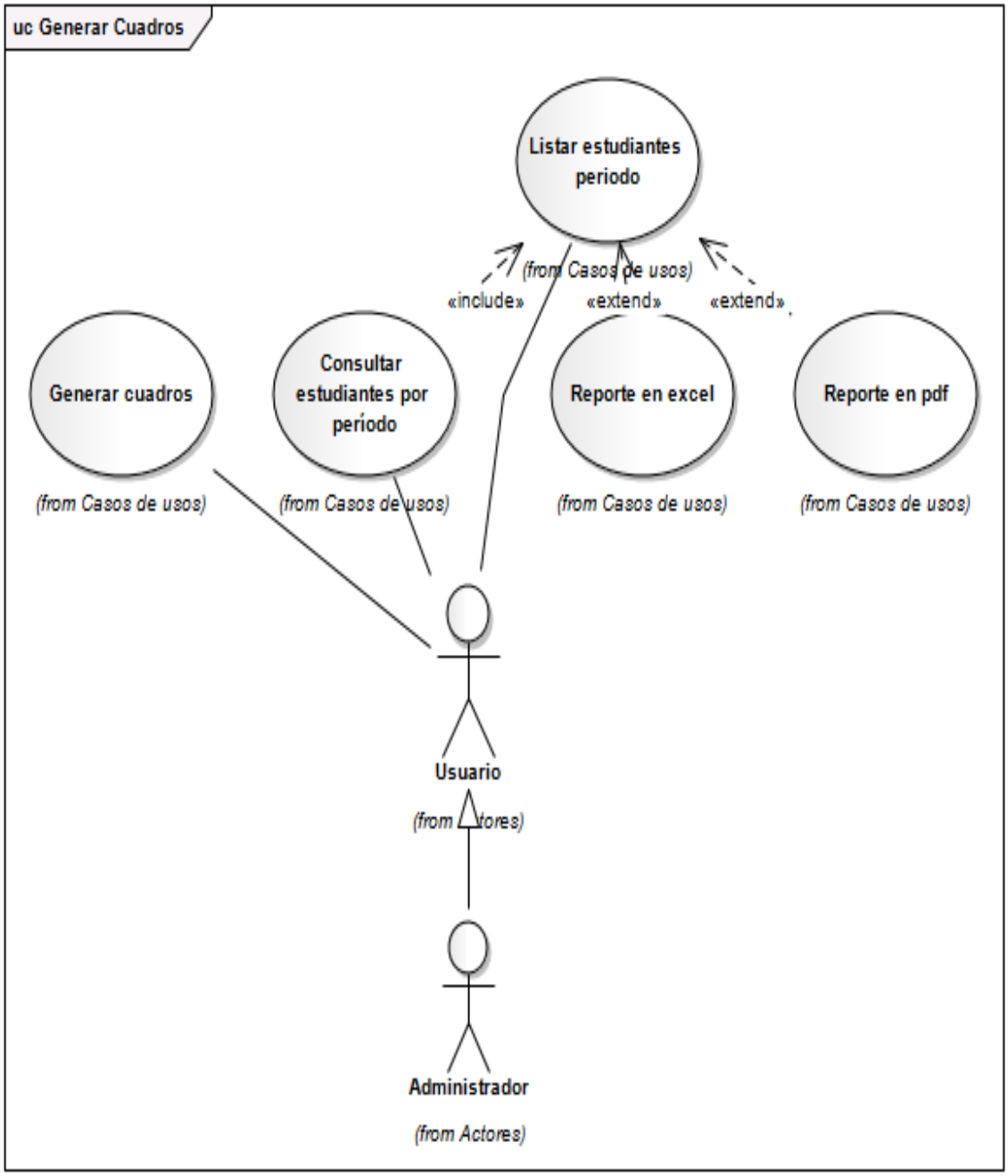
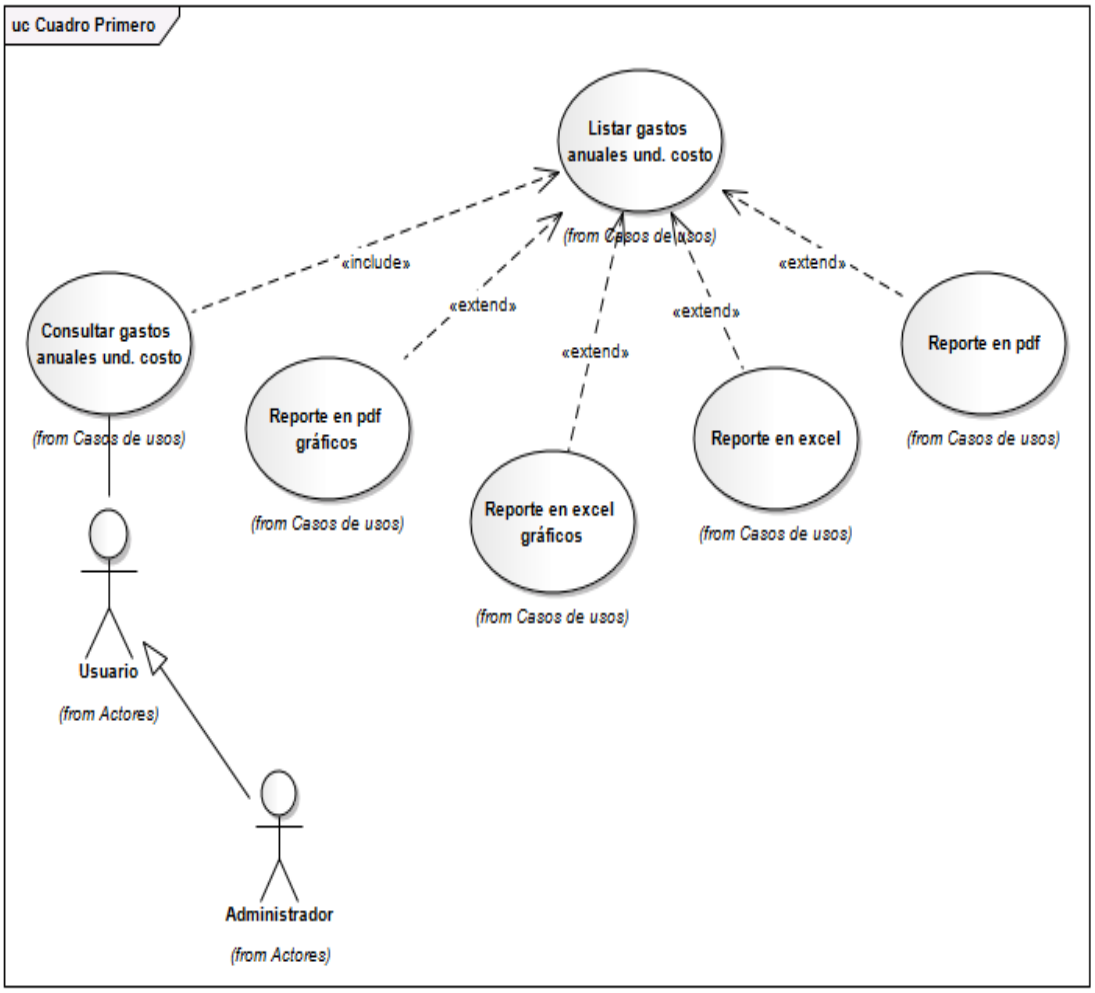


Figura 25. Caso de uso 'Generar Cuadro Primero'.

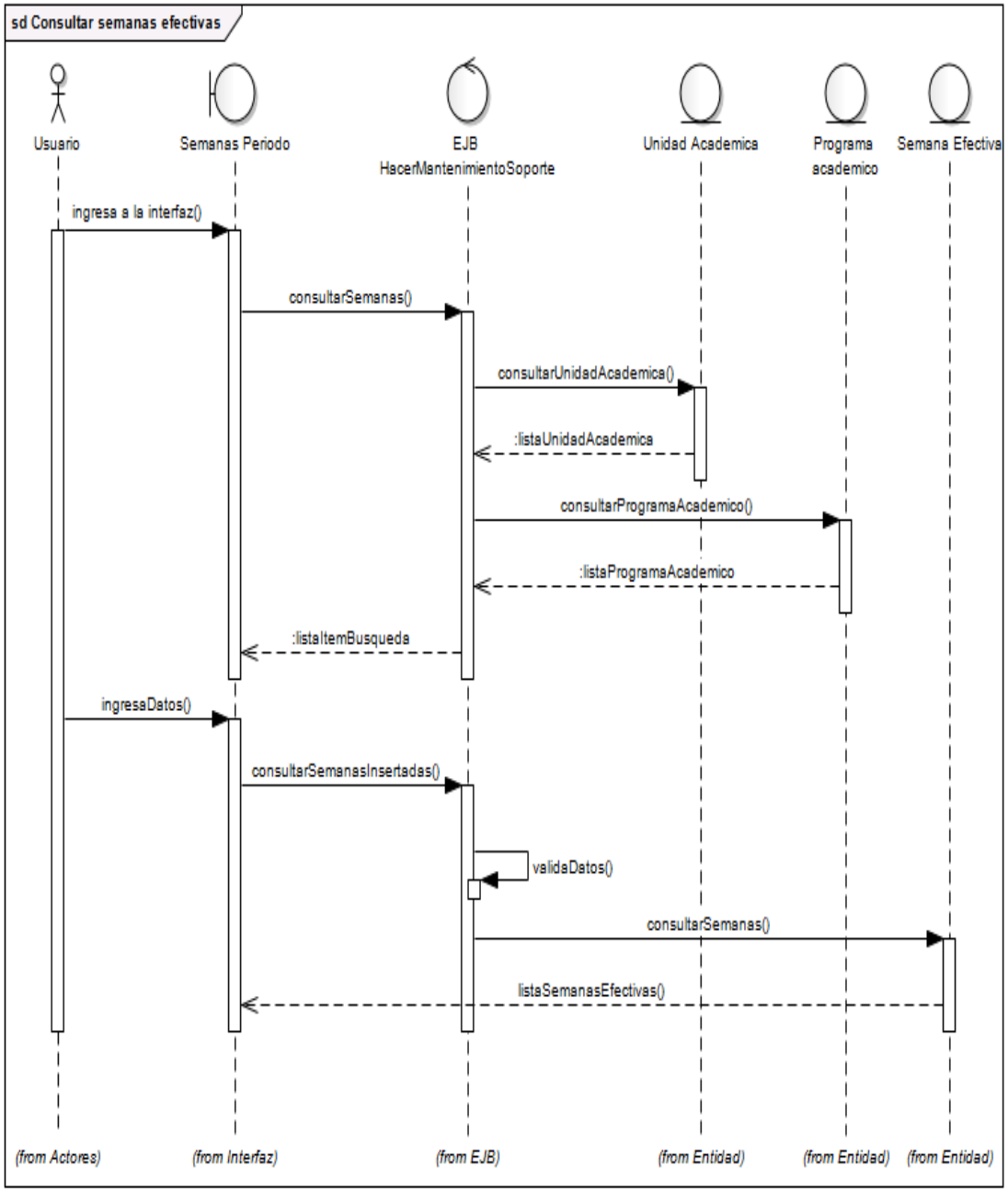


5.2.3 Elaboración de los diagramas de secuencia: Los diagramas de secuencia amplían la visión general del funcionamiento e interacción del sistema con el usuario final, visión que previamente se conformó a partir de la recolección de requerimientos del sistema.

A continuación se presentan algunos de los casos de uso representados en diagramas de secuencia y su respectiva explicación:

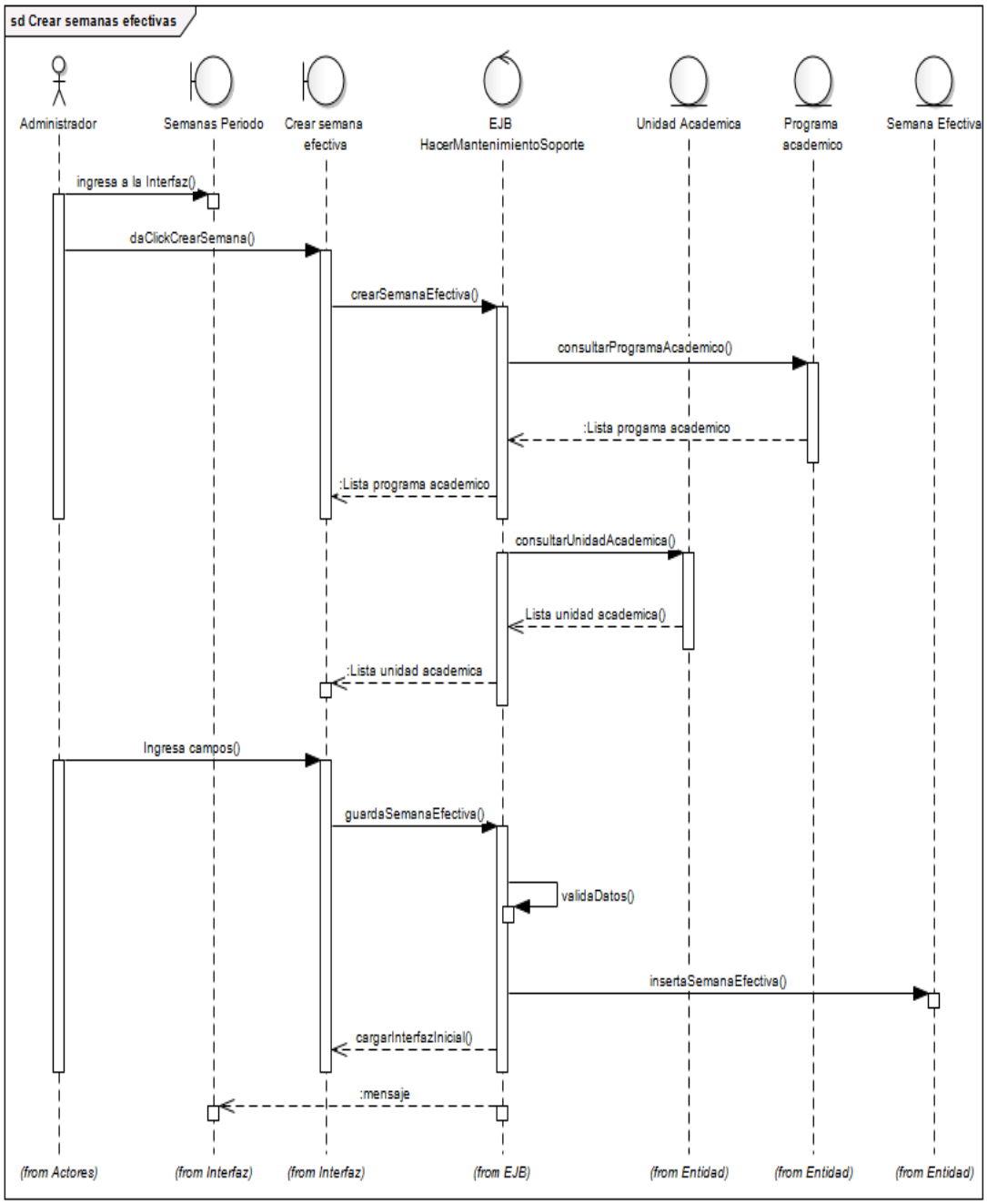
Consultar semanas efectivas.

Figura 26. Diagrama de secuencia 'Consultar semanas efectivas'.



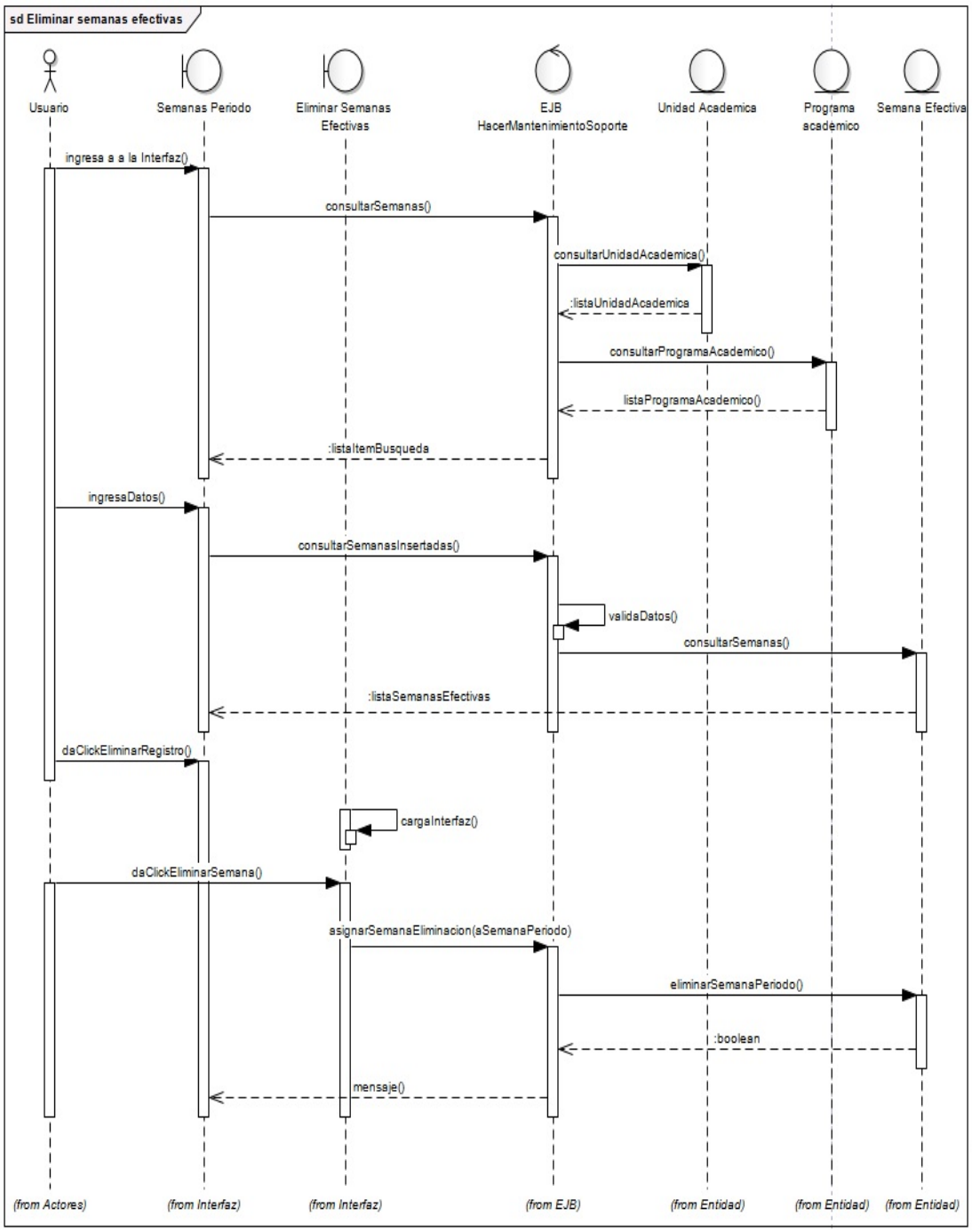
Crear semanas efectivas.

Figura 27. Diagrama de secuencia 'Crear semanas efectivas'.



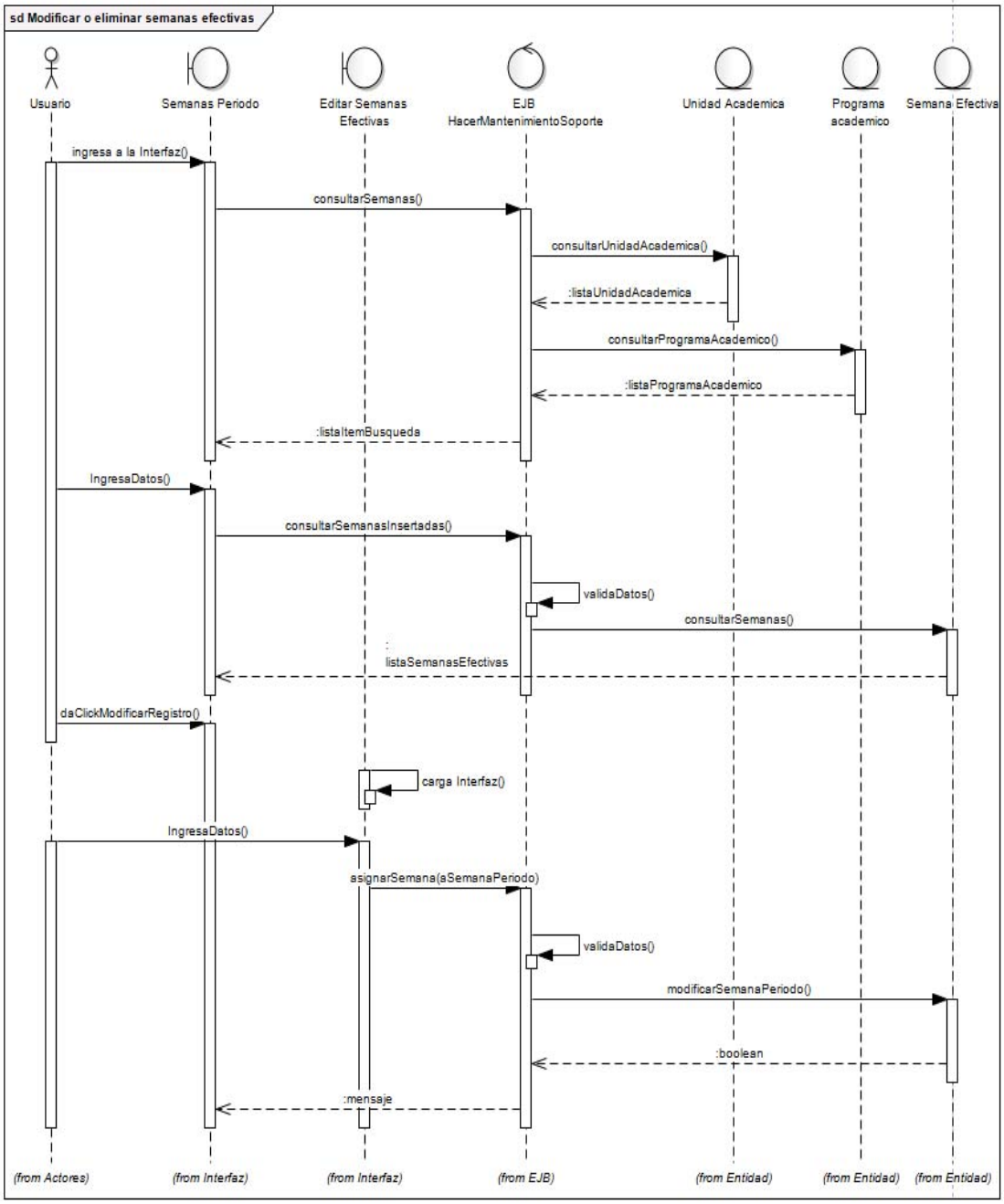
Eliminar semanas efectivas.

Figura 28. Diagrama de secuencia 'Eliminar semanas Efectivas'.



Modificar o eliminar semanas efectivas.

Figura 29. Diagrama de secuencia 'Modificar o Eliminar semanas efectivas'.



EJB

- *EJB HacerMantenimientSoporte:* Es el EJB (Entity Java Beans) dónde se hacen todos los métodos necesarios utilizados por las interfaces del caso de uso Hacer Mantenimiento Semanas Efectivas.

Entidad

- *Programa Académico:* Es la entidad dónde está toda la información de los programas académicos que ofrece la UIS. Esta entidad pertenece a la tabla: programas_acad.
- *Semana Efectiva:* Es la entidad dónde está toda la información de las semanas efectivas por programa académico que ofrece la UIS. Esta entidad pertenece a la tabla: semana_periodo_programa_ano_s.
- *Unidad Académica:* Es la entidad dónde está toda la información de las unidades académicas que tiene la UIS. Esta entidad pertenece a la tabla: uni_aca_adm_costos.

Interfaz

- *Crear semana efectiva:* Es la interfaz dónde se digitan todos los campos necesarios para crear las semanas efectivas, ver Figura 23.
- *Editar semana efectiva:* Es la interfaz dónde se muestran los campos de semana efectiva a editar, ver Figura 22.
- *Eliminar semanas:* Es la interfaz dónde se muestra la semana efectiva a eliminar.
- *Semanas Periodo:* Es la interfaz dónde se muestran los campos de consulta y los resultados de la misma en semanas efectivas, ver Figura 21.

5.3 ESTÁNDARES DE LA DIVISIÓN DE SERVICIOS DE INFORMACIÓN

5.3.1 Aspectos Generales

- *Interfaz de desarrollo:* El IDE de desarrollo a utilizar es el JBoss Developer Studio, el cual debe ser instalado en la carpeta por defecto del instalador. Este y todos los programas necesarios se pueden descargar, por el personal autorizado, del equipo establecido para tal fin.

- *Servidor de aplicaciones:* En cuanto al servidor de aplicaciones de desarrollo se debe utilizar el mismo que se encuentra en los servidores de producción y desarrollo, el cual debe ser instalado en la carpeta C:\jboss-eap-5.2.

- *JAVA:* La versión del compilador de JAVA debe ser la 1.7, la cual debe ser instalada en el directorio C:\java1.7.

- *SEAM:* La versión del seam a utilizar es la 2.1.2.GA. (jboss-seam-2.1.2.GA.zip).

  Espacio de trabajo: El espacio de trabajo se debe crear en C:\workspace.

  El nombre del proyecto para los JPA debe estar conformado de la siguiente manera:

  [Sistema]Entidades

  Por ejemplo: AcademicoEntidades (La primera letra de cada palabra en mayúscula).

  El repositorio para los JPA debe estar conformado de la siguiente manera:

  [Sistema]JPA

  Por ejemplo: AcademicoJPA (La primera letra de cada palabra en mayúscula).

  El Server name en el IDE de desarrollo se debe llamar JBoss 5 y el nombre del JBoss Runtime Enviroment se debe llamar JBoss 5.

- *Plantillas, estilos, imágenes y formateador:* Descargar del servidor de versiones de documentación las carpetas Estilos, Plantillas e Imágenes y copiarlas en la careta view de la aplicación.

- *Patrones a utilizar:* Los patrones a utilizar corresponden a cada una de las capas implementadas:

  Capa de presentación: Modelo Vista Controlador, el cual es implementado por Java Server Faces (JSF).

  Capa de lógica de negocio: Session Façade

  Capa de persistencia: Entity Access Object (EAO)

  Rich Faces: La versión a utilizar es la incluida en el jboss-seam-2.1.2.

- *Anotaciones en la entidad:* Las anotaciones en la entidad se deben colocar antes del método get correspondiente y no en la declaración del atributo.

- *Llaves compuestas:* Se debe utilizar la anotación EmbeddedId, la cual obliga a declara un objeto del tipo de la llave dentro del EJB de entidad.

- *HashCode e Equals en EJB de entidad:* Se debe utilizar únicamente los campos que conforman la llave primaria. En el caso de las llaves compuestas, el atributo que hace referencia a ésta.

- *JPA externos:* Para utilizar los JPA externos (academico.jar, recursos-humanos.jar, etc), éstos deben copiarse en la carpeta raíz. En el caso de Windows C:\, para Linux \).

  En cada uno de los archivos persistence.xml de su aplicación, se debe utilizar <jarfile>

  file:/jpa.jar</jar-file>.

  Por ejemplo:

  ```
  <jar-file>file:/academico.jar</jar-file>

  <jar-file>file:/recursos-humanos.jar</jar-file>

  <jar-file>file:/general-UIS.jar</jar-file>
  ```

- *Servicios:* Para crear un servicio se debe tener en cuenta las siguientes reglas:

  La interfaz debe contener la anotación Remote

La implementación de la interfaz debe contener la anotación RemoteBinding con su respectivo nombre jndi (igual al utilizado por la anotación Name).

Por ejemplo:

```
@RemoteBinding(jndiBinding="ConsultarGenerales")
```

Para utilizar el servicio se debe utilizar la anotación EJB indicando el nombre jndi.

De acuerdo al ejemplo anterior sería:

```
@EJB(mappedName = "ConsultarGenerales")
```

5.3.2 ***Sintaxis de Nombres en Java:*** Reglas de sintaxis generales:

En esta sección se especifican las reglas de sintaxis generales para todos los identificadores (nombres de variables, clases, métodos, etc.)

- Todos los nombres de los identificadores deben estar en español.
- Siempre se deben utilizar nombres que sean claros, concretos y libres de ambigüedades. Usando palabras completas evitando acrónimos y abreviaturas.
- Los nombres deben estar definidos sin espacios en blanco, sin guiones (_ ,-), ni comillas (“,’), sin operadores (+, - , / , *), sin tildes, utilizar la n en vez de la ñ y sin caracteres especiales.
- No se debe utilizar la mayúscula para diferenciar entre identificadores distintos. Ejemplo: contador y Contador.
- No se deben diferenciar dos identificadores solo con numerales en cualquier posición. Ejemplo: contador1, contador2, 1contador, 2contador.
- Las siguientes partículas están prohibidas en la declaración de los nombres de identificadores: artículos (el, la, los, unos, unas, un), determinantes demostrativos (este, ese, aquel, aquellos), cardinales (uno, dos, etc.), pronombres de cualquier tipo (yo, tu, el, me, te, se, este, ese, mi, tu, su, etc.).

- Se deben utilizar máximo 5 palabras por nombre, las 3 primeras palabras van completas, a partir de la cuarta palabra se quitan las vocales a la palabra exceptuando la última vocal y la primera si la palabra empieza por vocal. No se utiliza ningún separador entre las palabras, se separa cada palabra utilizando su primera letra en mayúscula.

  Ejemplo:

  `hacerMantenimientoConsultaUsros,`

  `hacerMantenimientoAsignaturasCntxto.`

Clases:

- Los nombres de las clases deben iniciar siempre en mayúscula, deben ser simples y descriptivos.
- Los nombres de las clases de Entidad deben ser sustantivos en singular.
- Para las clases de Entidad siempre se debe definir la anotación @Table que indica la tabla de la base de datos relacionada para su persistencia.
  Además se debe utilizar el parámetro name de la anotación @Entity para identificar el EJB (@Entity(name = “xxx”)). El nombre de la clase puede ser distinto al nombre de la tabla y debe seguir el estándar de identificadores mencionado en el numeral anterior.
- Los nombres de los EJB utilizados por el patrón Session Façade está conformado por un verbo autorizado. Además debe incluir al final las letras “EJB”.
  Ejemplo: `RegistrarMatriculaEstudianteEJB.`
- La interfaz asociada al EJB debe llevar el mismo nombre del EJB sin el sufijo “EJB”.
  Ejemplo: `RegistrarMatriculaEstudiante.`
- Para los EJB de entidad, cuando la clase representa una relación entre dos entidades, el nombre se forma uniendo los nombres de las entidades involucradas.

- Los nombres de las clases que representan excepciones terminaran siempre con el sufijo “EXCEPCION”.
- El nombre de la clase no contendrá detalles sobre la implementación interna de la misma. Por ejemplo `ArrayEstudiantes` no es nombre válido.

Métodos:

- Se deben utilizar los verbos autorizados para su codificación.
- El nombre de los métodos debe iniciar con minúscula la primera palabra, las siguientes palabras inician en mayúscula, sin separadores.
- La primera palabra del nombre de los métodos debe ser un verbo en infinitivo y debe representar una acción o comportamiento de la clase.
- El nombre del método debe describir claramente el comportamiento del mismo.
- En lo posible no se deben usar verbos genéricos aplicables a todo como: procesar, gestionar, manejar.
  Ejemplo: `procesarEstudiante()`, `gestionarCliente()`, en este caso el verbo no aclara el cometido real del método.

Paquetes: El nombre de los paquetes debe iniciar con minúscula la primera palabra, las siguientes palabras inician en mayúscula, sin separadores.

- La estructura de los paquetes es la siguiente:

  `co.edu.uis`.[sistema].[aplicación].[módulo].[caso de uso]

  Ejemplo:

  co.edu.uis.financiero.egresos.contratacion.ordenarPrestacionServicio.
- La estructura para el paquete donde van a estar las entidades comunes para todos los sistemas es el siguiente:

`co.edu.uis.sistema.entidades`

- La estructura para el paquete donde van a estar los servicios comunes para todos los sistemas es el siguiente:

  `co.edu.uis.sistema.servicios`

Variables: Para el nombre de las variables utilizar palabras completas en singular (máximo tres palabras). El nombre deber ir en plural cuando la variable representa una lista o un conjunto de elementos.

- Si las palabras no son suficientes para la descripción, se debe hacer un comentario, al frente de la variable.
- Las constantes (final) van en mayúscula sostenida separando las palabras por guión de piso (_).
- Para los argumentos, deben iniciar siempre con la letra “a” y posteriormente el nombre según lo establecido anteriormente.
- Para las instancias de las clases, las entidades llevan el mismo nombre de la clase, solo que la palabra inicial va en minúscula. Si se necesita más de una instancia, para diferenciarlas se debe adicionar una palabra que identifique el rol que desempeña.
  Ejemplo: `Estudiante estudiantePregrado`,
  `Estudiante estudiantePostgrado`.
- La variable del EntityManager se debe llamar em.
- La variable de tipo FacesMessages se debe llamar facesMessages.

Librerías (JAR): El nombre de las librerías no sigue el mismo estándar de las clases. Éstas se deben escribir en minúscula, separando cada palabra con un guión (-). Ejemplo:

`recursos-humanos.jar.`

Nombres de archivos: Se sigue el mismo estándar descrito en las reglas de sintaxis generales.

5.3.3 *Documentación:* La documentación relacionada con el diseño del sistema reside en la base de datos de Enterpirse Architect.

La documentación del código fuente se debe hacer en cada una de las clases, siguiendo el estándar de *JAVADOC* y documentando la definición de la clase, descripción de los métodos get y set y descripción de los parámetros de entrada y salida de cada uno de los métodos que componen la clase.

5.4 PROTOTIPO INICIAL

5.4.1 Generalidades del prototipo inicial.

De acuerdo al modelo de desarrollo implementado el prototipo es la forma más efectiva para los diseñadores y las partes interesadas para plasmar todas las ideas sobre el sistema de manera previa al desarrollo.

Se tomó atenta nota de cada uno de los requerimientos para el sistema, se realizó un correcto análisis de ellos y se comenzó a construir un prototipo no funcional que proporcionara una facilidad de exponer al cliente final, la manera en que se podría brindar una solución a los requerimientos analizados previamente.

El proceso de construcción del prototipo comenzó una vez se estabilizó el diseño, y se trabajó de manera conjunta con el Director del proyecto, durante reuniones periódicas en donde se fue perfeccionando el diseño del sistema, haciendo las debidas modificaciones en el diseño y se aterrizó la idea del sistema.

Para la elaboración del prototipo no funcional se recreó la sensación de estar navegando en el producto final debido a los efectos en la interfaz. Tomó tiempo lograr la satisfacción del cliente final, y la optimización del diseño del sistema y garantizar la eficiencia del desarrollo y la programación de la nueva versión del sistema de información SICU.

Para esta nueva versión de SICU, el prototipo inicial resultó tan claro que el trabajo fue efectivo y evitó cambios drásticos que podrían haber solicitado los usuarios al final del proyecto.

5.4.2 Prototipo final.

El prototipo final, es el prototipo resultado de las correcciones del prototipo inicial junto con una serie de cambios que aunque no fueron drásticos, son notorios en la interfaz e importantes en la programación para la generación de los cuadros.

Las revisiones previas realizadas junto con los funcionarios de Planeación y la Directora de proyecto, condujo a un resultado de prototipo funcional que cumple con todos los objetivos estipulados en el plan de proyecto.

Para dar una idea más clara del trabajo realizado, y entregado como prototipo final, se escogió un ejemplo del prototipo final para detallarlo en el presente informe.

Figura 30. Captura de la interfaz final del Menú general 'SICU'.

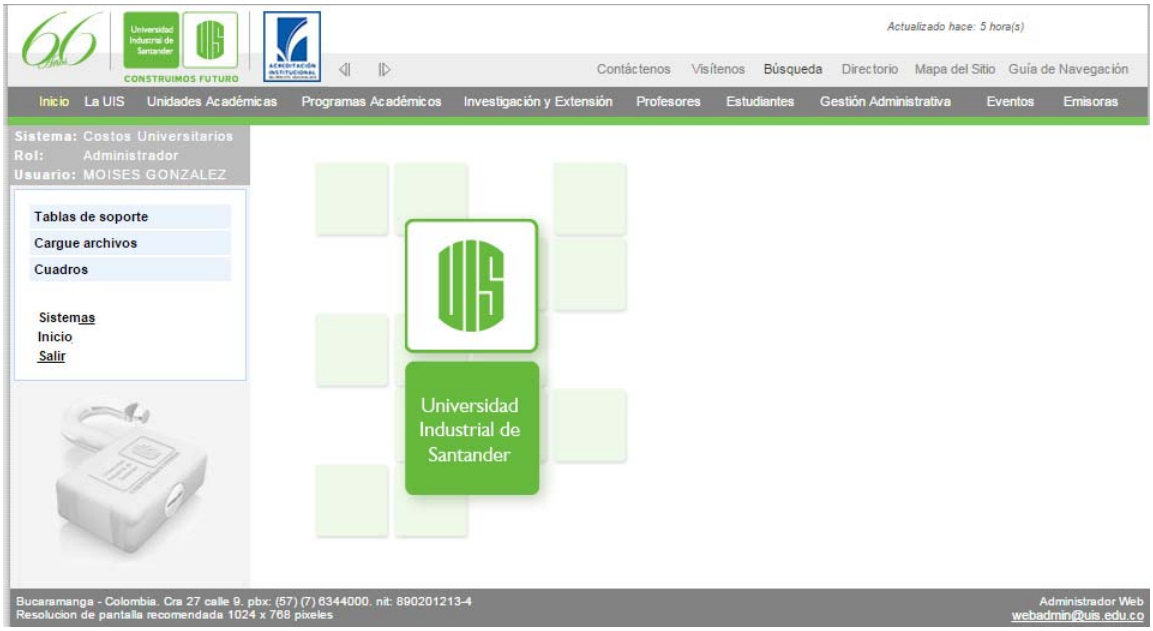
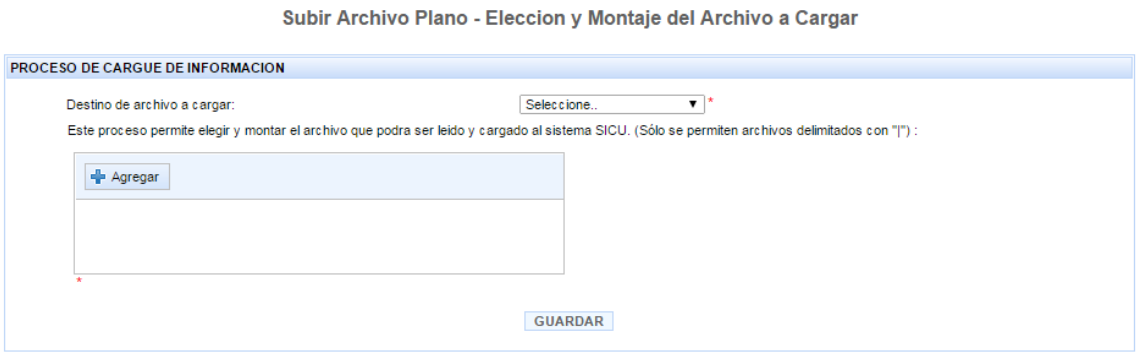


Figura 31. Captura de la interfaz final ‘Cargue de archivos’.



6 CONCLUSIONES

- La nueva versión del sistema de información de costos universitarios SICU llena las expectativas de Planeación en cuanto a oportunidad, calidad de la información y facilidad de uso.
- El reporte de información útil e inmediata es fundamental dentro de los objetivos organizacionales de la Universidad Industrial de Santander.
- Los sistemas de información de cualquier tipo, deben volverse fáciles de usar, flexibles y amigables para el usuario de tal forma que la dependencia con el desarrollador sea reducida al máximo.
- La utilización de herramientas libres como Java, reduce costos y permite la realización de aplicaciones seguras, robustas y de tipo empresarial, debido al permanente aporte que hacen usuarios de todo el mundo y la facilidad de capacitación online en foros de discusión.
- El proceso de construcción de prototipos no funcionales, permite un ahorro de tiempo debido a que el usuario interactúa con las entradas y salidas que va a tener el sistema y su esquema de navegación y hace las correcciones y sugerencias, sin desgastes de tiempo de programación.
- El software existente para el sistema de información SICU, sirvió como referencia para conocer de forma global como era la interacción entre el usuario y el software además de los resultados esperados.
  El diseño de la nueva versión permitió obtener una herramienta que satisface las necesidades del cálculo de costos de la UIS.

7 RECOMENDACIONES

- Implantar el prototipo funcional del sistema de información de costos universitarios ‘SICU’ desarrollado en este proyecto, en el menú de las nuevas versiones de software de la UIS.
- Es necesario que la persona encargada de la administración general del sistema tenga conocimiento completo y claro de cada una de las funcionalidades, de esta forma se garantizara un buen manejo acorde con los resultados esperados.
- Se recomienda actualizar el manual operativo del sistema de información de costos universitarios, una vez se haya implantado el prototipo funcional.
- Solicitar a las diferentes Unidades académicas implicadas en el proceso de costos, la información y método de obtención de los datos que son cargados a través de archivos planos al sistema de costos universitarios, para que a futuro se haga la implementación de Web Services y estos datos sean capturados directamente de las bases de datos.
- Adecuar el proceso de cálculo de costos de ‘Outsourcing’ a la nueva modalidad de contratación llamada ‘Planta temporal’, con el fin de mantener actualizado el sistema de costos universitarios ‘SICU’.

BIBLIOGRAFÍA

ARROYAVE, Trejos; M. H., & CARDONA Zamora, D. F. Criterios de evaluación de plataformas de desarrollo de aplicaciones empresariales para ambientes web. Pereira: UNIVERSIDAD TECNOLÓGICA DE PEREIRA, 2012.

BECK, K. Extreme Programming Explained: Embrace Change, 2nd Edition. Addison-Wesley, 2004.

DIMAGGIO, L., CONNER, K., KUMAR B, M., & CUNNIGHAM, K. . JBoss ESB. Packt Publishing Ltd, 2012.

DIVISIÓN DE SERVICIOS DE INFORMACIÓN, Metodología, modelado y descripción de procesos para el sistema de costos universitarios, Universidad Industrial de Santander, 1997.

ISAKOWITZ, T., BIEBER, M., & VITALI, F. Web Information Systems. Communications of the ACM, July 1998 , Volume 41 Issue 7, 78-80.

KODALI, R. R., WETHERBEE, J., & Zadrozny, P. Beginning EJB 3 Application Development: From Novice to Professional. Apress, 2006.

LAUDON, K., & LAUDON, J. Sistemas de información gerencial. Mexico: PEARSON EDUCACIÓN, 2012.

O'CONNOR, Rory; BADDOO, Nathan; CUADRADO, Juan; REJAS, Ricardo; SMOLAMDER, Kari; MESSNARZ, Richard. Software Process Improvement. 2008.

RUMBAUGH, J., JACOBSON, I., & BOOCH, G. El Lenguaje Unificado de Modelado. 2000: Pearson Education S.A. 2000.

SALAZAR BALLESTEROS, Adriana. Plan de proyecto. Manual operativo sistema de costos universitarios. Universidad Industrial de Santander. 2007.

SUMMERVILLE, I. Ingeniería del Software Séptima Edición. En Ingeniería del Software. Madrid: Pearson Educación S.A., 2005. (págs. 226-227).

VAL, Javier. Instituto de Investigaciones Tecnológicas. Web 2.0 [en línea] Wikidot.com [Santiago de Compostela, ES.] Master Comercio Exterior, Abril 2008. [citado 18 Enero 2015] Disponible en Internet: http://moduloweb.wikidot.com/web-2-0

ZAPATA, C. M., ESTRADA, B. M., & GONZALEZ, G. Reglas de conversión entre el diagrama de clases y los grafos conceptuales de Sowa. Revista Ingenierías Universidad de Medellín, 2006.