IMPLEMENTACIÓN DE UNA APLICACIÓN DISTRIBUIDA SOPORTADA EN WEB SERVICES BASADA EN LA METODOLOGÍA DE DESARROLLO DE NIPPON COMPUTER KAIHATSU LTD. (NCK)

RICARDO MARTÍNEZ CAMACHO
JUAN FERNANDO NOVA MARTÍNEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2012

IMPLEMENTACIÓN DE UNA APLICACIÓN DISTRIBUIDA SOPORTADA EN WEB SERVICES BASADA EN LA METODOLOGÍA DE DESARROLLO DE NIPPON COMPUTER KAIHATSU LTD. (NCK)

RICARDO MARTÍNEZ CAMACHO
JUAN FERNANDO NOVA MARTÍNEZ

Trabajo de grado para optar al título de Ingeniero de Sistemas

Director
FERNANDO ANTONIO ROJAS MORALES
Profesor Titular
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2012

A mis padres,

por su apoyo incondicional.

Ricardo Martínez Camacho

A Dios por ser el creador de todo lo existente, a mi padre que desde el cielo siempre sentí sus pasos a mi lado, a mi madre por enseñarme a ser una mejor persona llena de valores y principios bíblicos, a mis hermanos por ayudarme a cumplir mis objetivos como persona y estar siempre presentes en todas las etapas de mi vida.

Juan Fernando Nova Martínez

AGRADECIMIENTOS

A nuestro director de proyecto y profesor, Msc. Fernando Antonio Rojas Morales, por la confianza depositada en nosotros, sus invaluables enseñanzas y su compromiso y colaboración durante la realización del proyecto.

A todos nuestros profesores que compartieron con nosotros sus conocimientos durante el transcurso de la carrera.

A toda la comunidad de la Universidad Industrial de Santander que nos permitió crecer a nivel personal y profesional.

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TÍTULO: IMPLEMENTACIÓN DE UNA APLICACIÓN DISTRIBUIDA SOPORTADA EN WEB SERVICES BASADA EN LA METODOLOGÍA DE DESARROLLO DE NIPPON COMPUTER KAIHATSU LTD. (NCK)[*]

AUTORES: MARTÍNEZ CAMACHO, Ricardo. NOVA MARTÍNEZ, Juan Fernando.[**]

PALABRAS CLAVE: Aplicación distribuida, Web Services, NCK.

DESCRIPCIÓN

Este trabajo de investigación plantea el diseño de una aplicación distribuida que soporta procesos de negocios en un escenario de ejemplo representado en una organización de venta y alquiler de películas con necesidades de administración remota, operaciones locales y divulgación de información pública en Internet.

El escenario fue seleccionado y definido de acuerdo a necesidades reales identificadas en el entorno empresarial regional buscando que este proyecto permitiera no solamente la difusión de conocimientos actualizados de nuevas técnicas y tecnologías de ingeniería de software, sino también proveer las bases para la efectiva aplicación de este conocimiento permitiendo realizar aportes al desarrollo de emprendimientos similares a nivel regional o nacional.

Este documento comprende, inicialmente, una presentación de la investigación en donde se tratan temas como los objetivos que se quieren alcanzar, el por qué se plantea su realización, la metodología NCK, usada principalmente en proyectos de las tecnologías de información y una reseña de las tecnologías que dan soporte a la solución propuesta. Posteriormente, se encuentra el desarrollo del ejercicio planteado dividido en las etapas de investigación, diseño, trabajo y verificación. Finalmente, se presentan las conclusiones y los anexos usados en la realización del proyecto.

El desarrollo de este trabajo está enmarcado en la metodología aprendida a lo largo de un proceso de inmersión que se extendió por seis meses en el año 2011 en la compañía Nippon Computer Kaihatsu Ltd. localizada en Tokio, Japón.

[*] Trabajo de grado.

[**] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática. Director: MSc. Fernando Antonio Rojas Morales

ABSTRACT

TITLE: IMPLEMENTATION OF A DISTRIBUTED APPLICATION SUPORTED ON WEB SERVICES BASED ON NIPPON COMPUTER KAIHATSU (NCK) LTD'S SOFTWARE DEVELOPMENT METHODOLOGY[*]

AUTHORS: MARTÍNEZ CAMACHO, Ricardo. NOVA MARTÍNEZ, Juan Fernando.[**]

KEYWORDS: Distributed application, Web Services, NCK.

DESCRIPTION

This research work proposes the design of a distributed application that supports business processes on an example scenario represented by a video store company that has requirements for remote administration, local operations and publishing of public information on the Internet.

The scenario was selected and defined according to real needs identified on the regional business environment with the objective of not only to allow the spread of updated knowledge of new software engineering techniques and technologies, but also to provide the basis for the effective application of this knowledge allowing to make real contributions to the development of related entrepreneurship in the regional or national environment.

This document comprises, initially, the presentation of the research where the included topics are the objectives, the justification for its development, the NCK methodology, applied mainly on IT related projects and a description of the technologies supporting the proposed solution. Afterwards, the development of the proposed exercise is presented being divided in the stages of research, design, construction and verification. Lastly, the conclusions and annexes are presented.

This research is guided by a methodology experienced during an immersion process that lasted six months in the year 2011 at the company Nippon Computer Kaihatsu Ltd. located in Tokyo, Japan.

[*] Graduate thesis project.

[**] Physical-Mechanical Engineering Faculty, School of IT Engineering and Informatics. Director: MSc. Fernando Antonio Rojas Morales

INTRODUCCIÓN

Las aplicaciones web se han convertido en pocos años en complejos sistemas alcanzando capacidades que antes estaban reservadas para las aplicaciones de escritorio, soportando procesos de negocio de diversos tipos y requerimientos exigentes de funcionalidad y disponibilidad. El crecimiento de este tipo de aplicaciones se ha debido en gran parte a la creciente necesidad de contar con información confiable y de disponibilidad inmediata.

Reconociendo estas necesidades, las empresas han adoptado progresivamente soluciones de tecnologías de la información aplicadas en Internet, en un principio para hacer llegar a sus clientes su oferta de productos y servicios, y ahora para ofrecerles diferentes tipos de servicios que funcionan a través de este medio, convirtiéndolos en una ventaja competitiva.

Así mismo, la continua expansión de las redes y la Internet ha hecho posible el crecimiento del uso de sistemas distribuidos impulsados en los más variados entornos corporativos alrededor del mundo. De esta forma, la interoperabilidad de estos sistemas ha ganado importancia dentro de la investigación actual, promoviendo el origen y el uso de estándares de comunicación de la información.

En el mundo globalizado y competitivo actual, es de vital importancia que el entorno académico y empresarial de la región se mantenga actualizado respecto a nuevas técnicas y tecnologías que les brinden ventajas competitivas. En este aspecto, el desarrollo de este proyecto pretende proporcionar las bases para realizar aportes reales y significativos al desarrollo del emprendimiento empresarial en el ámbito regional y nacional, impulsado por los conocimientos generados al interior de la academia, de la forma en que se detalla en la presentación de este trabajo de investigación.

1. PRESENTACIÓN DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA

En las últimas décadas, los sistemas de software han estado basados principalmente en el desarrollo de aplicaciones de escritorio y en modelos Cliente-Servidor. En ambos casos, el software principal se ejecuta en una sola computadora, ya sea la computadora cliente o el servidor. Con la introducción de agentes inteligentes, Web APIs y la Web 2.0 y el surgimiento de Cloud computing, los enfoques de “múltiples máquinas” están siendo utilizados cada vez más al ganar mayor relevancia, donde varios sistemas en diferentes ubicaciones pueden ocuparse de la redistribución de diferentes tareas, o donde cada una de las máquinas cumple una tarea o propósito específico.

Con el incremento de la complejidad de diversos requerimientos de los usuarios de aplicación, en la actualidad, las aplicaciones distribuidas están siendo demandadas ampliamente por diferentes industrias como la industria bancaria, aérea, de telecomunicaciones, entre otras. Para satisfacer las demandas de cada sector de negocios los ingenieros de software están diseñando aplicaciones distribuidas, de acuerdo a los requerimientos de los proyectos de cada industria. Con el crecimiento regular en esos sectores, las compañías que están usando aplicaciones distribuidas siempre buscan la agregación de nuevas características, donde los desarrolladores puedan modificar e integrar fácilmente. Desde el punto de vista del desarrollador, realizar modificaciones en la aplicación es una tarea que se facilita cuando la aplicación está basada en una arquitectura distribuida.

Actualmente en la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander se evidencia una mínima cantidad de trabajos

de grado relacionados con aplicaciones distribuidas, motivo por el cual se hace pertinente la realización de este proyecto, motivando la adquisición de conocimientos vigentes de nuevas técnicas de ingeniería de software al interior de la escuela. Así mismo, es de vital importancia difundir conocimientos producidos en contextos diferentes a nuestra región, con el fin de permitir innovación en los trabajos de grado y lograr que causen un impacto positivo, real y tangible en el desarrollo de nuestro entorno regional o nacional, en lugar de quedar archivados habiendo cumplido únicamente una función de requisito de grado.

1.2 OBJETIVOS

1.2.1 Objetivo general

Diseñar, codificar y probar una aplicación distribuida funcional asíncrona soportada en Web Services usando la metodología de desarrollo implementada en Nippon Computer Kaihatsu Ltd. (NCK)

1.2.2 Objetivos específicos

- Desarrollar una aplicación software capaz de realizar las siguientes funciones:
    - Distribuir tareas en diferentes máquinas de una red.
    - Operar en plataforma Web y a nivel de máquina local.
    - Realizar comunicación entre componentes a través de Web Services.

- Realizar pruebas usando las herramientas y métodos que soportan la metodología de NCK.

1.3 JUSTIFICACIÓN

El resultado de este proyecto busca aportar una aplicación distribuida funcional que genere una mejora sustancial del manejo de la información en un plan de emprendimiento de la región, brindando ventajas competitivas como un alto nivel de tecnificación en sus procesos, difusión de información a lo largo de la organización y fácil acceso a la misma.

De igual forma, se aplica la metodología de desarrollo de NCK, que es ejemplo de éxito en el entorno de negocios japonés. Se pretende promover dicha alternativa a la hora de administrar proyectos de base tecnológica, por medio de la creación de una aplicación funcional que pruebe la viabilidad del uso de la misma en soluciones para nuestra región.

Debido a que la metodología de desarrollo de NCK es estricta en cuanto a la calidad de los productos resultantes en cada uno de los procesos, es necesario crear un grupo de trabajo que siga los preceptos básicos de la misma. La complejidad del proyecto y las restricciones de tiempo de desarrollo hacen necesario distribuir las actividades en un equipo de trabajo de mínimo dos personas.

Lo anteriormente expuesto se fundamenta en la normatividad vigente sobre los trabajos de grado registrada en el Reglamento Académico de la Universidad Industrial de Santander, que define la modalidad de Trabajo de Investigación como "el diseño y ejecución de un plan que busca aportar soluciones a problemas teóricos o prácticos, vigentes en el entorno local, regional o nacional; adecuar y apropiar tecnologías; replicar y validar conocimientos producidos en otros contextos; generar innovación o realizar el estudio y análisis teórico de un problema mediante un trabajo monográfico".

Así mismo, se procura despertar interés en áreas de investigación en la Universidad Industrial de Santander relacionadas con aplicaciones distribuidas e interoperabilidad de sistemas. Además, el desarrollo de esta iniciativa permite la aplicación de procesos de ingeniería, abarcando el ciclo completo de desarrollo de software, razón por la cual se hace pertinente su realización como proyecto de grado de Ingeniería de Sistemas e Informática.

2. MARCO TEÓRICO

A continuación se proporcionará un resumen general de los conceptos teóricos que enmarcarán el desarrollo de este proyecto.

2.1 APLICACIONES DISTRIBUIDAS

Una aplicación distribuida es una aplicación con diferentes componentes que se ejecutan en entornos separados, comúnmente en diferentes plataformas conectadas a través de una aplicación que sigue el modelo cliente/servidor.

En este tipo de aplicaciones suelen identificarse los siguientes elementos:

Lado del servidor: Tiene un programa que se ejecuta y está conectado a una red. Está a la escucha en un puerto, esperando las peticiones de los clientes; por ejemplo, un servidor Web escucha en el puerto 80. Un computador que ejecuta un servidor de aplicación necesita estar conectado a la red para responder a las peticiones de los clientes.

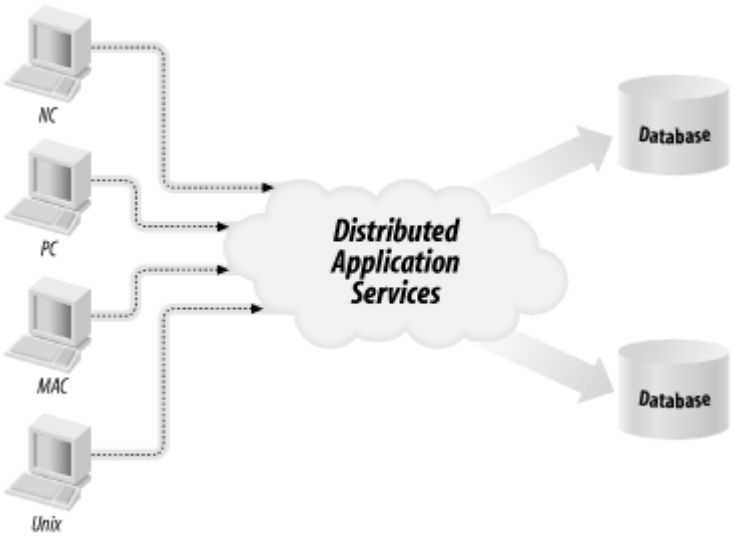
Lado cliente: Programa que ejecuta el usuario de la aplicación. El cliente hace sus peticiones al servidor a través de la red. Por ejemplo, un navegador Web.

Protocolo de aplicación para la comunicación entre el cliente y el servidor: El protocolo define el tipo de mensajes intercambiados; por ejemplo, el protocolo de la capa de aplicación de la Web, HTTP, define el formato y la secuencia de los mensajes transmitidos entre el navegador y el servidor Web.

Formato de los mensajes que se intercambian: Algunas veces forma parte del servicio; por ejemplo, en el correo electrónico se define el formato de los mensajes electrónicos.

Estos componentes son independientes de la arquitectura de red que se utiliza. Las típicas aplicaciones distribuidas son de dos niveles (cliente-servidor), tres niveles (cliente-middleware-servidor) y multiniveles.

Figura 1. Ejemplo de arquitectura de una aplicación distribuida



Como se puede ver, esta arquitectura es básicamente la arquitectura cliente/servidor con un lugar especial para la lógica de la aplicación (la capa intermedia). Sin embargo, esta pequeña diferencia representa un gran cambio filosófico del diseño cliente/servidor. En resumen, es importante separar la lógica de la aplicación de otros tipos de lógica.

De hecho, ubicar la lógica de la aplicación en la base de datos o en la interfaz de usuario limita la capacidad de la aplicación para crecer con demandas cambiantes. Por ejemplo, si se necesita un cambio simple en el modelo de datos, se tendrán que hacer cambios significativos en los procedimientos almacenados si la lógica de la aplicación está en la base de datos. Un cambio a la lógica de aplicación de la interfaz de usuario, por el contrario, obliga a que sea necesario modificar también el código de interfaz de usuario con el consiguiente riesgo de agregar errores en los sistemas que no tienen nada que ver con los cambios que se están introduciendo.

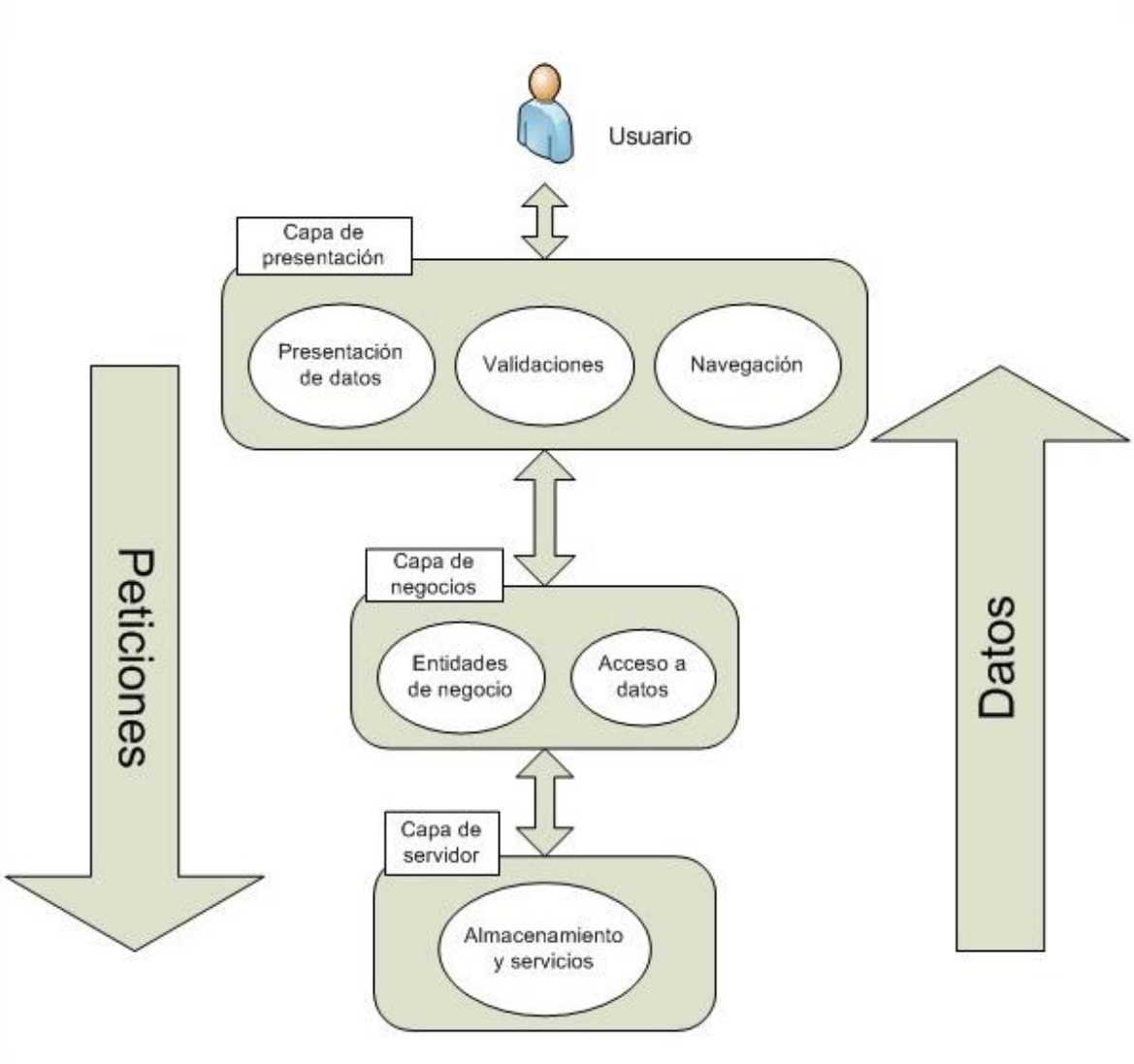
La arquitectura de aplicaciones distribuidas permite realizar dos funciones esencialmente. La primera es proporcionar un hogar para el procesamiento de la aplicación para que no se mezcle con la base de datos o el código de la interfaz de usuario. Sin embargo, también hace que la interfaz de usuario sea independiente del modelo de datos subyacente. Bajo esta arquitectura, los cambios en el modelo de datos afectan sólo a la forma en que la capa intermedia obtiene y recibe la información en la base de datos. El cliente no tiene conocimiento de esta lógica y por lo tanto no se preocupa por tales cambios.

La arquitectura de aplicaciones distribuidas introduce dos elementos críticos. En primer lugar, la capa de la lógica de aplicación permite la reutilización de la lógica de aplicación por parte de varios clientes. En concreto, mediante la abstracción de la lógica de la aplicación de puntos de integración bien definidos, es posible volver a usar esa lógica con interfaces de usuario no previstas cuando la lógica de aplicación fue escrita.

En segundo lugar, esta arquitectura ofrece a las aplicaciones la capacidad de proporcionar soporte de tolerancia a fallos y escalabilidad. Los componentes de esta arquitectura son componentes lógicos, lo que significa que se pueden distribuir a través de múltiples instancias reales. Cuando una base de datos o un

servidor de aplicación implementa clustering, puede actuar y comportarse como una sola capa al mismo tiempo que distribuye procesamiento en múltiples máquinas físicas. Si una de las máquinas falla, la capa intermedia aún seguirá en funcionamiento.

Figura 2. Esquema lógico de las capas en una aplicación distribuida



2.2 SERVICIOS WEB

Los servicios Web extienden la infraestructura de la World Wide Web para proporcionar los medios que le permitan a las aplicaciones software conectarse unas con otras. Las aplicaciones acceden a los servicios Web a través de protocolos y formatos de datos de uso extendido como HTTP, XML y SOAP, sin necesidad de preocuparse por la forma en que se implementa cada servicio Web.

2.2.1 Definición de servicio Web

Según la W3C, un servicio Web es un sistema de software diseñado para permitir la interoperabilidad de máquina a máquina en una red. Cuenta con una interfaz que se describe en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el servicio Web de una manera definida por su descripción utilizando mensajes SOAP, típicamente transmitido a través de HTTP con una serialización XML en conjunto con otros estándares web relacionados.

En general, los servicios web son sólo APIs Web que pueden ser accedidas en una red, como Internet, y ejecutadas en un sistema de alojamiento web remoto.

2.2.2 Agentes y servicios

Un servicio Web es un concepto abstracto que debe ser aplicado por un agente concreto. El agente es la pieza concreta de hardware o software que envía y recibe mensajes, mientras que el servicio es el recurso que se caracteriza por el conjunto abstracto de funcionalidad que se proporciona.

Para ilustrar esta distinción, se podría implementar un servicio Web en particular usando un agente un día (tal vez escrito en un lenguaje de programación) y un agente diferente al otro día (tal vez escrito en un lenguaje de programación

diferente) con la misma funcionalidad. Aunque el agente puede haber cambiado, el servicio web sigue siendo el mismo.

2.2.3 Los solicitantes y los proveedores

El propósito de un servicio Web es proporcionar algunas funciones en nombre de su dueño (una persona u organización, como por ejemplo un negocio o un individuo). La entidad proveedora es la persona u organización que ofrece un agente apropiado para implementar un servicio particular. La entidad solicitante es una persona u organización que desee hacer uso del servicio Web de una entidad proveedora. Utilizará un agente solicitante para intercambiar mensajes con el agente proveedor de la entidad proveedora. En la mayoría de los casos, el solicitante es quien inicia este intercambio de mensajes, aunque no siempre. Sin embargo, por consistencia todavía se usa el término "agente solicitante" para el agente que interactúa con el agente proveedor, incluso en los casos en que el agente proveedor en quien inicia el intercambio.

2.2.4 Descripción del servicio

La mecánica del intercambio de mensajes se documenta en una descripción de servicio Web (WSD). La WSD es una especificación procesable por máquina de la interfaz del servicio Web, escrita en WSDL. En ella se definen los formatos de los mensajes, tipos de datos, protocolos de transporte y formatos de serialización de transporte que se debe utilizar entre el agente solicitante y el agente proveedor.

También especifica una o más ubicaciones de red en las cuales puede ser invocado un agente proveedor, y puede proporcionar alguna información sobre el patrón de intercambio de mensajes que se espera.

En esencia, la descripción del servicio representa un acuerdo que rige la mecánica de la interacción con ese servicio.

2.2.5 Semántica

La semántica de un servicio Web es la expectativa compartida sobre el comportamiento del servicio, en particular en respuesta a los mensajes que se envían a él. En efecto, este es el "contrato" entre la entidad solicitante y la entidad proveedora con respecto al propósito y las consecuencias de la interacción. Aunque este contrato representa el acuerdo general entre la entidad solicitante y la entidad proveedora de cómo y por qué sus respectivos agentes interactuarán, no necesariamente es escrito o negociado explícitamente. Puede ser implícito, oral o escrito, procesable por máquina u orientado al humano, y puede ser un acuerdo legal o informal.

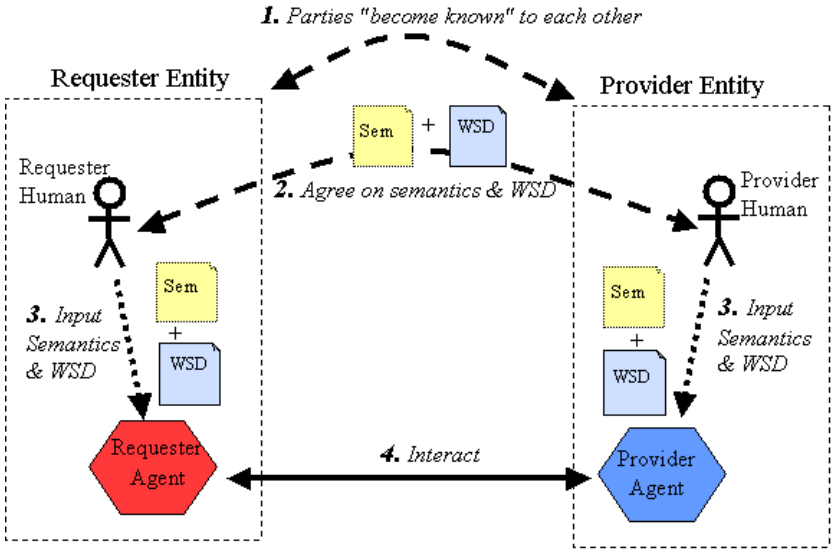
Mientras que la descripción del servicio representa un contrato que regula los mecanismos de interacción con un servicio en particular, la semántica representa un contrato que regula el significado y el propósito de esa interacción. La línea divisoria entre estos dos no es necesariamente rígida. Mientras más lenguajes ricos en semántica se utilizen para describir la mecánica de la interacción, mayor información esencial puede migrar de la semántica informal a la descripción del servicio. A medida que esta migración se produce, más de la labor requerida para lograr una interacción exitosa puede ser automatizada.

2.2.6 Descripción general de contratación de un servicio Web

Hay muchas maneras en que la entidad solicitante puede utilizar un servicio Web. En general, los siguientes pasos generales son necesarios: (1) las entidades

solicitante y proveedoras se conocen el uno al otro (o por lo menos uno conoce al otro), (2) la entidad solicitante y la entidad proveedora de alguna manera se ponen de acuerdo en la descripción del servicio y la semántica que regirá la interacción entre el agente proveedor y el agente solicitante; (3) la descripción del servicio y la semántica son reconocidos por el agente proveedor y solicitante; y (4) el agente solicitante y proveedor intercambian mensajes, realizando así alguna tarea en nombre del solicitante y las entidades prestadoras. (Es decir, el intercambio de mensajes con el agente proveedor representa la manifestación concreta de interactuar con la entidad proveedora del servicio Web). Algunos de estos pasos pueden ser automatizados, y otros pueden llevarse a cabo manualmente.

Figura 3. Proceso general de contratación de un servicio Web



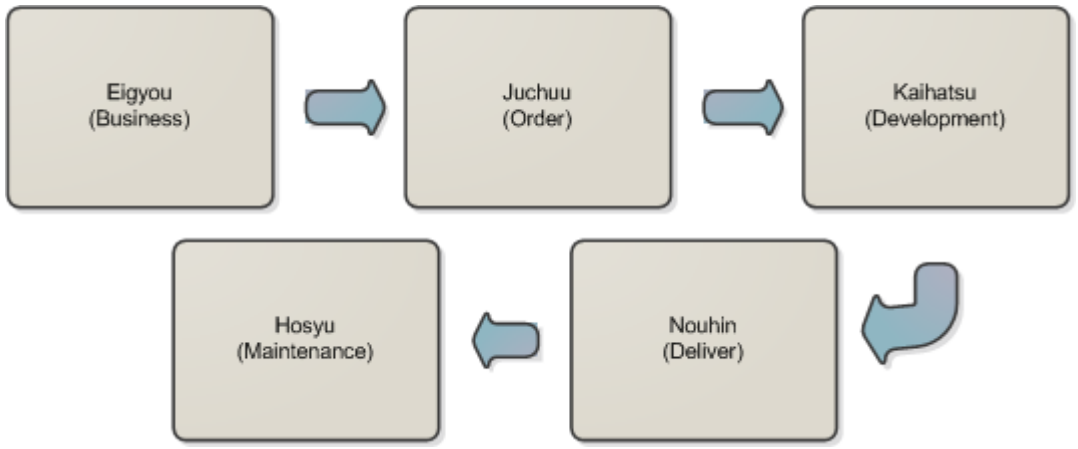
3. METODOLOGÍA NCK

3.1 DESCRIPCIÓN DE LA METODOLOGÍA DE DESARROLLO DE NIPPON COMPUTER KAIHATSU LTD. (NCK)

3.1.1 Un modelo general de negocio

Para entender la metodología a usar es importante partir de un flujo que busque simular una relación de negocio habitual, es por esto que se plantea el siguiente flujo de trabajo que supone el funcionamiento de un negocio.

Figura 4. Un flujo general de trabajo



EIGYOU (EL NEGOCIO)

En esta etapa es que se da la necesidad del cliente, este descubre los problemas que está teniendo su organización y decide hacer algo al respecto. Se dice que en

esta etapa nace el negocio ya que del decidir actuar se desprende toda la dinámica que implica la creación de proyecto IT.

El nacimiento de las necesidades del cliente se puede dar bien sea por intervención externa (consultores, clientes) o por entorpecimiento de los procesos internos, es decir, estas necesidades pueden aparecer en el día a día o en por iniciativas de mejoramiento empresarial en las que personal externo evalúa a la compañía y propone salidas alternativas a los inconvenientes encontrados.

JUCHUU (LA ORDEN)

Una vez descubiertas las problemáticas de la organización y bien definidos los alcances de lo que se desea hacer se hace la orden de los requerimientos deseados. Este proceso se debería llevar a cabo por los miembros de la compañía ya que son estos los que conocen a profundidad la organización o bien por los consultores externos que realizan recomendaciones a la entidad.

KAIHATSU (EL DESARROLLO)

En muchas ocasiones hasta este punto el grupo de desarrollo es ajeno al proceso y en muchas otras también lo es la compañía de la cual hace parte el mismo. En esta sección del flujo se desarrolla todo lo referente al desarrollo de programa, la instalación del hardware, la construcción de las redes, la instalación y configuración de los servidores y la personalización de los paquetes.

NOUHIN (LA ENTREGA)

Como su nombre lo indica en esta fase se entrega bien sea el programa, el hardware, el sistema, la infraestructura, los paquetes o la combinación de cualquiera de estos.

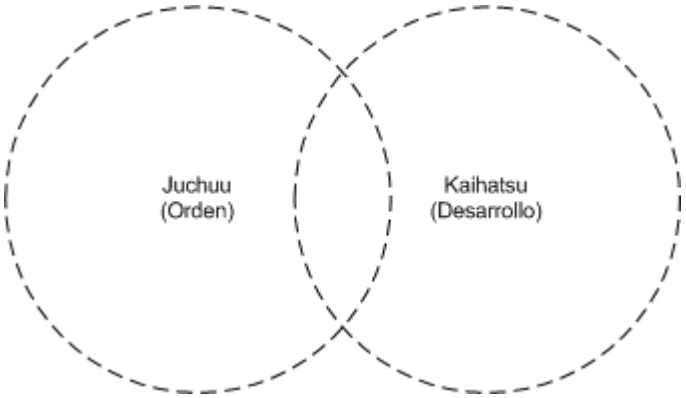
HOSYU (EL MANTENIMIENTO)

Esta fase es la que parecería menos importante en el proceso, sin embargo, en ella se llevan procesos como el seguimiento que pueden llevar a desprender nuevos ciclos de flujos similares al descrito y el mantenimiento que podría para muchas soluciones extenderse de forma vitalicia dependiendo de la importancia de la solución.

3.1.2 Un flujo general de desarrollo

Las relaciones más significativas para el desarrollo del producto a entregar se dan en los procesos del modelo de negocio Orden y Desarrollo, por lo que es de esperarse que la metodología de desarrollo de software a continuación expuesta se centre en estas dos etapas del negocio y por lo que una mirada más cercana a lo que sucede en estos ciclos se hace necesaria.

Figura 5. Etapas Juchuu y Kaihatsu

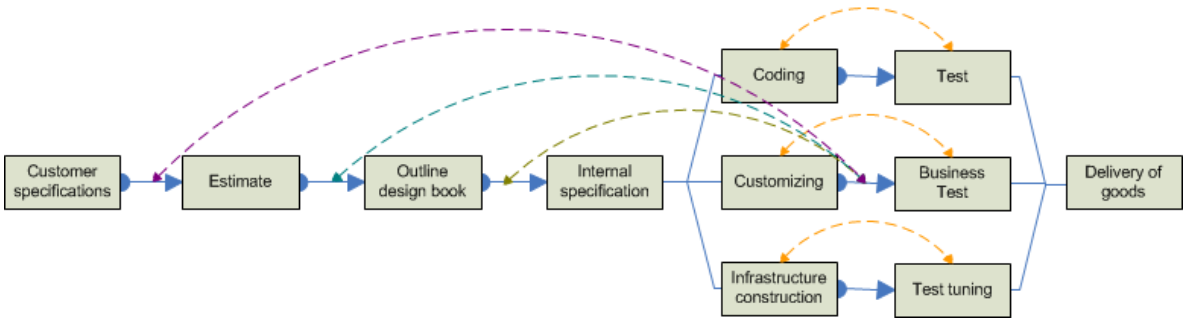


Existen momentos en los que se hace complicado el diferenciar el estado en el cual se encuentra un flujo de trabajo y muchas veces los límites de los mismos se hacen confusos. Estos casos suelen presentarse con regularidad y dependen de que tanto este involucrado el cliente en los procesos de desarrollo y hasta donde llegue su cooperación.

3.1.3 La metodología de desarrollo

El siguiente grafico intenta explicar cómo funcionaría un flujo de desarrollo o una aplicación de la metodología de NCK a la fabricación de un entregable cualquiera.

Figura 6. Flujo general de desarrollo



Lo expuesto en el grafico anterior se profundiza en las cuatro etapas en las que se centra la metodología a usar: Investigación, Diseño, Trabajo y Verificación y que se describen detalladamente a continuación.

INVESTIGACIÓN

Especificaciones del cliente. Las especificaciones presentadas por el cliente son confirmadas y reunidas en el libro de definiciones de especificaciones. La información solo puede ser obtenida desde el cliente ya que esta proviene del negocio del cliente y es él quien conoce su negocio mejor. Lo aportado por el cliente es evaluado desde el lado de la compañía y se convierte en un flujo hacia el cliente con el fin de verificar. Es responsabilidad de la compañía el dar resultado tal cual y como se especificaron en esta fase.

DISEÑO

Libro de bosquejo de diseño. Es un documento de funciones que reúne todas las demandas del cliente. Cuando el objeto de desarrollo es un programa, las funciones de IN/OUT deben estar claramente definidas, junto con la composición de red, la configuración del sistema y la infraestructura en general.

Diseño interno. Debe ser tan detallado como sea posible, se encuentra en ella la infraestructura de construcción basándose en el diseño exterior de funciones. Cuando el programa se empieza a desarrollar se convierte en el libro detallado de especificaciones, en las especificaciones del servidor, de red, de bases de datos, etc.

TRABAJO

Codificación. La codificación se hace a partir del libro detallado de especificaciones y la herramienta difiere según el proyecto.

Personalización. La personalización de las herramientas incluye todo lo que se menciona a continuación y en la etapa de codificación. El paquete implica un programa en el mercado.

Construcción de infraestructura. La infraestructura es decidida en base a las necesidades del cliente y con esto es construida. El Web Server y DB Server son indicados dentro de la infraestructura.

Construcción de hardware. La construcción del hardware necesario diferente a los servidores es ejecutada. Las especificaciones del hardware son necesarias.

VERIFICACIÓN

Pruebas de escritorio. El programa principal no es movido de la maquina en la cual es desarrollada. Se dice que el 80% de los bugs se pierden con este trabajo. Checklist (CL) son hechas y probadas. El contenido de las listas poseen ítems especializados en la confirmación de los mismos.

Pruebas de unidad. Se cambia de máquina para la aplicación de la prueba, porque la prueba se ejecuta con el programa unidad se convierte en una prueba de caja blanca. Checklist (PCL) la prueba de unidad son diseñadas y aplicadas.

Pruebas de combinación. Se cambia de máquina para la aplicación de la prueba, porque la prueba se ejecuta con el programa sincronizador se convierte en una prueba de caja negra. Checklist (CCL) para las pruebas de combinación son diseñadas y aplicadas.

Pruebas de negocio. Se ejecuta en la máquina en la que va a correr la solución, el cliente se une a la prueba y se convierte a lo más cercano a una muestra real. Checklist (SCL) para la pruebas de negocio son diseñadas y aplicadas.

Operación de verificación. La operación de verificación incluye el hardware construido para la solución. La operación de verificación incluye todos los servidores e infraestructuras de red.

Tuning. Se corre sobre la infraestructura final del cliente. Se afina la solución a la infraestructura final.

4. ANÁLISIS Y DISEÑO DE LA APLICACIÓN

El presente capítulo describe las actividades realizadas en la primera etapa de análisis y diseño de la aplicación.

Es importante precisar que la metodología NCK proporciona un marco general en las etapas del desarrollo de software y no extiende restricciones en las actividades a desarrollar, debido principalmente a la variabilidad de los tipos de entradas y salidas entre diferentes clientes y productos a elaborar. Como se evidencia desde los objetivos mostrados en la presentación de este proyecto, se usarán herramientas y métodos diseñados específicamente para soportar la metodología NCK en la etapa de verificación y pruebas. En etapas diferentes, la teoría del diseño orientado a objetos sustentará el desarrollo de las actividades, guiado por diferentes autores que serán mencionados a través de este documento. También se hará uso de herramientas que tengan como base el desarrollo ágil de software, teniendo en cuenta el valor agregado de generación de conocimiento que pueden dar a este proyecto y a sus autores, quienes buscan realizar un aporte mediante este trabajo de investigación que pueda ser transferido de manera práctica a otros proyectos que tengan un impacto real en el entorno local de la universidad y de la región.

4.1 ESPECIFICACIÓN DE REQUERIMIENTOS DE SOFTWARE

Un requerimiento es una característica que el sistema *debe* tener o es una restricción que el sistema *debe* satisfacer para ser aceptada por el cliente. La especificación de requerimientos de software es la especificación del sistema en

términos que el cliente entienda, de forma que constituya el contrato entre el cliente y los desarrolladores.

Este trabajo de investigación plantea el diseño de una aplicación distribuida que soporta procesos de negocios en un escenario de ejemplo representado en una organización de venta y alquiler de películas con necesidades de administración remota, operaciones locales y divulgación de información pública en Internet.

El escenario fue seleccionado y definido de acuerdo a necesidades reales identificadas en el entorno empresarial regional buscando que este proyecto permitiera no solamente la difusión de conocimientos actualizados de nuevas técnicas y tecnologías de ingeniería de software, sino también proveer las bases para la efectiva aplicación de este conocimiento permitiendo realizar aportes al desarrollo de emprendimientos similares a nivel regional o nacional.

En el contexto de este proyecto, las referencias al cliente de la aplicación pueden referirse a diferentes personas o entidades voluntarias que fueron consultadas por los autores, con conocimiento de las reglas de negocio del tipo de organización a la que está dirigido un eventual producto final derivado a partir del conocimiento generado por este proyecto.

4.1.1 Requerimientos funcionales

Los requerimientos funcionales describen la interacción entre el sistema y su ambiente independientemente de su implementación. El ambiente incluye al usuario y cualquier otro sistema externo que interactúe con el sistema.[1] En esta

[1] El Lenguaje Unificado de Modelado. Grady Booch, James Rumbaugh e Ivar Jacobson. Addison Wesley, 1999.

etapa se identifican los actores del sistema y los de casos de uso que pueden describirse mediante diagramas de casos de uso detallados.

4.1.1.1 Identificación de actores del sistema

Un actor representa un conjunto coherente de roles que son jugados por una persona, un dispositivo de hardware o incluso un sistema externo que interactúa con el sistema en desarrollo. Esta actividad implica identificar roles, es decir, usuarios que realizan un conjunto de actividades definidas respecto a la funcionalidad del sistema.

Existen preguntas genéricas que pueden ayudar a identificar los actores del sistema, entre ellas es posible mencionar las siguientes:

- ¿Cuáles usuarios están soportados por el sistema para desarrollar su trabajo?
- ¿Cuáles usuarios ejecutan las funciones principales del sistema?
- ¿Cuáles usuarios desempeñan funciones secundarias, como mantenimiento y administración?
- ¿El sistema interactúa con hardware o software externo?

Como resultado de este proceso, se identificaron los siguientes actores en el sistema:

- **Administrador:** Es el usuario identificado que accede a la aplicación web. Es el único tipo de usuario que puede ingresar a la aplicación web de forma autenticada.
- **Visitante:** Es el usuario que accede a la aplicación web sin identificarse.

- **Operador:** Es el usuario que accede a la aplicación cliente de forma autenticada.

Para permitir una mayor comprensión de los roles de los actores identificados, se proporciona a continuación una breve descripción de la división que el sistema tendrá. Esta división se clasificará posteriormente como un pseudo-requisito o requisito impuesto.

- **Aplicación web:** Es la parte de la aplicación que reside en un servidor web y es accesible a través del navegador de Internet.

- **Aplicación cliente:** Es la parte de la aplicación que reside y es accesible solo en la computadora local del operador.

4.1.1.2 Identificación de casos de uso

Mediante la identificación de casos de uso se busca capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar cómo se implementa este comportamiento. Los casos de uso proporcionan un medio para que los desarrolladores, los usuarios finales del sistema y los expertos del dominio lleguen a una comprensión común del sistema.

Un caso de uso contiene una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor. Se utilizan verbos por lo general en infinitivo que representan las acciones del usuario con el sistema, por lo que siempre debe existir un tipo de usuario (actor) que lo utilice. Las relaciones entre actores y casos

de uso representan el flujo de la información durante el caso de uso, que a su vez representa qué funcionalidad puede ser realizada por un actor en particular.

Teniendo en cuenta que los casos de uso no deben ser excesivamente genéricos ni demasiado específicos, la definición de casos de uso es un proceso cíclico donde se busca refinar cada vez más los casos de uso que finalmente responderán a los requerimientos funcionales.

Inicialmente se definió funcionalidad de forma narrativa en conjunto con el cliente, para facilitar la posterior identificación y definición de los casos de uso del sistema. En este proceso, se clasificó la funcionalidad a desarrollar en tres partes principales: aplicación web, aplicación cliente y servicios web, cuyos requerimientos se detallan a continuación.

- **Aplicación web: Sección pública.**

Se refiere a la funcionalidad accesible para cualquier visitante de Internet en la aplicación web. El sistema debe:

- Permitir consultar el catálogo de películas especificando la disponibilidad de cada una.
- Proporcionar un formulario de contacto en línea basado en email.
- Tener una forma de mostrar la información de contacto de la organización cliente (dirección, teléfono, email).

- **Aplicación web: Sección administrativa**

Se refiere a la funcionalidad accesible sólo para administradores del sistema previamente identificados en la aplicación web. El sistema debe:

- Mostrar informe de alquileres de películas con rango de fecha editable.
- Mostrar informe de ventas de películas con rango de fecha editable.
- Permitir administrar la información de películas y usuarios (ver, agregar, editar, borrar).

- **Aplicación cliente**

Es la parte de la aplicación que reside y es accesible solo en la computadora local del operador. El sistema debe permitir:

- Administrar alquileres y ventas de películas, realizando control de salidas y entradas en el caso de los alquileres.
- Administrar la información de clientes e inventario (ver, agregar, editar, borrar).
- Iniciar el proceso de sincronización.

- **Servicios Web**

Se refiere a los métodos que permiten la comunicación entre la aplicación cliente y la aplicación web. El sistema debe permitir:

En flujo de información de servidor a cliente:
- Actualizar el catálogo de películas
- Actualizar el listado de usuarios

En flujo de información de cliente a servidor:
- Registrar el consolidado de alquileres y ventas de películas
- Actualizar la información de clientes e inventario

El proceso de actualización de catálogos de películas debe ser iniciado manualmente por el operador de la aplicación cliente. También puede ser iniciado

automáticamente al registrar un alquiler de película, pero en este caso su correcta ejecución no está indicada como obligatoria, considerando que puede haber momentos en que la aplicación cliente no tenga conexión a Internet.

Definición de los casos de uso generales

En esta etapa del proceso, se definen los casos de uso más generales de la aplicación, buscando lograr un consenso inicial entre las partes interesadas en el proyecto que funcione como una base del punto de partida para las posteriores definiciones de requerimientos.

Los siguientes diagramas de casos de uso representan el resultado de estas definiciones:

Figura 7. Casos de uso de la aplicación web

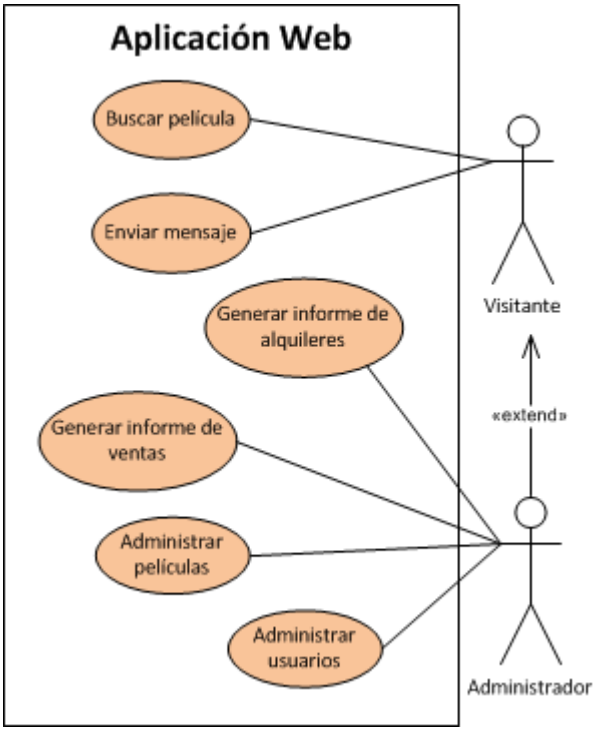
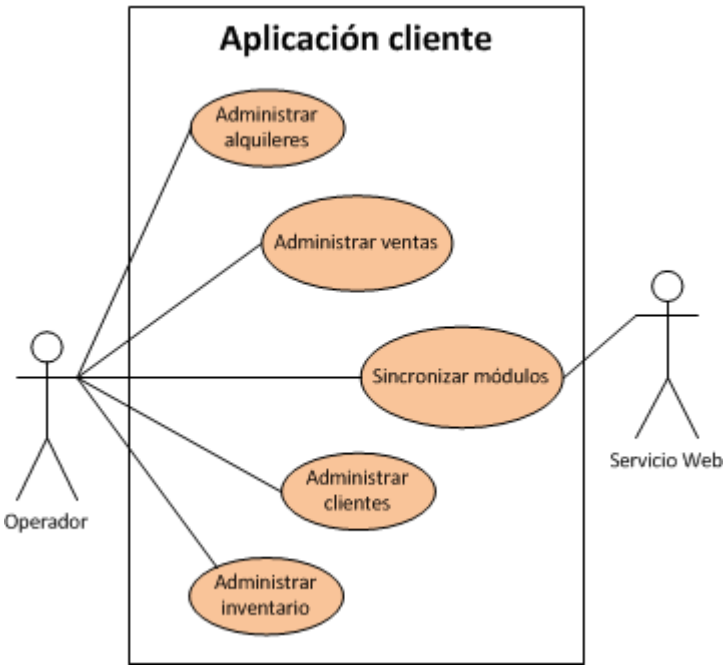


Figura 8. Casos de uso de la aplicación cliente



Índice de casos de uso. Después de haber producido una vista inicial de los actores y objetivos de los casos de uso más generales, es posible crear una tabla inicial que proporcione una base para el índice de los casos de uso. Cada caso de uso tendrá varios atributos relacionados con el proyecto y con el caso de uso mismo. A nivel de proyecto, estos atributos pueden incluir alcance, complejidad y prioridad.

Esta tabla puede ser usada por el equipo del proyecto para definir casos de uso posteriores, funcionando como un inventario maestro que ayude a escribir casos de uso efectivos en esta etapa de definición de requerimientos del proyecto.[2]

[2] Writing effective use cases and user story examples. Gatherspace.com Agile Project Management Company. http://www.gatherspace.com/static/use_case_example.html

Tabla 1. Índice de casos de uso

ID	Caso de uso	Actor primario	Actor secundario	Alcance	Prioridad
1	Buscar película	Visitante		Dentro	Media
2	Enviar mensaje	Visitante		Dentro	Baja
3	Administrar películas	Administrador		Dentro	Alta
4	Administrar usuarios	Administrador		Dentro	Alta
5	Generar informe de alquileres	Administrador		Dentro	Media
6	Generar informe de ventas	Administrador		Dentro	Media
7	Sincronizar módulos	Operador	Servicio web	Dentro	Alta
8	Administrar clientes	Operador		Dentro	Alta
9	Administrar alquileres	Operador		Dentro	Alta
10	Administrar ventas	Operador		Dentro	Alta
11	Administrar inventario	Operador		Dentro	Alta
12	Recibir mensaje	Administrador		Fuera	Baja

La columna de alcance indica si el caso de uso se incluye o no en el alcance del sistema. La columna de prioridad puede usarse para ordenar el índice.

Tabla 2. Índice priorizado de casos de uso

ID	Caso de uso	Actor primario	Actor secundario	Alcance	Prioridad
3	Administrar películas	Administrador		Dentro	Alta
4	Administrar usuarios	Administrador		Dentro	Alta
7	Sincronizar módulos	Operador	Servicio web	Dentro	Alta
8	Administrar clientes	Operador		Dentro	Alta
9	Administrar alquileres	Operador		Dentro	Alta
10	Administrar ventas	Operador		Dentro	Alta
11	Administrar inventario	Operador		Dentro	Alta
1	Buscar película	Visitante		Dentro	Media
5	Generar informe de alquileres	Administrador		Dentro	Media
6	Generar informe de ventas	Administrador		Dentro	Media
2	Enviar mensaje	Visitante		Dentro	Baja
12	Recibir mensaje	Administrador		Fuera	Baja

A continuación, se muestra el detalle del primer caso de uso del índice, descrito de forma narrativa. La lista completa de los casos de uso del sistema se puede encontrar en el anexo A del documento.

Tabla 3. Caso de uso: Buscar película.

Caso de uso	**Buscar película**	
Número	1	
Prioridad	Media	
Objetivo	Proporcionar al visitante la información de la película que busca.	
Actores	Visitante	
Precondiciones	Deben haber ítems registrados en el catálogo	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El visitante ingresa el nombre de la película (ítem) que busca. 3. El visitante selecciona uno de los ítems devueltos por el sistema.	2. El sistema presenta la lista de ítems que coinciden con el criterio de búsqueda. 4. El sistema muestra información detallada del ítem seleccionado por el visitante.
Variaciones	No hay ítems registrados en el catálogo. Mostrar un mensaje informativo al visitante. Termina el caso de uso.	

4.1.1.3 Definición de diagramas de secuencia

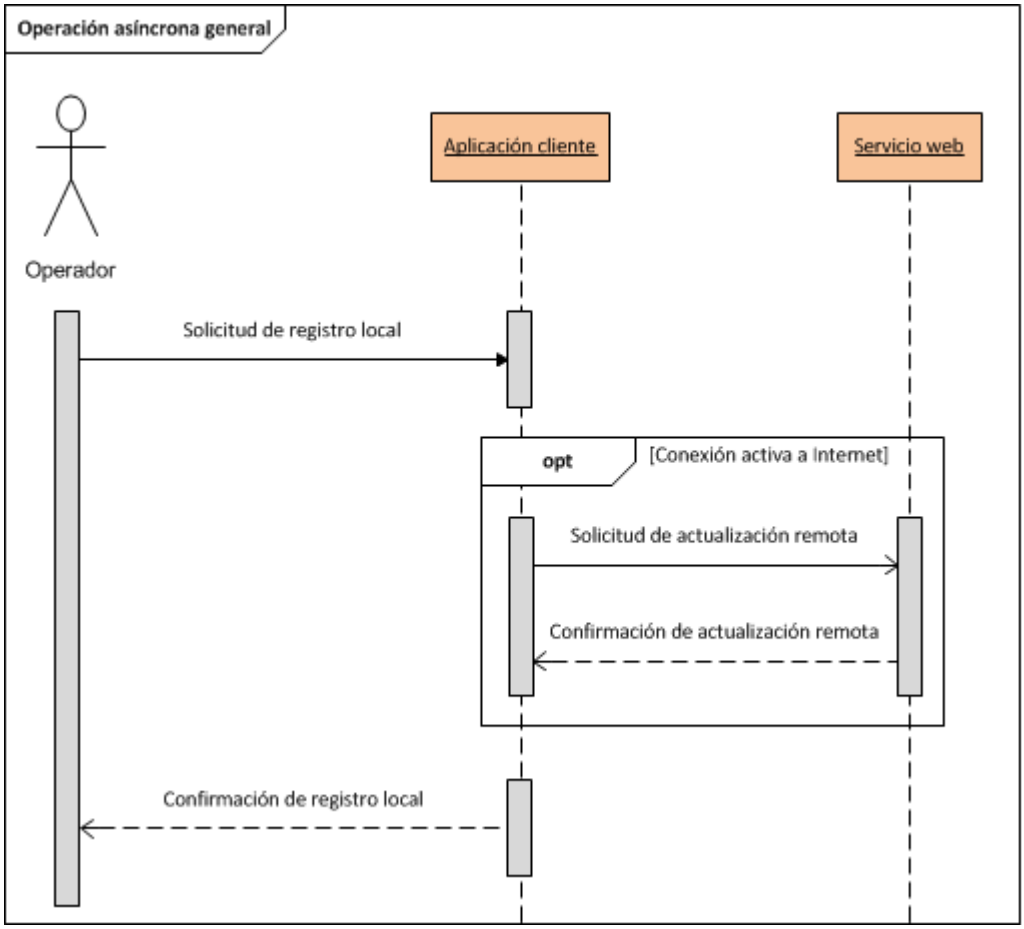
Un diagrama de secuencia es un tipo de diagrama de interacción que muestra cómo los objetos operan unos con otros y en qué orden. Se usa principalmente para mostrar las interacciones entre los objetos en el orden secuencial en que ocurren dichas interacciones. Así pues, el objetivo principal de un diagrama de secuencia es definir secuencias de eventos que llevan a un resultado deseado. El enfoque es menor en los mensajes mismos y mayor en el orden en que ocurren estos mensajes. Sin embargo, la mayoría de los diagramas de secuencia comunicarán qué mensajes son enviados entre los objetos de un sistema así como también el orden en que ocurren. El diagrama transmite esta información a través de las dimensiones horizontales y verticales. La dimensión vertical muestra, de arriba hacia abajo, la secuencia de tiempo de mensajes y llamadas a medida que ocurren, y la dimensión horizontal muestra, de izquierda a derecha, las instancias de los objetos a los que se envían los mensajes.[3]

Con este tipo de diagrama se buscó mostrar de forma clara las interacciones ocurridas entre los diferentes componentes del sistema al momento de realizar tareas que requieren soporte de los servicios web implementados. Estas interacciones en particular tienen especial importancia por estar relacionadas con requerimientos impuestos por el cliente, como se listará en secciones posteriores de este documento y además debido a que su definición ayudará a dar forma de manera indirecta a la arquitectura general de la aplicación.

De esta forma se definió el siguiente diagrama de secuencia que exhibe las interacciones de una operación asíncrona general iniciada por el usuario operador de la aplicación cliente.

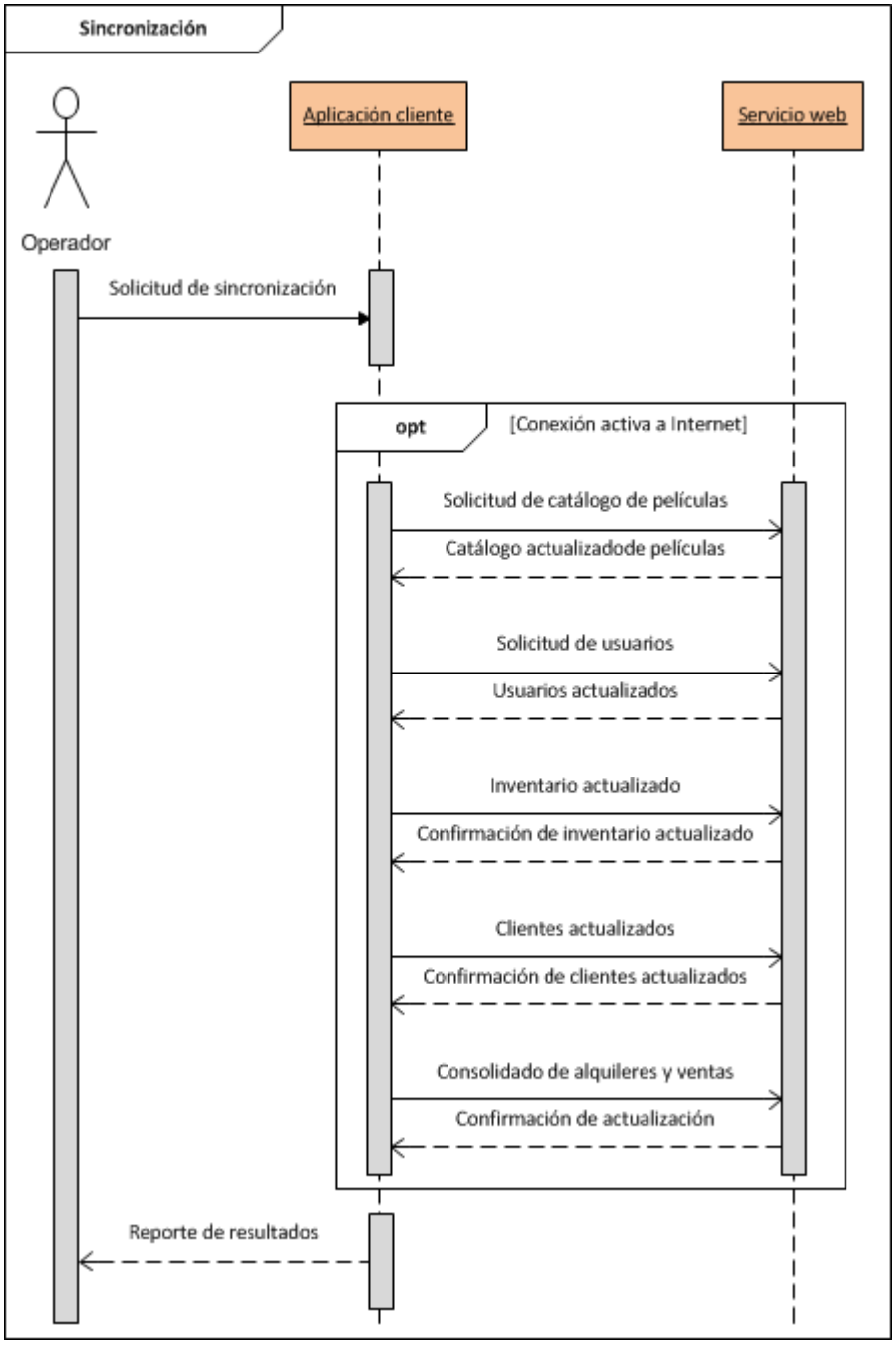
[3] UML basics: The sequence diagram. Donald Bell, IT Architect, IBM Corporation. http://www.ibm.com/developerworks/rational/library/3101.html

Figura 9. Diagrama de secuencia: Operación asíncrona general



En el diagrama se manifiesta que las solicitudes son iniciadas por el actor operador del sistema. Las comunicaciones entre la aplicación cliente y los servicios web dependen de la disponibilidad de una conexión activa a Internet. De forma general, una solicitud de actualización es enviada a la aplicación web de forma remota usando servicios web. El retorno de confirmación permite a la aplicación cliente decidir si puede continuar con las operaciones de actualización. Es importante destacar que el orden en que se envían y reciben los objetos actualizados es vital para el correcto funcionamiento de la operación de sincronización, como se detalla en el siguiente diagrama de secuencia.

Figura 10. Diagrama de secuencia: Sincronización



Se consideró que más diagramas de secuencia en este punto del proyecto no eran necesarios para proporcionar más información que la descrita en forma narrativa en los casos de uso de la sección anterior. Lo anterior, pretende seguir recomendaciones de prácticas ágiles de desarrollo de software que a lo largo de este proyecto se seguirán tratando de aplicar. En particular, se menciona como un error común gastar una buena parte de tiempo del proyecto intentado crear un conjunto completo de diagramas de secuencia para cada uno de los casos de uso del sistema, uno para el curso básico de acción y otro para cada curso alternativo. Para este tipo de situaciones, la recomendación es crear un diagrama de secuencia solamente cuando se tenga lógica compleja que se deba examinar detalladamente. Si la lógica es clara y concisa, el diagrama de secuencia no agregará ningún valor.[4] Sin embargo, es preciso hacer énfasis en que estas recomendaciones no implican que se deba descartar la realización de diagramas de secuencia con el único objetivo de ahorrar tiempo en el cronograma del proyecto, porque los mismos pueden proporcionar una vista más clara de información que aún puede ser confusa para el cliente en los casos de uso.

4.1.1.4 Definición de diagramas de estado

Los diagramas de estados son una técnica conocida para describir el comportamiento de un sistema. Describen todos los estados posibles en los que puede estar un objeto particular y la manera en que cambia el estado del objeto como resultado de los eventos que llegan a él. En la mayor parte de las técnicas orientadas a objetos, los diagramas de estados se dibujan para una sola clase, mostrando el comportamiento de un solo objeto durante todo su ciclo de vida.[5]

[4] Ambler, Scott. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 2004.

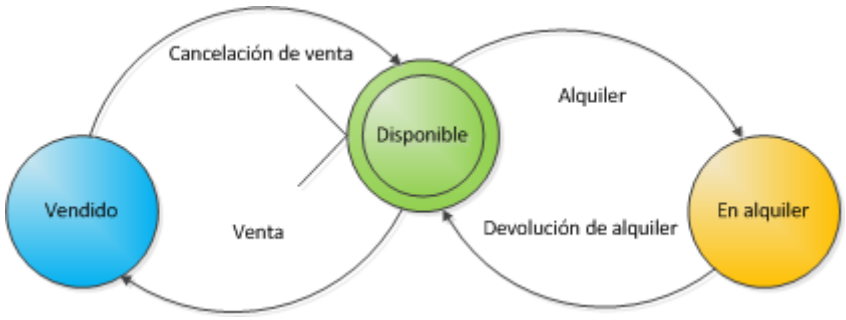
[5] Fowler, Martin. UML Distilled. Addison-Wesley Professional, 2003.

Los diagramas de estado permiten entender mejor clases complejas, particularmente aquellas que se comportan de forma diferente de acuerdo a su estado. Para este proyecto, se consideró importante aclarar el comportamiento del inventario y la forma en que cambia su estado de acuerdo a los alquileres y ventas.

Figura 11. Diagrama de estado general de inventario

El estado *No disponible* impide que el ítem de inventario en cuestión sea usado en un proceso de venta o alquiler, hasta que una devolución de alquiler o cancelación de venta tenga lugar. Sin embargo, para obtener mayor claridad, es posible desglosar el estado *No disponible* de acuerdo al tipo de proceso realizado.

Figura 12. Diagrama de estado detallado de inventario



4.1.2 Requerimientos no funcionales

Los requerimientos no funcionales describen aspectos del sistema que son visibles por el usuario pero que no incluyen una relación directa con el comportamiento funcional del sistema. Los requerimientos no funcionales incluyen restricciones como el tiempo de respuesta (desempeño), la precisión, recursos consumidos, seguridad, entre otros. A los requerimientos no funcionales se les suele llamar *cualidades del sistema*. Otros términos por los cuales se les conoce son: restricciones, atributos de calidad, objetivos de calidad y requerimientos de calidad del servicio. [6]

Las siguientes cualidades requeridas en el sistema a desarrollar, fueron especificadas para este proyecto:

- El sistema debe ser capaz de registrar alquileres de películas y ventas de productos aunque no haya conexión a Internet.
- La disponibilidad de las películas que se muestran en el catálogo consultado en la sección pública de la aplicación web, es información para la cual se debe solicitar actualización en periodos de una hora, mientras haya una conexión activa a Internet en la aplicación cliente.

4.1.3 Requerimientos impuestos

También llamados pseudo-requerimientos, son requerimientos impuestos por el cliente que restringen la implementación del sistema. Entre los ejemplos comunes se encuentran el lenguaje de programación, la plataforma en que el sistema debe

[6] Object Oriented Software Engineering. Bernd Bruegge y Allen H. Dutoit. Prentice Hall, 2000.

ser implementado y requerimientos del proceso de documentación (por ejemplo, la utilización de un lenguaje formal específico). Los siguientes requerimientos fueron impuestos al desarrollo de este proyecto:

- El sistema debe funcionar en la web (aplicación web) y en una computadora local (aplicación cliente).
- La aplicación cliente debe funcionar en el sistema operativo Windows.
- La aplicación web debe funcionar en el navegador de Internet Firefox.

4.1.4 Requerimientos inversos

Este tipo de requerimientos establece lo que el sistema no debe hacer por encontrarse fuera de su alcance. Es natural que para cada proyecto, exista un número ilimitado de requerimientos inversos, de modo que los que se enuncian son los que permiten aclarar requerimientos reales y evitar posibles confusiones.

Los siguientes requerimientos inversos fueron definidos para el desarrollo de este proyecto.

- El sistema no contará con una interfaz de usuario definitiva para uso comercial. Contará con una interfaz de usuario básica que permita demostrar la funcionalidad proporcionada. Sin embargo, en un eventual desarrollo posterior a este proyecto, debe ser posible construir una interfaz de usuario mejorada sin afectar la funcionalidad debido al diseño de la arquitectura de la aplicación que se tratará más adelante en este documento.
- No se realizará registro o control de crédito financiero de los usuarios clientes de la aplicación.

4.2 MODELO DE DOMINIO

Un modelo de dominio es un modelo conceptual de todos los elementos relacionados a un sistema o problema específico que incorpora tanto el comportamiento como los datos. Un modelo de dominio crea una red de objetos interconectados, donde cada objeto representa un individuo significativo, ya sea tan grande como una corporación o tan pequeño como una única línea en una orden de pedido.[7]

Este proceso de análisis del sistema orientado a objetos fue guiado por lo expuesto en el texto *Object-Oriented Systems Analysis and Design With UML* de los autores Robert Stumpf y Lavette Teague, mediante la elaboración de un modelo de dominio como punto de partida para el desarrollo del diseño de la base de datos y cualquier otro tipo de modelo posterior que pueda aprovechar la información definida.

En primer lugar, se identifican los elementos llamados conceptos, que son abstracciones de cosas, personas o ideas relevantes para el sistema diseñado.

Figura 13. Conceptos básicos del modelo de dominio

[7] Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

Posteriormente, se procede a identificar los atributos de cada elemento listado. Los atributos son las características de un concepto que pueden tener valor. Cuando un atributo requiere de otros atributos para ser descrito de la forma requerida, se trata como un concepto que se puede definir de forma independiente. De esta forma, se pueden encontrar nuevos conceptos que antes no eran evidentes.

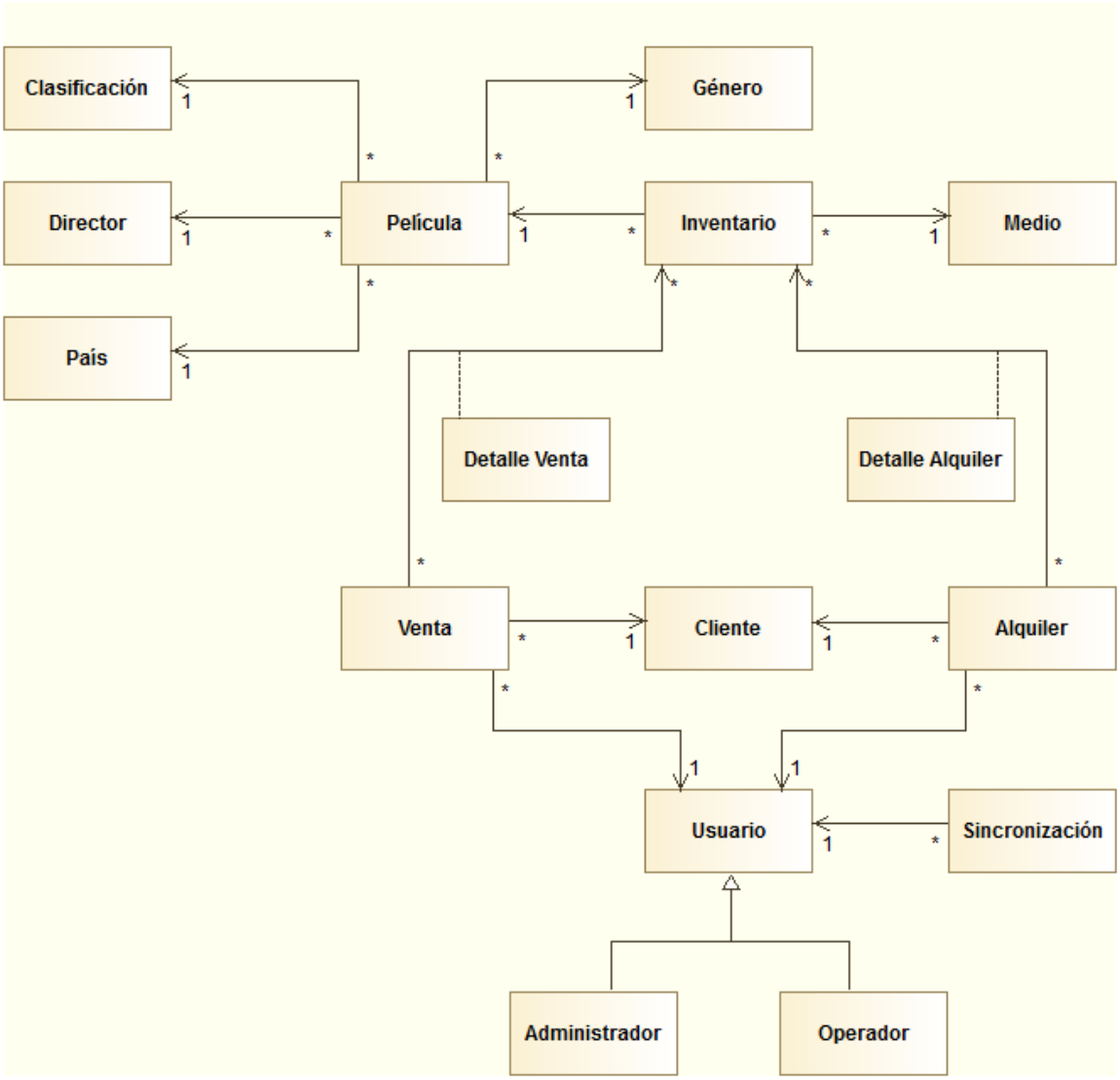
Figura 14. Conceptos extendidos del modelo de dominio

Una vez se tienen los conceptos identificados con sus respectivos atributos, se procede a definir sus asociaciones. Una asociación es una conexión significativa entre conceptos. Por consideraciones de legibilidad, los atributos no están especificados en este punto del documento. Sin embargo, es posible consultar la lista completa y detallada de los atributos posteriormente en el capítulo del diseño de la base de datos.

A continuación, se muestra el diagrama del modelo de dominio resultante, que incluye la multiplicidad de las asociaciones identificadas. La multiplicidad de una

asociación es el número de instancias de un concepto que pueden ser asociadas con una instancia de otro concepto.

Figura 15. Diagrama de modelo de dominio

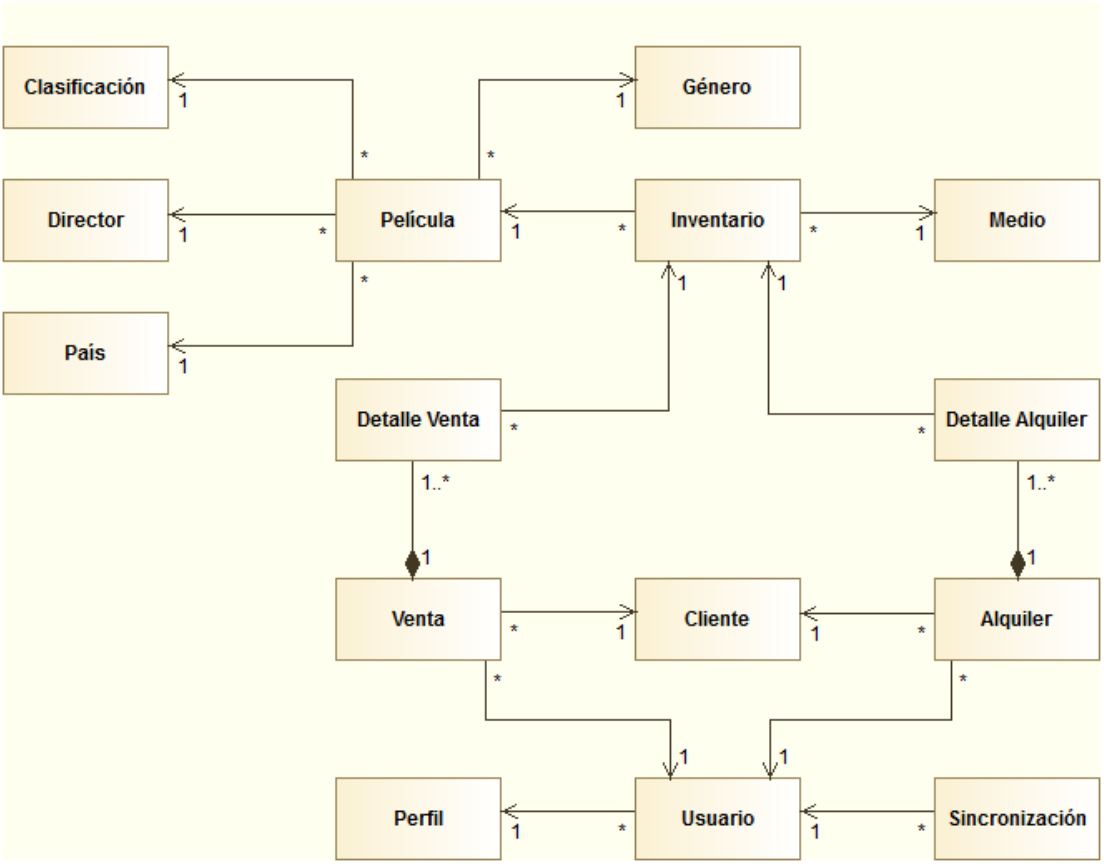


Se identifica la existencia de conceptos de asociación en el diagrama resultante. Un concepto de asociación permite incluir uno o más atributos que dependen de

una asociación de clases. Esta condición se presenta entre los conceptos *Venta* e *Inventario* y entre *Alquiler* e *Inventario*. De igual forma, se identifica una jerarquía de generalización en el concepto de *Usuario*. Una jerarquía de generalización clasifica un tipo de concepto en sus subtipos.

El modelo de dominio es lo suficientemente flexible como para sus partes puedan ser rescritas de forma que permita una mayor claridad de acuerdo al uso deseado. Para el caso de este proyecto, una de las principales funciones para el modelo de dominio es proporcionar una base conceptual que facilite el posterior diseño de la base de datos, de manera que el diagrama definitivo se planteó de la forma que se muestra a continuación.

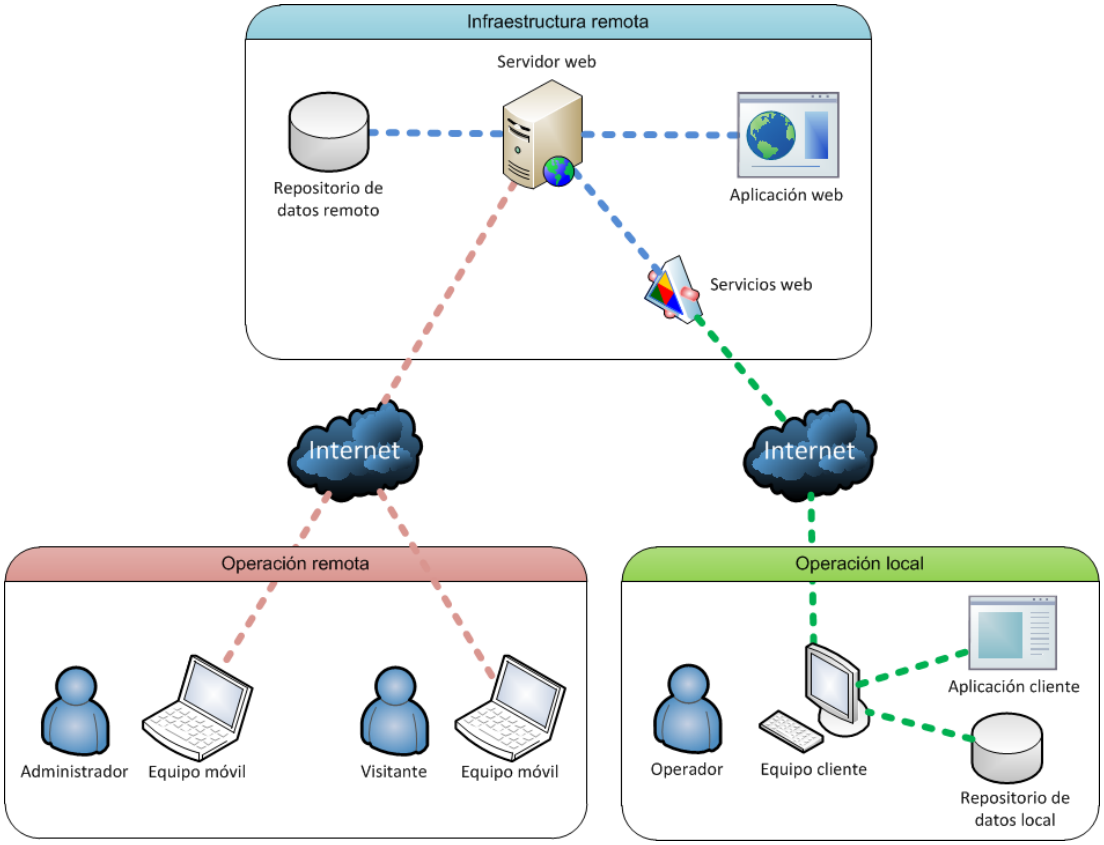
Figura 16. Diagrama extendido de modelo de dominio



4.3 ARQUITECTURA DE LA APLICACIÓN

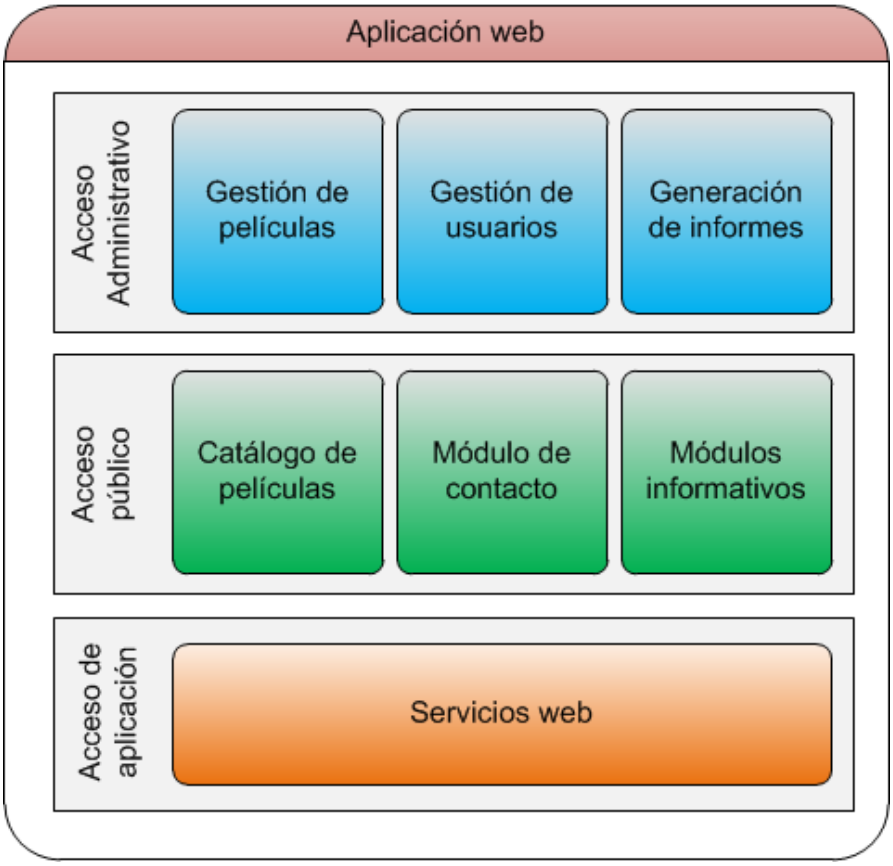
La definición de la arquitectura de la aplicación se basó en modelos generales de aplicaciones distribuidas, teniendo en cuenta los requerimientos específicos de infraestructura física de este proyecto. Las siguientes figuras representan el resultado de la definición de la arquitectura general de la aplicación, tanto en el ámbito de la web como en el ámbito local. En todos los casos, los servicios web son identificados como base fundamental en las comunicaciones de los diferentes módulos del sistema. En el caso de la operación realizada a nivel local, el diseño se definió de tal forma que pueda seguir en funcionamiento normal aun cuando se deje de tener una conexión activa a Internet que permita la comunicación con los demás componentes del sistema.

Figura 17. Arquitectura general de la aplicación



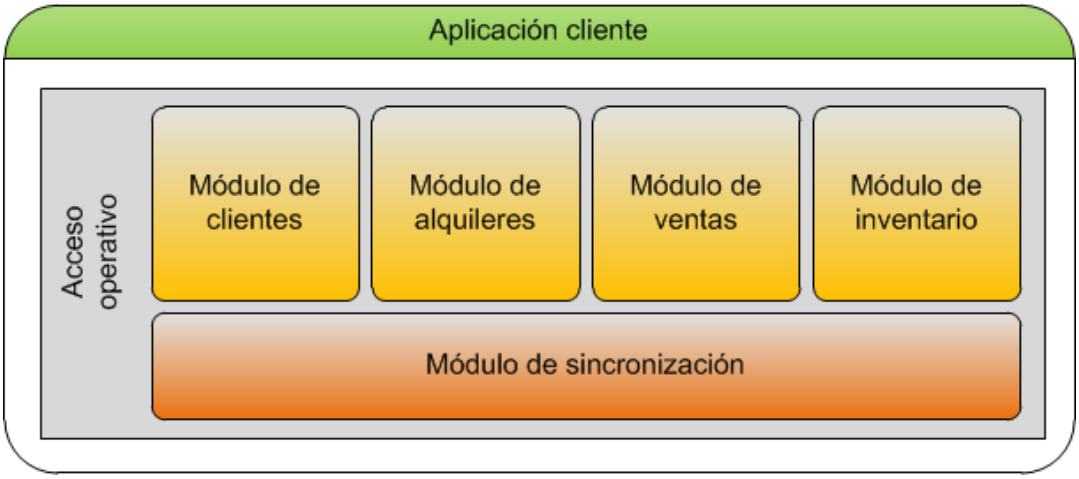
Los servicios web quedan definidos como el puente de comunicación entre la aplicación cliente local y la aplicación web remota. De forma general, ellos permitirán a estas dos partes del sistema hablar entre sí. Sin embargo, no son requeridos para que los actores definidos como administrador y visitante, accedan a la funcionalidad ofrecida por las instancias remotas del sistema.

Figura 18. Componentes de la aplicación web



Los usuarios con rol de operador del sistema no están contemplados para proporcionarles acceso a instancias remotas de la aplicación. El papel que cumplen en el sistema está previsto para desarrollarse en instancias locales de la aplicación.

Figura 19. Componentes de la aplicación cliente



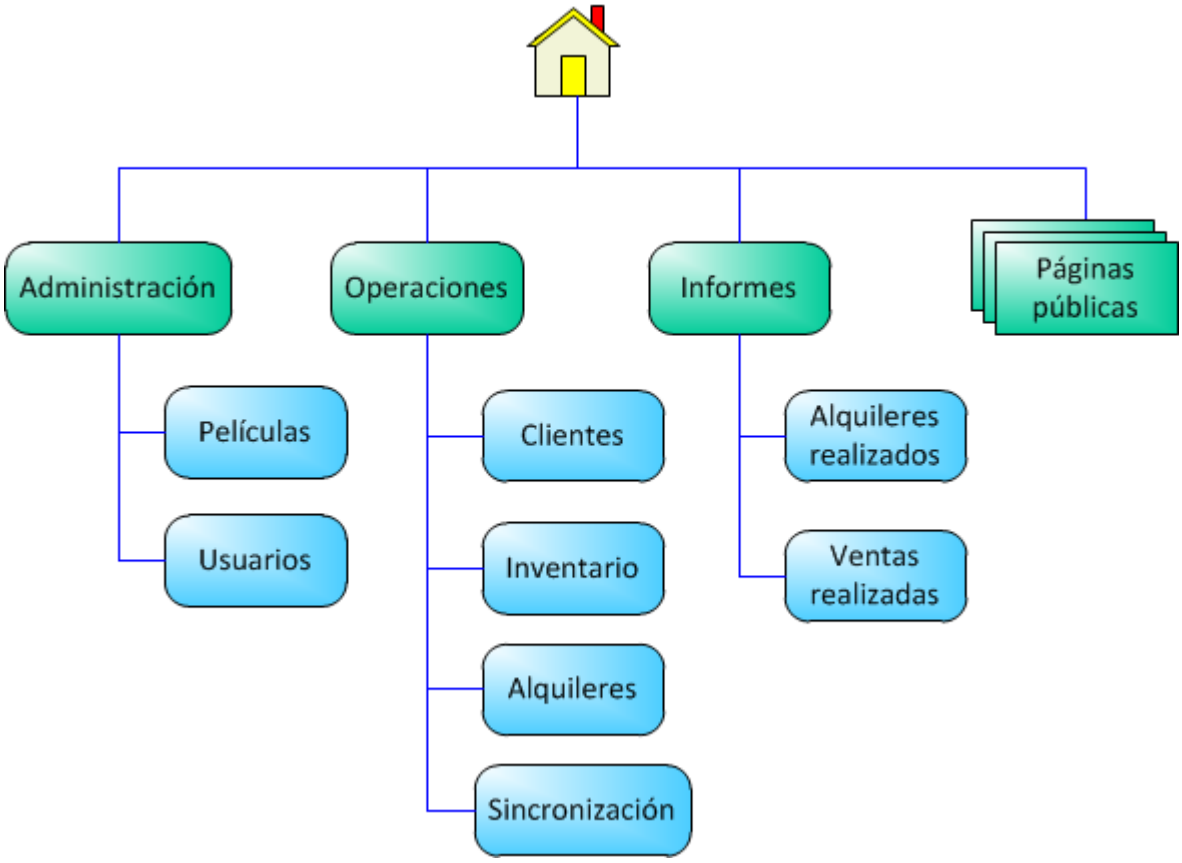
El módulo de sincronización usa los recursos proporcionados por los servicios web para llevar a cabo sus funciones. Los demás módulos de la aplicación cliente no requieren de una conexión activa a Internet para funcionar, de modo que las funciones básicas están siempre disponibles para los operadores del sistema.

4.4 ESTRUCTURA DE NAVEGACIÓN DE PÁGINAS

La estructura de navegación de las páginas de la aplicación fue definida de forma que los módulos principales de la aplicación pudieran ser accedidos de forma rápida para evitar interrumpir el flujo de trabajo de los usuarios.

Se buscó que la categorización fuera sencilla e intuitiva. De igual forma, se buscó que todas las funciones consideradas como básicas, no estuvieran a una mayor profundidad de dos jerarquías en la estructura de navegación.

Figura 20. Diagrama de navegación de páginas



En el diagrama anterior, las páginas públicas se refieren al conjunto de páginas que son accesibles sin necesidad de autenticarse como usuario en la aplicación. Entre ellas se encuentran la página de inicio, la búsqueda de películas disponibles, el formulario de contacto y la página de inicio de sesión.

Las páginas incluidas adentro de la jerarquía de nivel dos se omiten de la estructura de navegación que se muestra visualmente en la interfaz de usuario para evitar proporcionarle información no relevante ni necesaria para la correcta navegación de la aplicación.

4.5 PERMISOS DE USUARIO

La autenticación y autorización de usuarios de la aplicación se lleva a cabo mediante control de acceso basado en perfiles. Cada usuario obtiene permisos de acceso a módulos según el perfil al que pertenece, de acuerdo a la siguiente tabla.

Tabla 4. Permisos de usuario

Módulo \ Perfil	**Administrativo**	**Operativo**	**Todos***
Administración	Sí	No	No
Operaciones	Sí	Sí	No
Informes	Sí	No	No
Básicos	Sí	Sí	Sí

* Usuarios autenticados y no autenticados

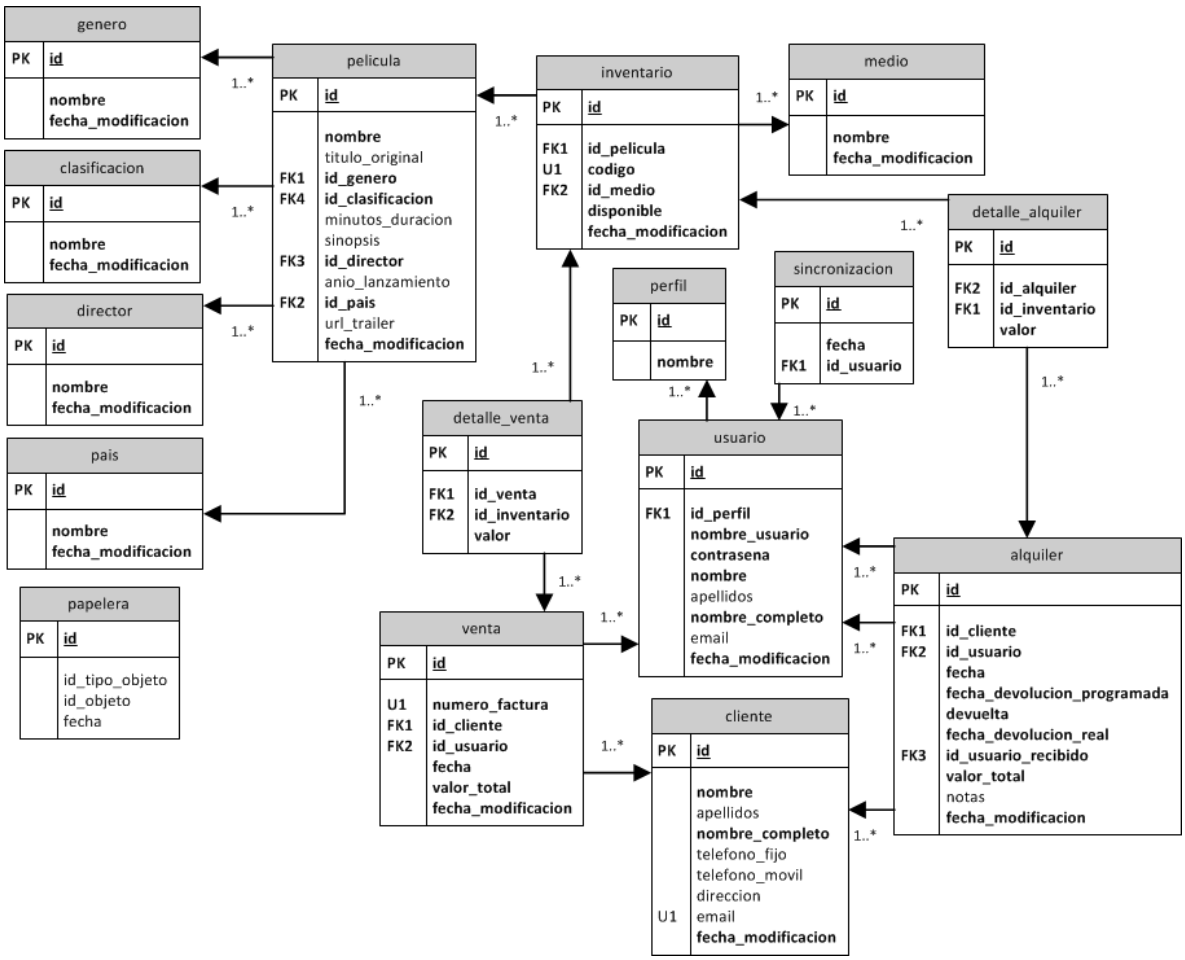
El módulo de administración incluye los siguientes elementos: usuarios y películas, las cuales incluyen también la edición de géneros, clasificaciones, directores, países y medios. El módulo de operaciones incluye los siguientes elementos: clientes, inventario, alquileres, ventas y sincronización. El módulo de informes incluye los siguientes elementos: ventas realizadas y alquileres realizados. El módulo básico de páginas públicas incluye los siguientes elementos: página de inicio, páginas de error, página de contacto y página de inicio de sesión.

Por consideraciones de seguridad, todos los módulos requieren de la inclusión explicita de reglas de que otorguen acceso a un perfil determinado. De esta forma, se evita el acceso a recursos de la aplicación hasta que se asignen permisos de forma explícita.

4.6 DISEÑO DE LA BASE DE DATOS

En esta etapa se realizó la definición del modelo de datos que soportara los requerimientos de almacenamiento de información de la aplicación. El diseño se basó en el modelo de dominio definido previamente y está representado mediante el modelo entidad-relación asegurando la integridad referencial y evitando la duplicidad de la información. A continuación, se muestra el modelo general de la base de datos. Información más detallada como la descripción de las tablas y diagramas adicionales se puede encontrar en el anexo B de este documento.

Figura 21. Diagrama general de la base de datos



4.7 TECNOLOGÍAS DE DESARROLLO

Se definieron los siguientes parámetros para realizar la selección de las tecnologías de desarrollo a utilizar, de acuerdo a la especificación de requerimientos del proyecto.

- **No debe requerir compra de licencia para uso comercial.** Un eventual producto final derivado del conocimiento proporcionado por este proyecto, está dirigido a pequeñas y medianas empresas que pueden no tener recursos suficientes para adquirir licencias de uso comercial que usualmente pueden llegar a costar cientos o miles de dólares en el mercado actual.
- **La instalación y configuración de la infraestructura debe ser sencilla.** Una parte del sistema está destinado a ser usado de forma local en máquinas cliente, por lo que es deseable que ante futuros cambios de hardware, la instalación y configuración (*deployment*) sea un proceso que consuma la menor cantidad posible de tiempo y recursos del cliente.
- Debe tener soporte de operaciones basadas en servicios web.
- Debe soportar el paradigma de desarrollo orientado a objetos (OOP).
- Debe tener soporte para el patrón de diseño MVC[8].

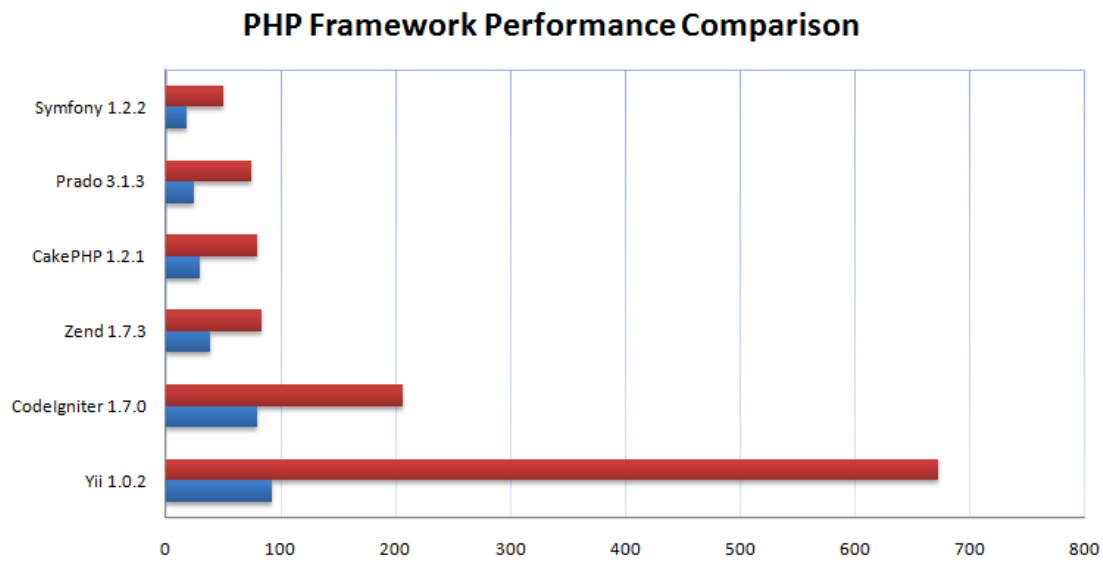
Con los anteriores parámetros siendo tenidos en cuenta, se seleccionó a PHP como el lenguaje de programación principal del proyecto debido a su bajo costo de adquisición ya que viene pre-instalado en la mayoría de servidores web compartidos de propósito general, quienes suelen ofrecer el conjunto de solución LAMP[9] por parte de los proveedores de alojamiento web.

[8] Modelo Vista Controlador (MVC) es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocios en tres componentes distintos.

[9] LAMP es un conjunto de soluciones de software libre de código abierto que incluye: Linux (sistema operativo), Apache (servidor web), MySQL (gestor de base de datos) y PHP (lenguaje de programación).

Así mismo, se seleccionó a yii como el *framework* de desarrollo de PHP, por cumplir con los requerimientos de funcionalidad buscados. En particular, ofrece soporte para operaciones con servicios web y se basa en el patrón de diseño MVC. También ofrece otras funcionalidades útiles aunque no de vital interés para este proyecto, como por ejemplo, soporte integrado de autenticación y autorización, integración con jQuery, internacionalización y localización, control y registro personalizado de errores, entre otros. El desempeño de yii ha sido comparado con otros framework de PHP en pruebas *benchmark* especializadas. Una de estas pruebas de comparación se incluye en este informe solo como referencia. En el gráfico que se muestra a continuación, RPS significa "peticiones por segundo" (por sus siglas en inglés) y describe cuántas peticiones por segundo puede procesar una aplicación escrita en un framework determinado. Entre más alto el número, más eficiente es. El desempeño de yii es especialmente superior cuando la extensión APC (Alternative PHP Cache) de PHP está activada.

Figura 22. Desempeño de frameworks PHP



	Yii 1.0.2	CodeIgniter 1.7.0	Zend 1.7.3	CakePHP 1.2.1	Prado 3.1.3	Symfony 1.2.2
RPS with APC	673	206	83	79	75	50
RPS without APC	93	79	38	29	24	18

Fuente: http://www.yiiframework.com/performance/

A continuación se detalla la lista completa de tecnologías seleccionadas para el desarrollo del proyecto.

- Lenguaje de programación principal: PHP 5.3.10
- Framework de desarrollo: yii 1.1.10
- Gestor de base de datos: MySQL 5.5.23
- Servidor web: Apache HTTP Server 2.2.22
- Sistema operativo de producción: Linux CentOS 6.2
- Sistemas operativos de desarrollo: Ubuntu 12.04 y Windows 7
- Entorno de desarrollo integrado: NetBeans IDE 7.1.2
- Sistema de control de versiones distribuido: Git 1.7.10
- Navegador web de pruebas: Mozilla Firefox 14

Es preciso resaltar que este proyecto no está orientado a la demostración de una tecnología de desarrollo particular, como se puede notar a partir de sus objetivos documentados, de manera que las selecciones se basaron en los elementos que mejor se adaptaran a los requerimientos del proyecto permitiendo la realización de tareas en un tiempo ajustado al estimado en el cronograma de actividades de la etapa de planeación, teniendo en cuenta la experiencia con las tecnologías candidatas que los autores han tenido previamente al desarrollo de este proyecto.

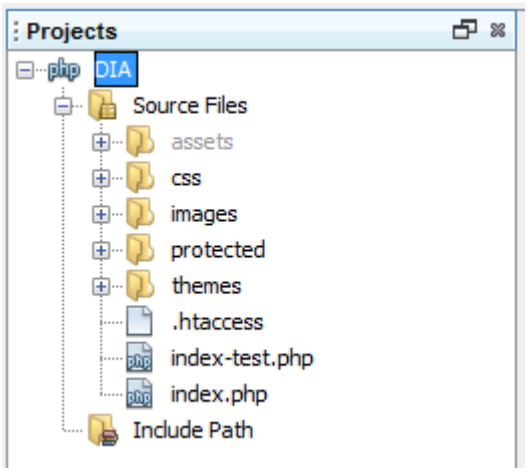
5. CONSTRUCCIÓN DE LA APLICACIÓN

En esta etapa se realizó la implementación de todos los aspectos definidos en las etapas anteriores del proyecto.

5.1 CONVENCIONES USADAS

Los archivos de código fuente se estructuraron de acuerdo a las convenciones del framework Yii[10].

Figura 23. Estructura de directorios de la aplicación



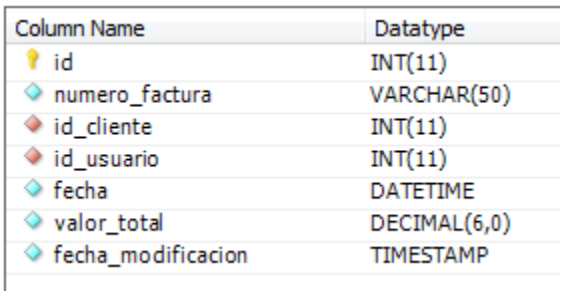
- **assets:** Contiene archivos autogenerados en tiempo de ejecución.
- **css:** Contiene archivos de estilo (CSS) de la aplicación.
- **images:** Contiene archivos de imágenes.

[10] The Definitive Guide to Yii. Yii Software LLC. 2010. http://www.yiiframework.com/doc/guide/1.1/en/basics.convention

- **protected:** Es el directorio principal que contiene los archivos de código fuente relacionados con los modelos, los controladores y las vistas de la aplicación. También es el lugar donde se ubican los componentes de terceros.
- **themes:** Contiene archivos que permiten agrupar los estilos visuales de la aplicación en un conjunto independiente llamado tema.

No sólo se siguieron las convenciones recomendadas en la estructura de directorios de código fuente, sino también en otras áreas de la aplicación como estilos de codificación, nombres de los elementos de la base de datos, formatos de URL y los demás mencionados en la documentación oficial del framework. Seguir este tipo de indicaciones permite agilizar el desarrollo al evitar escribir y administrar configuraciones complejas que adapten la tecnología a convenciones personalizadas.

Figura 24. Ejemplo de convenciones de nombres de campos de base de datos.



Column Name	Datatype
id	INT(11)
numero_factura	VARCHAR(50)
id_cliente	INT(11)
id_usuario	INT(11)
fecha	DATETIME
valor_total	DECIMAL(6,0)
fecha_modificacion	TIMESTAMP

5.2 SISTEMA DE CONTROL DE VERSIONES

Control de versiones es como se denomina a un sistema que registra los cambios hechos a un conjunto de archivos en el tiempo de manera que se puedan recuperar versiones específicas después. Este tipo de sistemas son de gran utilidad en proyectos de desarrollo de software al permitir revertir el sistema en

desarrollo a un estado previo, comparar los cambios que se han hecho en el tiempo, revisar quién ha introducido *bugs* en el código fuente, cuándo y cómo, entre otras cosas.

Para este proyecto se usó Git, un sistema de control de versiones distribuido (DVCS, por sus siglas en inglés). Los sistemas de control de versiones centralizados (CVCS), como por ejemplo Subversion, CSV y Perforce, tienen un solo servidor que contiene todos los archivos controlados, actuando como una ubicación central desde donde los clientes obtienen la última versión de los archivos.

Aunque sea fácil de administrar, una configuración centralizada tiene desventajas importantes. Una de las más obvias es el punto simple de falla que el servidor centralizado representa. Si dicho servidor dejara de funcionar durante un tiempo, nadie podría guardar nuevas versiones de archivos durante ese tiempo. Si el disco duro del servidor centralizado se dañara irreparablemente y no se contaran con copias de respaldo, se perdería absolutamente todo el historial de versiones de los archivos, a excepción de las versiones que llegaran a tener los clientes en sus máquinas locales.

En un sistema de control de versiones distribuido, como por ejemplo Git, Mercurial o Bazaar, cada cliente cuenta con el repositorio completo de los archivos, de manera que si un servidor dejara de funcionar, todos los clientes pueden seguir guardando nuevas versiones de archivos a nivel local y además, cualquier cliente tendría la capacidad de restaurar el repositorio de vuelta al servidor en caso de que, por ejemplo, su disco duro haya sido remplazado.[11]

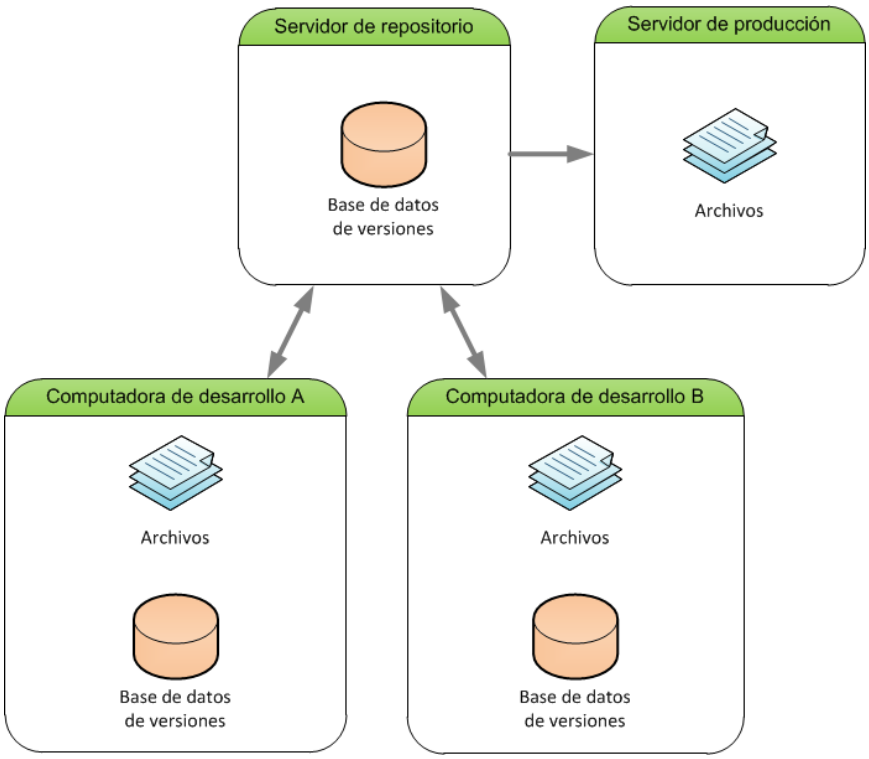
[11] Scott Chacon. Pro Git. Apress, 2009.
http://git-scm.com/book/en/Getting-Started-About-Version-Control

Figura 25. Ejemplo de historial de versiones en Git

Revision	Message	Date
5d3667dd0267	UI formulario contáctenos	5 days ago
e865a516a9e1	Botón de regresar movido	5 days ago
fa79adf46999	Renombrada clase de estilo	7 days ago
534c9ba52bbc	UI de cuadro de búsqueda pública de películas	7 days ago
2f1195a23346	Visualización de disponbilidad de películas en la búsqueda pública	7 days ago
caccd86d5205	Integración de YouTube, estilos de CDetailView.	11 days ago
957ab716ccaf	Módulo público para buscar películas	11 days ago

La arquitectura distribuida de Git permite diseñar diferentes flujos de trabajo de acuerdo a los requerimientos de los desarrolladores. En este proyecto, se usó el flujo de trabajo que se muestra en la figura a continuación.

Figura 26. Flujo de trabajo con Git



En la figura anterior se muestra que los desarrolladores tienen a su disposición un servidor de repositorio que comparten para mantener actualizados sus repositorios de trabajo local. El servidor de producción recibe directamente los archivos desde el servidor de repositorio, aunque no cuenta con base de datos de versiones por consideraciones de seguridad y para permitirle concentrar sus recursos en la ejecución de las instancias remotas de la aplicación.

5.3 CALENDARIO DE DISEÑO Y DESARROLLO

Durante el desarrollo de este proyecto se realizó un registro detallado del tiempo empleado para la realización de las tareas necesarias en cada etapa, de forma que se pudiera realizar un control sobre el calendario del proyecto y usarlo como una herramienta que permitiera detectar y corregir a tiempo cualquier desfase respecto al cronograma previsto en la etapa de planeación. De igual forma, se buscó un enfoque de desarrollo incremental e iterativo.

Con el desarrollo incremental, se desarrolla la funcionalidad de la aplicación en varios grupos pequeños a la vez. El desarrollo iterativo sucede cuando se llevan a cabo las tareas del desarrollo en pequeños y repetitivos ciclos llamados iteraciones. El fin de una iteración marca un hito en el calendario. Sin embargo, en ese momento el producto puede no estar disponible todavía para uso real. Un incremento del producto se completa cuando al final de una iteración el producto está listo para uso real. Cada incremento generalmente incluye varias iteraciones.

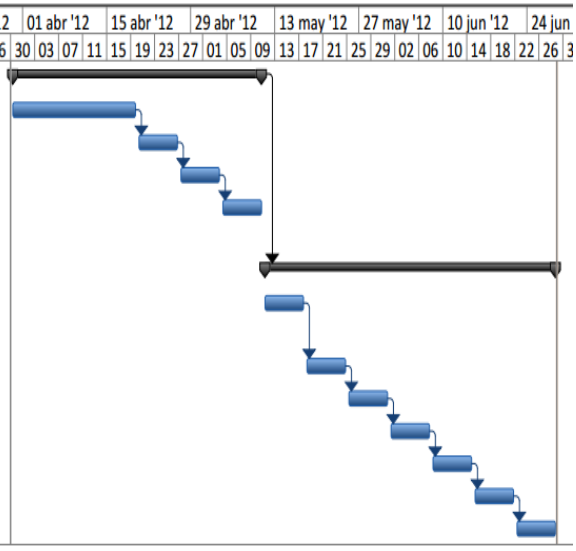
La idea de afrontar un proyecto largo dando pequeños pasos es clave para un enfoque ágil en el desarrollo. Los saltos grandes incrementan el riesgo, mientras que los pequeños pasos ayudan a mantener el balance. [12]

[12] Hunt, Andy. Subramaniam, Venkat. Practices of an Agile Developer. Pragmatic Bookshelf, 2006.

La duración de cada iteración depende del tamaño de cada proyecto y la complejidad de los objetivos para cada iteración puede ser refinada de acuerdo a los resultados registrados al final de cada ciclo. Para este proyecto, se definieron iteraciones de una semana de duración y los resultados se ilustran en la figura que se muestra a continuación.

Figura 27. Calendario de diseño y desarrollo

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Diseño de la aplicación**	**42 días**	**vie 30/03/12**	**jue 10/05/12**
2	Especificación de requerimientos	3 sem.	vie 30/03/12	jue 19/04/12
3	Definición de arquitectura	1 sem	vie 20/04/12	jue 26/04/12
4	Diseño de la base de datos	1 sem	vie 27/04/12	jue 03/05/12
5	Selección de tecnologías de desarrollo	1 sem	vie 04/05/12	jue 10/05/12
6	**Construcción de la aplicación**	**49 días**	**vie 11/05/12**	**jue 28/06/12**
7	Configuración de plataformas de desarrollo	1 sem	vie 11/05/12	jue 17/05/12
8	Módulos tipo CRUD	1 sem	vie 18/05/12	jue 24/05/12
9	Interfaz de usuario	1 sem	vie 25/05/12	jue 31/05/12
10	Módulos de procesos	1 sem	vie 01/06/12	jue 07/06/12
11	Implementación de sincronización	1 sem	vie 08/06/12	jue 14/06/12
12	Módulos de acceso público	1 sem	vie 15/06/12	jue 21/06/12
13	Integración de componentes de UI	1 sem	vie 22/06/12	jue 28/06/12



5.4 CONSIDERACIONES DE LA INTERFAZ DE USUARIO

Durante la construcción de la interfaz de usuario se tuvieron en cuenta conceptos guías para el diseño de las pantallas de navegación y los elementos visuales. El objetivo fue lograr una interfaz que fuera clara, sencilla, consistente e intuitiva sin que eso afectara la funcionalidad que se buscó ofrecer. A continuación se muestran ejemplos que ilustran los conceptos que se consideraron durante el desarrollo de la interfaz.

En primer lugar, se tiene una barra de navegación superior que actúa como un clasificador de las secciones principales de la aplicación. De sus opciones se derivan menús de primer nivel distribuidos en forma de cuadrícula con íconos que permiten su rápida identificación y ayudan a agilizar mentalmente su ubicación en la pantalla.

Figura 28. Elementos en pantalla de primer nivel

Las opciones del menú de primer nivel derivan en una pantalla de segundo nivel que convierte las opciones del menú de la pantalla anterior, en una barra de

navegación lateral, actuando de forma efectiva como un acceso directo a las opciones sin tener que regresar a la pantalla anterior manualmente. En esta pantalla, los botones cuentan con un estilo que se estandarizó en toda la aplicación para mantener un diseño consistente. De igual forma, se cuentan con dos tipos de selectores de objetos, uno para objetos simples y otro para objetos compuestos.

Figura 29. Elementos en pantalla de segundo nivel

Los objetos simples son aquellos que sólo cuentan con un atributo descriptivo mostrado en pantalla. Los objetos compuestos son aquellos que requieren un nivel de elaboración mayor en su presentación debido a que cuentan con varios atributos descriptivos que se deben mostrar en pantalla. En algunos casos, sólo se

muestra la lista completa de atributos, una vez han sido seleccionados de forma individual.

Figura 30. Selector de objetos compuestos

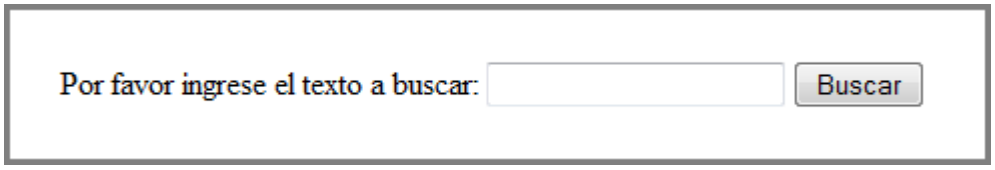
Para los tipos compuestos de objetos, también se cuenta con una vista de tabla que permite visualizar una mayor cantidad de elementos al mismo tiempo, a la vez que facilita su búsqueda mediante opciones de filtrado especializadas ubicadas en la cabecera de la tabla.

Figura 31. Vista de tabla de objetos compuestos

Un ejemplo adicional de la interfaz simplificada que se buscó desarrollar, se encuentra en el cuadro de búsqueda. Tradicionalmente, este cuadro cuenta con una etiqueta de texto que enuncia su función y además un botón con un texto adicional que indica que su función es "buscar".

Estos dos elementos fueron sustituidos por un ícono de fondo que sugiere la función del cuadro de texto, al mismo tiempo que el texto de la pestaña del menú de nivel superior actúa como una confirmación de la función que realiza.

Figura 32. Ejemplo de cuadro de búsqueda tradicional



El anterior ejemplo de un cuadro tradicional de búsqueda puede compararse visualmente con el realizado bajo un parámetro de diseño simplificado en este proyecto.

Figura 33. Cuadro de búsqueda de diseño simplificado

Estas diferencias en la interfaz de usuario que aparentemente son menores y que en ocasiones no son tenidas en cuenta o se les da poca importancia, pueden hacer la diferencia entre un producto funcional que agrade al usuario y uno que también es funcional pero que el usuario puede llegar a clasificar como deficiente por su apariencia o por lo complicado que puede resultarle realizar las funciones más utilizadas de la aplicación. En este tipo de casos, la percepción del usuario puede afectar seriamente la continuidad del desarrollo de un proyecto o la vida útil de una aplicación en producción y por este motivo es necesario darle una importancia adecuada a las consideraciones de diseño que orientan el desarrollo de la interfaz de usuario de la aplicación.

5.5 CONSIDERACIONES DE SEGURIDAD

La naturaleza de una aplicación distribuida hace que sea importante tener en cuenta prácticas de seguridad informática que eviten interrupciones de los servicios de la aplicación o robo de información privada por parte de agentes externos que puedan aprovechar las vulnerabilidades encontradas en el sistema.

La seguridad informática es un tema amplio, complejo y en continua evolución cuyo análisis extensivo se encuentra fuera del alcance de este proyecto. Sin embargo, se tuvieron en cuenta diversas consideraciones básicas de seguridad que se aplicaron en el desarrollo de la aplicación, las cuales se listan a continuación.

Es importante tener en cuenta que ninguna implementación de una consideración de seguridad puede ofrecer una solución completa o definitiva, pero el uso combinado de estas medidas permite aumentar de forma incremental la postura de seguridad de las aplicaciones web reduciendo la cantidad de vulnerabilidades que puedan ser aprovechadas.

5.5.1 Validación de datos de usuario

La validación permite asegurar que los datos que ingresa el usuario son válidos para el sistema, teniendo en cuenta criterios como el tipo de datos, campos requeridos, longitud de datos, rangos de valores, formatos, etc. Por ejemplo, si se espera el ingreso de un email por parte de un usuario, se debe validar que contenga un formato correcto de email.

La validación de datos de usuario no es una protección de seguridad definitiva, pero es un excelente primer paso en la construcción de una aplicación segura. Estas validaciones pueden realizarse tanto en el lado del cliente como en el lado del servidor. Las validaciones en lado del cliente no proporcionan ningún tipo de seguridad y deben usarse sólo como una forma de mejorar la interfaz, la comodidad de su uso y el alivio de carga de procesamiento en el servidor para errores comunes de validación.

5.5.2 Prevención de cross-site scripting (XSS)

XSS es una vulnerabilidad de seguridad que un atacante puede aprovechar para inyectar código arbitrario, como por ejemplo scripts de JavaScript, VBScript, ActiveX, HTML o Flash, en páginas web que son vistas por otros usuarios, permitiéndole potencialmente robar información privada de dichos usuarios.

Dentro de los métodos que pueden usarse para prevenir el XSS, se encuentran:

- Realizar la conversión de caracteres de escape especiales de HTML. Esto permite que las salidas sean tratadas por el navegador web como texto simple, en lugar de interpretar las etiquetas de scripts.
- "Purificar" el código HTML en caso de que se requiera aceptar código HTML por parte de los usuarios, utilizando componentes especializados para esta tarea, que buscan remover todo el código que sea potencialmente peligroso.
- Quitar todas las etiquetas HTML encontradas. Este método por sí solo puede llegar a ser inseguro y debería complementarse con otros.

5.5.3 SQL injection

La inyección de SQL (SQL injection) es una técnica usada para inyectar código SQL arbitrario en una aplicación y que permite atacar una base de datos usualmente con el fin de obtener acceso no autorizado al sistema o robar o eliminar información directamente de la base de datos.

Una de las formas más efectivas de evitar prevenir la inyección de SQL es usar *prepared statements* o consultas parametrizadas que son fuertemente tipadas y evitan que las consultas SQL sean alteradas mediante el ingreso de los parámetros de una forma independiente que no afecta la estructura de la consulta. De igual forma, se puede realizar la conversión de los caracteres de escape en los valores ingresados por el usuario para evitar que tengan un significado potencialmente peligroso para el motor de base de datos que evalúa la consulta.

5.5.4 Seguridad a partir de la configuración de PHP

La configuración de PHP, también conocida por el archivo donde está contenida, php.ini, contiene diversa funcionalidad que puede ser usada para incrementar la seguridad de las aplicaciones web. Al utilizar algunas de las sus opciones relacionadas con la seguridad es posible reducir considerablemente la cantidad de vulnerabilidades potenciales de las aplicaciones web ejecutándose en un servidor.

Algunas características como el modo seguro de PHP (safe mode), buscaban implementar medidas de seguridad a partir de la verificación de los permisos asignados a cada archivo para asegurar que los scripts fueran ejecutados por entidades autorizadas. Sin embargo, esta característica fue marcada como obsoleta en la versión 5.3 de PHP y eliminada en la versión 5.4. La documentación

oficial de PHP indica que desde el punto de vista de la arquitectura, es incorrecto tratar de solucionar este tipo de problemas a nivel de PHP[13].

Sin embargo, existen otras opciones que pueden ser configuradas de forma que resulten útiles para aumentar la seguridad de la aplicación web. Por ejemplo, se pueden ajustar directivas como *memory_limit* y *max_execution_time* para limitar la capacidad de ejecución de un script malicioso o *session.cookie_httponly* y *session.referer_check* para impedir que scripts maliciosos obtengan información de sesión de los usuarios mediante la exposición de cookies. La lista completa de directivas PHP puede encontrarse en la documentación oficial del lenguaje[14].

5.5.5 Prevención de cross-site request forgery (CSRF)

Un ataque de CSRF ocurre cuando un sitio web malicioso hace que el navegador web de un usuario realice acciones no deseadas en un sitio web que confía en dicho usuario. A diferencia del XSS, que aprovecha la confianza que un usuario tiene por un sitio web particular, el CSRF aprovecha la confianza que un sitio web tiene por un usuario particular.

Para prevenir ataques de CSRF es importante seguir la regla de no utilizar peticiones GET del protocolo HTTP para acciones que modifiquen el estado del servidor, como creación de nuevos datos, modificación o eliminación. En su lugar, deben usarse peticiones POST. Sin embargo, una petición POST por sí sola no previene el CSRF y prácticamente no representa ningún tipo de mejora en la seguridad debido a que los datos transmitidos aún pueden ser interceptados y alterados con el uso de herramientas especializadas. Por este motivo, uno de los

[13] http://php.net/manual/en/features.safe-mode.php

[14] http://www.php.net/manual/en/ini.list.php

métodos que pueden usarse para prevenir ataques de CSRF es incluir valores aleatorios en los datos transmitidos por peticiones POST que puedan ser reconocidos por el servidor para asegurar que las solicitudes de ejecución de acciones provengan efectivamente de un usuario confiable y la información sea devuelta al mismo origen.

5.5.6 Prevención de ataque a cookies

Proteger las cookies[15] de ataques es de vital importancia porque suelen contener información crítica como los códigos identificadores de sesión que potencialmente pueden proporcionar acceso completo a todas las operaciones permitidas por la sesión vulnerada.

Dentro de los métodos que se pueden usar para prevenir el ataque a cookies, se tienen:

- Usar SSL[16] para crear un canal seguro de comunicación y realizar el envío de las cookies de autenticación únicamente a través de una conexión HTTPS[17]. De esta forma, un atacante es incapaz de descifrar el contenido de las cookies transferidas.
- Expirar las cookies de sesión de forma apropiada.

[15] Las cookies son archivos de datos usualmente pequeños enviados por un sitio web y almacenados por el navegador web un usuario que permiten llevar registros de la relación entre el usuario y el sitio web.

[16] SSL (Secure Sockets Layer) o su sucesor, TLS (Transport Layer Security), es un protocolo criptográfico que proporciona comunicaciones seguras en una red, comúnmente Internet.

[17] HTTPS (Hypertext Transfer Protocol Secure) es un protocolo que agrega las capacidades de seguridad de SSL/TLS a las comunicaciones HTTP estándar.

- Prevenir XSS para evitar que código inyectado pueda exponer las cookies de los usuarios.
- Validar el contenido de las cookies y detectar si son alteradas. Esto puede realizarse mediante el establecimiento de una firma criptográfica para las cookies almacenadas, que permitan al servidor implementar métodos de verificación que rechacen cookies alteradas, debido a que las mismas dejan de corresponder con su firma criptográfica original.

6. VERIFICACIÓN

A diferencia de otras etapas del proyecto, la verificación es una actividad que se desarrolló en paralelo durante todo el proceso de creación de la aplicación, enfocándose en realizar control sobre el código fuente desarrollado, buscando asegurar su calidad y permitiendo descubrir a tiempo posibles errores en los productos finales, evitando que se propaguen a las diferentes partes de la aplicación y permitiendo que el mantenimiento del sistema se mantenga en un nivel de complejidad aceptable.

Este proceso de verificación está basado en el uso de formatos especializados denominados Program Checklist (PCL) y Buglist. Ambos formatos tienen su origen en Nippon Computer Kaihatsu, Ltd.y una descripción de sus componentes y funcionalidades es mostrada a continuación.

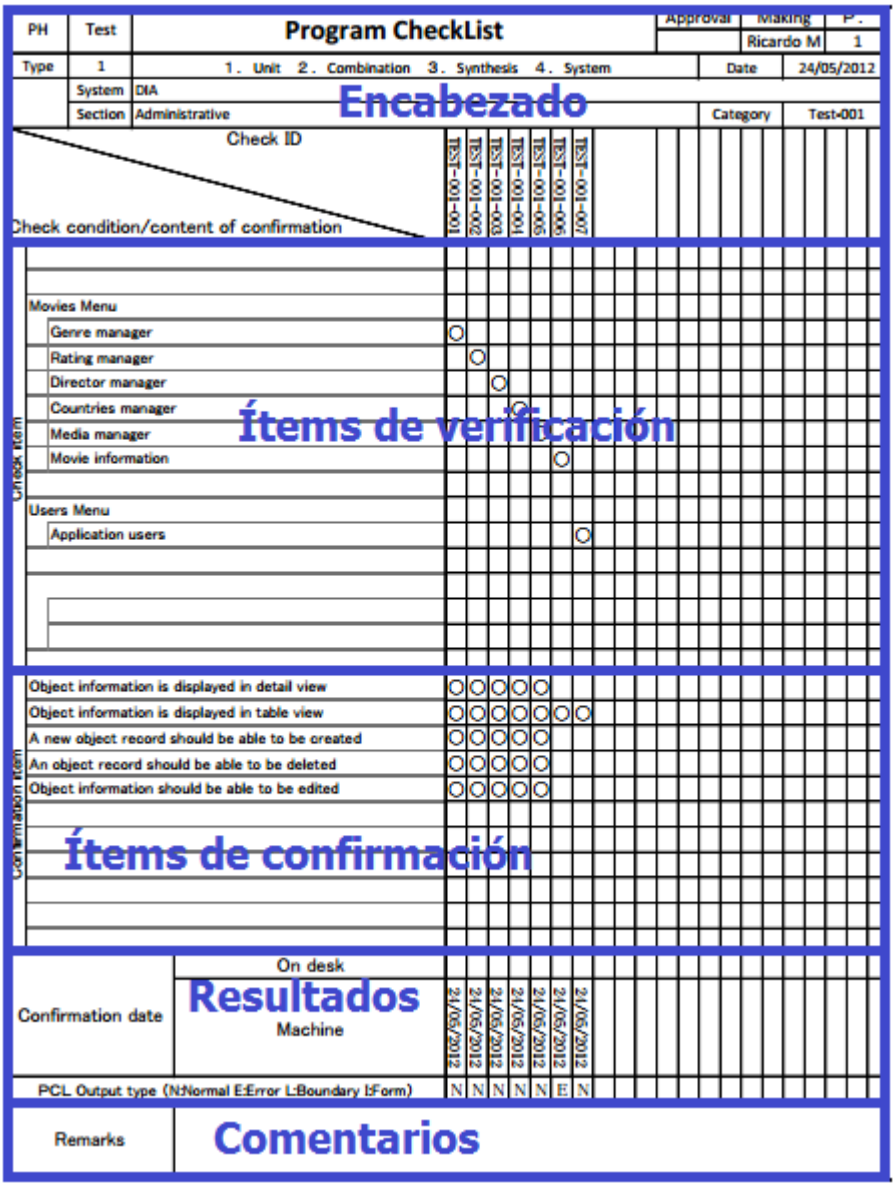
6.1 FORMATO PROGRAM CHECKLIST (PCL)

El formato de verificación PCL busca detectar la mayor cantidad de errores posibles, antes de que los mismos puedan ocasionar defectos derivados en el sistema. Utiliza una perspectiva de "caja negra", donde los errores detectados son registrados pero no se conoce a fondo su naturaleza, debido a que está previsto que esta función la realice el formato Buglist del cual se trata en el siguiente apartado.

Este formato se convierte en una herramienta útil que puede ser aplicada a un sistema con un nivel de granularidad ajustable, desde métodos individuales hasta paquetes o librerías que trabajan de forma anexa al sistema.

Al desglosar el formato PCL, es posible identificar las partes esenciales en que se compone: Encabezado, ítems de verificación, ítems de confirmación, resultados y comentarios.

Figura 34. Composición del formato PCL



Fuente: Nippon Computer Kaihatsu, Ltd.

En el encabezado se especifica información general del sistema objeto de la prueba, el tipo de prueba (de unidad, combinación, síntesis y sistema), el archivo o sección a evaluar e información adicional de control por ejemplo la fecha y el nombre del evaluador.

Los ítems de verificación constituyen una parte fundamental de la verificación. Deben ser definidos de forma tal que sean relevantes para la sección del sistema que está siendo evaluada. Una lista rigurosa con un nivel de granularidad alto permitiría un alto nivel de depuración de la aplicación mediante el uso de este formato, pero también implicaría un incremento en la complejidad de la prueba, situación que puede ser solucionada mediante la descomposición de los objetos complejos en elementos más sencillos de evaluar.

Los ítems de confirmación por su parte, actúan como una lista de resultados esperados proporcionados por los ítems de verificación al exponer el sistema a una serie de entradas para verificar su correcto funcionamiento.

La sección de resultados permite registrar las diferentes salidas proporcionadas por el sistema al final de la prueba. Estas pueden ser clasificadas en N:Normal, E:Error, L:Boundary y I:Form.

N:Normal representa una salida normal de acuerdo a la respuesta esperada por el evaluador, E:Error representa un error común en la aplicación, L:Boundary representa un error de frontera, como por ejemplo, cuando un tipo de dato es insuficiente para contener un valor requerido o la memoria disponible actualmente es insuficiente para la operación a realizar. Finalmente, I:Form representa un error en el cual el formulario usado para la respuesta no es apto para realizar una presentación completa o sin errores de la información requerida.

El espacio destinado a los comentarios se provee como un espacio opcional que el evaluador puede usar para documentar las observaciones que considere adecuadas. Por ejemplo, podría usarse para documentar condiciones especiales que tuvieron que ser tenidas en cuenta para realizar una prueba en particular.

Algunos campos del formato tienen mayor sentido en un entorno de trabajo con muchas personas y máquinas realizando las pruebas, de modo que su uso puede ser opcional en equipos de trabajo pequeños.

La lista completa de documentos de verificación PCL usados en este proyecto se puede encontrar en el anexo C de este documento.

6.2 FORMATO BUGLIST

El formato Buglist actúa como un complemento para el formato PCL, permitiendo documentar los errores detectados con un nivel mayor de detalle. Si bien es importante realizar detección de errores de forma oportuna, esto no es suficiente si no se registran las acciones que fueron necesarias para corregirlos de modo que este control se convierta por sí mismo en una fuente generadora de conocimiento de valor para el desarrollo y que evite que los errores se propaguen contribuyendo de igual forma a agilizar la corrección de errores similares que se presenten en etapas posteriores del desarrollo.

Dentro de los elementos incluidos en la composición del formato Buglist, se encuentra el fenómeno que se observa, su causa y las medidas que se toman para corregirlo. Esto en conjunto describe la naturaleza del error que se registra.

Un segundo grupo de datos contiene información de control referente al archivo donde se realizan correcciones, quién descubrió el error y en qué fecha, así como también identifica a la persona responsable de corregirlo y la fecha en que se realizaron las acciones.

El tercer grupo de datos busca identificar y clasificar el error dentro de una serie de parámetros y categorías predefinidas, con el fin de que la corrección del error sea aplicada en forma ordenada. Esto cobra especial importancia en grupos grandes de trabajo, donde se pueden realizar divisiones especializadas dedicadas a resolver tipos particulares de errores a medida que se presenten.

Figura 35. Control de errores con el formato Buglist

Bug No.	Object	Naturaleza del error			Información de control						Clasificación del error			
		Phenomenon	Cause	Measures	Corrected file	Discover date	Discovered by	Correction date	Correction by	Fix commit ID	Code			
											Phenomenon	Cause	Factor	Process stage
001	Login	Verified login credentials are not being accepted on login.	Real user name was being validated instead of symbolic username.	Validate symbolic username instead of the real user full name.	UserIdentity.php	31/05/2012	Juan N	31/05/2012	Ricardo M	e3adc470adde	11	23	9	C
002	Rental and sales	Attempt to delete a sales or rental record without items, results in error.	No verifications was made before attempting to set inventory availability status when deleting the record.	Check if the record has related items before attempting to change their inventory availability status.	Multiple files. Check commit.	04/06/2012	Ricardo M	04/06/2012	Ricardo M	47ead7ce90af	2	1	9	C
003	Rental and sales	Additional null verification test for sales or rental records was failing to update status availability.	Empty arrays were passing the verification as non-null objects, causing error afterwards.	Check for empty arrays instead of null objects.	Multiple files. Check commit.	04/06/2012	Ricardo M	04/06/2012	Ricardo M	dc073fd16bdc	2	4	9	C
004	Sync operation	Rental records are not being processed on sync operation.	Sales table name was being used on rental sync method.	Use the proper rental table name on rental sync method.	SincronizacionController.php	11/06/2012	Juan N	11/06/2012	Ricardo M	34ed5abf6844	10	22	9	C

Fuente: Nippon Computer Kaihatsu, Ltd.

El grupo de datos de clasificación de errores usa los códigos predefinidos que aparecen listados a continuación.

En primer lugar, se tienen los códigos de los fenómenos observados.

Tabla 5. Códigos de clasificación de fenómenos

Phenomenon code	
1	Time out, infinite loop
2	ABEND (Abnormal termination)
3	Memory (table) corruption
4	File corruption
5	Incorrect calculated value
6	Incorrect list output
7	Incorrect presentation on screen
8	Incorrect message
9	Performance
10	Operability
11	Incorrect expected value
12	Incorrect DB update

Fuente: Nippon Computer Kaihatsu, Ltd. Traducción por los autores.

Por su parte, las causas de los errores son clasificadas de acuerdo al listado que se muestra a continuación.

Tabla 6. Códigos de clasificación de causas de errores

Cause code					
1	Processing omission	11	Defective procedure processing	21	Same item
2	Defective interface	12	Common module implementation error	22	Tag implementation error
3	Defective initial setting	13	Command implementarion error	23	Variable implementation error
4	Incorrect operation processing	14	I/O processing error	24	SQL error
5	Incorrect table processing	15	Operation error	25	Incorrect memory processing
6	Incorrect pointer processing	16	Data environmental error	26	Browser settings error
7	Incorrect counter processing	17	System environment error		
8	Incorrect flag processing	18	Hard abnormality		
9	Incorrect criteria processing	19	Within specification		
10	Incorrect edit processing	20	Reproduction waiting (insufficient data)		

Fuente: Nippon Computer Kaihatsu, Ltd. Traducción por los autores.

El siguiente grupo de códigos de clasificación identifica los factores que contribuyen a que un error ocurra en el desarrollo del proyecto. Esta clasificación funciona como un indicador que permite conocer y tomar acciones sobre los factores que han promovido errores aunque no sean las causas directas de los mismos.

Tabla 7. Códigos de clasificación de factores de error

Factor code	
1	Unclear documentation
2	Defective document
3	Operating system understanding deficiency
4	Business specification understanding deficiency
5	Common module understanding deficiency
6	Operational aspects considerations deficiency
7	Standards understanding deficiency
8	Correction confirmation deficiency
9	Simple mistake
10	Language specification understanding deficiency
11	Execution environment considerations deficiency

Fuente: Nippon Computer Kaihatsu, Ltd. Traducción por los autores.

El último grupo de códigos de clasificación permite ubicar la etapa del proceso de desarrollo en que se generó el error. Esta información resulta valiosa en el momento de identificar las etapas que son más propensas a errores de forma que se puedan tomar medidas para minimizar su ocurrencia en proyectos posteriores.

El equipo de trabajo dedicado a cada una de las etapas del proceso puede ser notificado de la detección de errores permitiéndoles la oportunidad de revisar si la documentación sobre la cual están basando sus procesos de desarrollo sigue siendo válida y realizar las correcciones que sean pertinentes.

Tabla 8. Códigos de clasificación de las etapas del proceso del desarrollo

Process stage code	
UI	User interface design
SS	Detailed design
PS	Program design
C	Coding

Fuente: Nippon Computer Kaihatsu, Ltd. Traducción por los autores.

El documento de control de errores Buglist usado en este proyecto se puede encontrar en el anexo D de este documento.

6.3 INTEGRACIÓN DEL SISTEMA DE CONTROL DE VERSIONES

Una de las grandes ventajas de usar un sistema de control de versiones en un proyecto de desarrollo de software, es la cantidad de registros históricos detallados que pueden llegar a ser usados como parte de la documentación final.

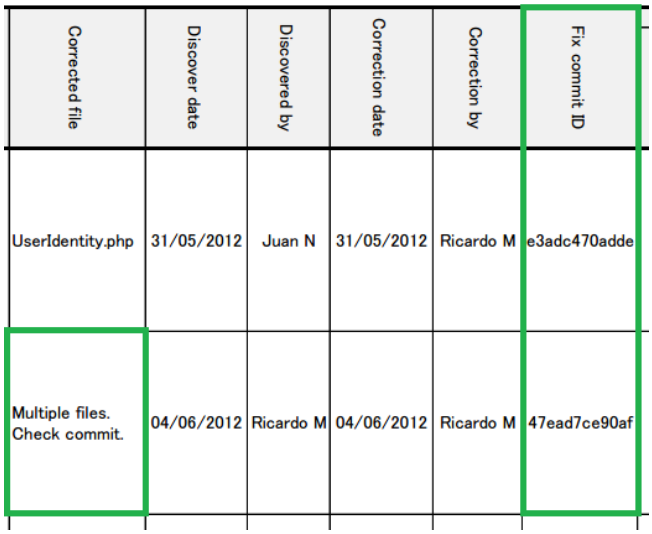
Para este proyecto se tomó la decisión de integrar el sistema de control de versiones en los formatos usados para la etapa de verificación. En particular, el formato Buglist fue modificado para incluir dentro de sus campos de información de control, el código de identificación único de una modificación o *commit* realizada en el código fuente con el fin de corregir un error detectado con el formato de verificación PCL.

Lo anterior, le proporciona un mayor sustento de información verificable al formato usado, permitiendo rastrear las medidas tomadas para corregir un error incluso hasta el nivel de diferencia de archivos de código fuente. De esta forma, se

aumenta de forma considerable el nivel de granularidad al que es posible llegar en el proceso de control y seguimiento de errores y sus correcciones.

Teniendo en cuenta que el flujo de trabajo distribuido inherente a Git, el sistema de control de versiones usado en este proyecto, esta información de seguimiento de las correcciones realizadas está disponible para todos los integrantes del equipo de desarrollo[18].

Figura 36. Ejemplo de integración de Git en el formato Buglist



Corrected file	Discover date	Discovered by	Correction date	Correction by	Fix commit ID
UserIdentity.php	31/05/2012	Juan N	31/05/2012	Ricardo M	e3adc470adde
Multiple files. Check commit.	04/06/2012	Ricardo M	04/06/2012	Ricardo M	47ead7ce90af

Dentro de la información adicional que puede proporcional el sistema de control de versiones al formato Buglist, se encuentra el nombre del responsable de la modificación junto con su email y nombre de usuario si tiene alguno, un código que identifica de forma única a la modificación realizada, un mensaje explicativo realizado por el autor de la modificación, la fecha y hora exacta en que se realizó el cambio y la ubicación exacta de los archivos modificados dentro de la estructura

[18] Mayor información sobre el flujo de trabajo con Git se encuentra en el apartado 5.2 de este documento.

de directorios de la aplicación. De igual forma, también es posible acceder a una visualización de las diferencias de los archivos modificados a nivel de código fuente, incluyendo los números exactos de líneas del archivo.

Gracias a que Git es un proyecto de código abierto, existe una gran variedad en las herramientas disponibles para visualizar los cambios en los archivos. A continuación se muestran algunos ejemplos de cómo se ven los detalles de un error corregido, en la interfaz web del servicio Bitbucket[19].

Figura 37. Corrección de error que afecta múltiples archivos

Este tipo de visualización de los cambios representa una mejora respecto a la forma como se tendrían que agrupar en el formato Buglist. Además, la libertad para seleccionar las herramientas que mejor se adapten al proyecto brinda un grado de flexibilidad que no estaría disponible si sólo se llevara registro en un documento fijo.

[19] Bitbucket es un servicio de alojamiento basado en la web para proyectos que usan los sistemas de control de versiones Git o Mercurial. https://bitbucket.org

Para cada archivo, es posible visualizar los cambios realizados mediante un análisis de diferencias que identifica con códigos de color diferentes los segmentos de código que fueron eliminados y los que fueron agregados.

Figura 38. Análisis de diferencias de archivo en vista única

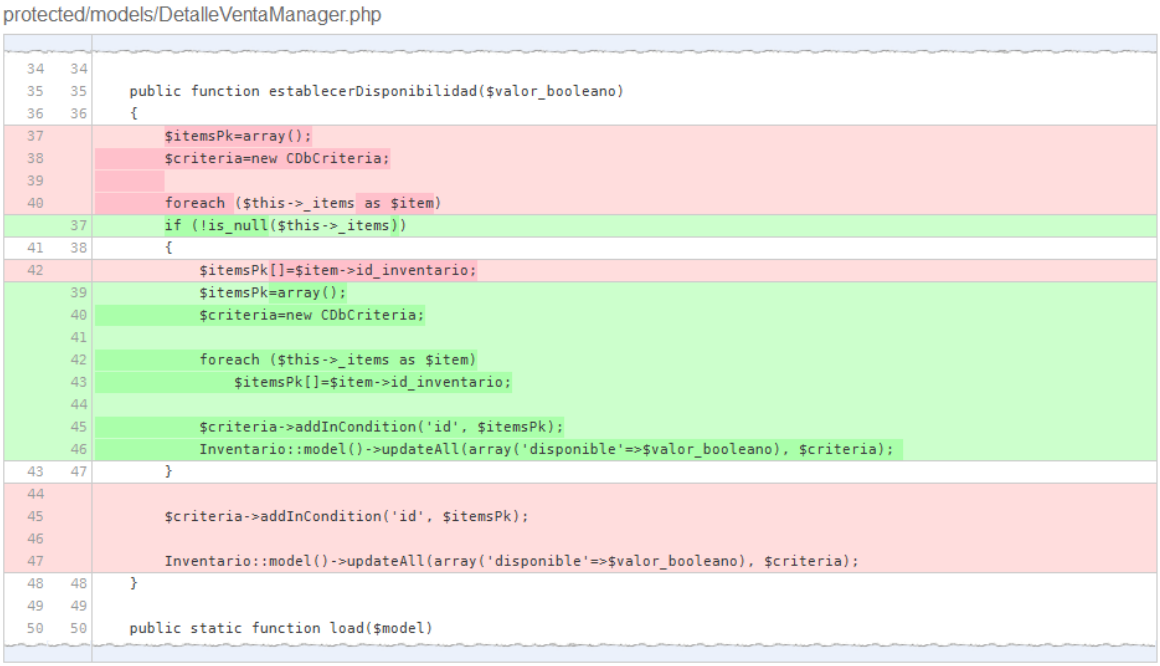
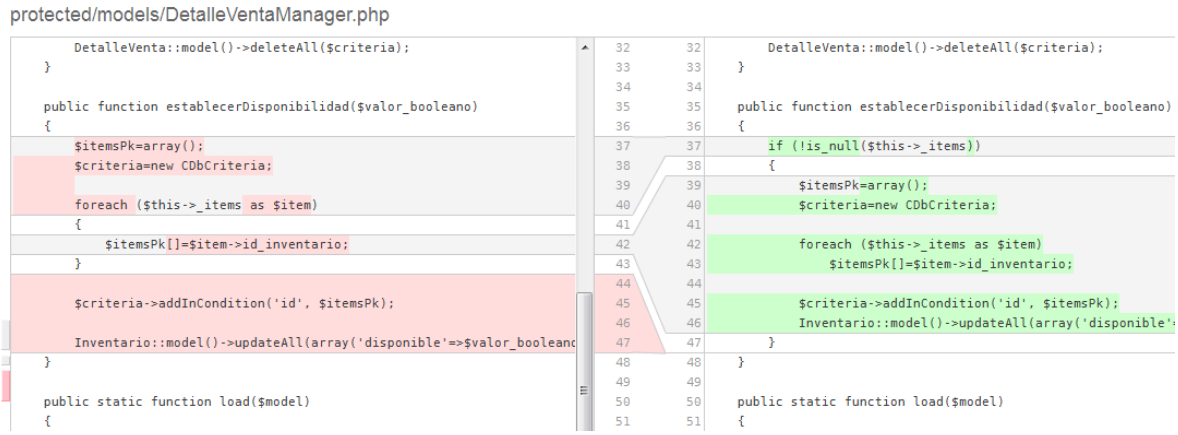


Figura 39. Análisis de diferencias de archivo en vista separada



7. CONCLUSIONES

Al terminar el presente trabajo de investigación se obtuvo como producto final una aplicación distribuida funcional asíncrona basada en la tecnología de Web Services, capaz de operar en plataforma web y a nivel de máquina local y cuyo proceso de verificación estuvo basado en métodos de pruebas que soportan la metodología de NCK. De esta forma, todos los objetivos planteados se cumplieron satisfactoriamente.

La investigación y el desarrollo realizado permite concluir que:

- La implementación de una aplicación distribuida como forma de soportar procesos de negocios de organizaciones tiene como ventaja la flexibilidad en el desarrollo de las operaciones basadas en tecnología sin que dependan de un solo sistema de servicios, lo cual les permite seguir operando aunque hayan interrupciones temporales, por ejemplo, en los servicios que les proveen conectividad a la red. Dependiendo del volumen de operaciones de la organización, la imposibilidad de ofrecer servicios a sus clientes por este tipo de fallas comunes puede representar grandes pérdidas a nivel financiero, de manera que la implementación de aplicaciones distribuidas proporciona una alternativa de solución a este problema.

- El uso de servicios web como base para la comunicación en una aplicación distribuida, permite un mayor grado de independencia de la aplicación respecto a las tecnologías que pueden usarse para extender su funcionalidad. Esto presenta la ventaja de que se puede conseguir una mayor oferta de servicios profesionales para eventuales trabajos de actualización de una aplicación sin necesidad de tener restricciones de una tecnología particular, lo cual tiene

como beneficio adicional el potencial de reducir los costos en el presupuesto del proyecto, gracias a la mayor cantidad de oferta de servicios.

- La replicación y validación de conocimientos producidos en otros contextos como lo es la aplicación de la metodología NCK en este proyecto de investigación, se convierte en un pilar fundamental para la generación de nuevos conocimientos al interior de la universidad. Sin embargo, durante este proyecto no sólo se aplicó esta metodología sino que también se tomó la iniciativa de realizarle mejoras y adaptarla para probar la viabilidad del uso de la misma en soluciones para nuestro entorno local. En particular, la integración de los métodos de verificación basados en la metodología NCK y el sistema de control de versiones, proporcionó un mayor sustento de información verificable para las actividades de pruebas de software junto con una mayor posibilidad de control y seguimiento de los errores y correcciones.

- La incorporación de enfoques ágiles de desarrollo de software cumplió un papel primordial para el cumplimiento oportuno de los objetivos planteados. Se encontró que dichos enfoques no son incompatibles con metodologías más tradicionales cuando se aplican de forma equilibrada. Es decir, manteniendo un balance entre la agilidad que se busca conseguir y los aspectos de calidad requeridos para el proyecto.

- La considerable cantidad de tecnologías de desarrollo usadas permitieron una amplia apropiación de conocimientos en aspectos como el análisis de sistemas orientado a objetos, UML, administración de servidores remotos y bases de datos, sistemas de control de versiones, interoperabilidad en servicios de software distribuidos, entre otros, los cuales constituyen un aporte significativo para la formación profesional de un ingeniero de sistemas en el mundo actual.

8. RECOMENDACIONES

Es importante tener en cuenta que el producto final obtenido en este proyecto no buscó ser una solución definitiva para uso en entornos de producción real. Su propósito fue la replicación, validación y difusión de conocimientos actualizados de técnicas y tecnologías de ingeniería de software de forma que proporcionara las bases para la efectiva aplicación de este conocimiento en el entorno regional. Sin embargo, teniendo como base el estudio realizado junto con las tecnologías que se implementaron durante la realización del presente proyecto, se pueden hacer diferentes recomendaciones a nivel técnico para proyectos futuros que busquen ampliar el escenario de aplicación del sistema desarrollado. A continuación, se presentan algunas de ellas.

- Considerar aumentar la seguridad de la aplicación mediante el uso de técnicas modernas de encriptación para datos críticos como las contraseñas o datos personales de los usuarios.

- Incluir los costos de adquisición de inventario para tener la posibilidad de proporcionar informes financieros de mayor relevancia a la administración de la organización.

- Incorporar registro de elementos que vinculan a los clientes finales con la organización, como por ejemplo, control de deudas, suscripciones y descuentos por membresía.

- Integrar el sistema de autenticación de email con el sistema de autenticación de la aplicación distribuida, de manera que permita una consolidación en el uso de las credenciales que usan los usuarios para autenticarse en todo el sistema.

9. BIBLIOGRAFÍA

AMBLER, Scott. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 2004. ISBN 978-0521540186.

BIRMAN, Kenneth P. Reliable Distributed Systems: Technologies, Web Services, and Applications. Springer, 2010. ISBN 978-1441919502.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. Unified Modeling Language User Guide. Addison-Wesley Professional, 2005. ISBN 978-0321267979.

BRUEGGE, Bernd; DUTOIT, Allen H. Object Oriented Software Engineering. Prentice Hall, 2000. ISBN 978-0136061250.

CHACON, Scott. Pro Git. Apress, 2009. ISBN 978-1430218333.
Publicación en web: http://git-scm.com/book

CERAMI, Ethan. Web Services Essentials. O'Reilly Media, 2002. ISBN 978-0596002244.

FOWLER, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2003. ISBN 978-0321193681.

FOWLER, Martin. Patterns of Enterprise Application Architecture Language. Addison-Wesley Professional, 2002. ISBN 978-0321127426.

HUNT, Andy; SUBRAMANIAM, Venkat. Practices of an Agile Developer. Pragmatic Bookshelf, 2006. ISBN 978-0974514086.

NAVARRO, Leandro. Arquitectura de aplicaciones distribuidas. Ediciones UPC, 2001. ISBN 84-8301-547-1.

STUMPF, Robert; TEAGUE, Lavette. Object-Oriented Systems Analysis and Design With UML. Prentice Hall, 2004. ISBN 978-0131434066.

WEITZENFELD, Alfredo. Software Engineering applied to UML, Java and Internet Object. Thomson International, 2004. ISBN 978-9706861900.

W3C (World Wide Web Consortium). Workshop on Declarative Models of Distributed Web Applications. 2007.
Publicación en web: http://www.w3.org/2007/02/dmdwa-ws/

XUE, Qiang; WEI ZHUO, Xiang. The Definitive Guide to Yii. Yii Software LLC, 2010. Publicación en web: http://www.yiiframework.com/doc/guide/

ANEXOS

ANEXO A. CASOS DE USO DEL SISTEMA

<table>
<tr><td>Caso de uso</td><td colspan="2">Buscar película</td></tr>
<tr><td>Número</td><td colspan="2">1</td></tr>
<tr><td>Prioridad</td><td colspan="2">Media</td></tr>
<tr><td>Objetivo</td><td colspan="2">Proporcionar al visitante la información de la película que busca.</td></tr>
<tr><td>Actores</td><td colspan="2">Visitante</td></tr>
<tr><td>Precondiciones</td><td colspan="2">Deben haber ítems registrados en el catálogo</td></tr>
<tr><td rowspan="2">Flujo de eventos</td><td>Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td>1. El visitante ingresa el nombre de la película (ítem) que busca.
3. El visitante selecciona uno de los ítems devueltos por el sistema.</td><td>2. El sistema presenta la lista de ítems que coinciden con el criterio de búsqueda.
4. El sistema muestra información detallada del ítem seleccionado por el visitante.</td></tr>
<tr><td>Variaciones</td><td colspan="2">No hay ítems registrados en el catálogo. Mostrar un mensaje informativo al visitante. Termina el caso de uso.</td></tr>
</table>

Caso de uso	**Enviar mensaje**	
Número	2	
Prioridad	Baja	
Objetivo	Brindar los medios al visitante para que pueda enviar un mensaje al administrador.	
Actores	Visitante	
Precondiciones	Ninguna	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El visitante ingresa a la sección de "Contacto". 3. El visitante ingresa el mensaje junto con información opcional que lo identifique, como su nombre, email y teléfono.	2. El sistema presenta un formulario donde el visitante pueda ingresar la información pertinente al mensaje. 4. El sistema realiza el envío del mensaje a través de email.
Variaciones	El visitante no ingresa ningún mensaje e intenta enviarlo. Mostrar un mensaje informativo al visitante y regresar al paso 3.	

Caso de uso	**Administrar películas**	
Número	3	
Prioridad	Alta	
Objetivo	Ver, agregar, modificar y eliminar información relacionada a una película.	
Actores	Administrador	
Precondiciones	El administrador debe iniciar sesión	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El administrador ingresa al módulo de administrar películas. 3. Selecciona una de las opciones. 5. Agrega, modifica o elimina la información necesaria.	2. El sistema presenta las opciones de ver, agregar, modificar y eliminar películas. 4. Presenta la información correspondiente, de acuerdo a la opción seleccionada. 6. Actualiza la información en la base de datos.
Variaciones	Se ingresa información no válida en alguno de los campos de información a guardar. Mostrar un mensaje de error y regresar al paso 5.	

Caso de uso	**Administrar usuarios**	
Número	4	
Prioridad	Alta	
Objetivo	Ver, agregar, modificar y eliminar información relacionada a los usuarios de la aplicación.	
Actores	Administrador	
Precondiciones	El administrador debe iniciar sesión	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El administrador ingresa al módulo de administrar usuarios. 3. Selecciona una de las opciones. 5. Agrega, modifica o elimina la información necesaria.	2. El sistema presenta las opciones de ver, agregar, modificar y eliminar productos. 4. Presenta la información correspondiente, de acuerdo a la opción seleccionada. 6. Actualiza la información en la base de datos.
Variaciones	Se ingresa información no válida en alguno de los campos de información a guardar. Mostrar un mensaje de error y regresar al paso 5.	

Caso de uso	Generar informe de alquileres	
Número	5	
Prioridad	Media	
Objetivo	Proporcionar al administrador un reporte detallado de los alquileres de películas registrados en un tiempo determinado.	
Actores	Administrador	
Precondiciones	El administrador debe iniciar sesión	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El administrador ingresa al módulo de informes de alquileres de películas. 3. Ingresa el rango de fechas deseado e inicia el proceso de generación del reporte.	2. El sistema presenta un formulario donde se puede especificar un rango de fechas para la generación del informe. 4. Muestra el reporte generado.
Variaciones	No hay registros existentes para el rango de fecha seleccionado por el administrador. Mostrar un mensaje informativo y regresar al paso 2.	

Caso de uso	**Generar informe de ventas**	
Número	6	
Prioridad	Media	
Objetivo	Proporcionar al administrador un reporte detallado de las ventas de películas registradas en un tiempo determinado.	
Actores	Administrador	
Precondiciones	El administrador debe iniciar sesión	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El administrador ingresa al módulo de informes de ventas de películas. 3. Ingresa el rango de fechas deseado e inicia el proceso de generación del reporte.	2. El sistema presenta un formulario donde se puede especificar un rango de fechas para la generación del informe. 4. Muestra el reporte generado.
Variaciones	No hay registros existentes para el rango de fecha seleccionado por el administrador. Mostrar un mensaje informativo y regresar al paso 2.	

Caso de uso	**Sincronizar módulos**	
Número	7	
Prioridad	Alta	
Objetivo	Realizar el proceso de actualización de los registros de los diferentes elementos del sistema desde la aplicación cliente hacia el servidor web y también en dirección contraria mediante el uso de servicios web.	
Actores	Operador	
Precondiciones	La aplicación cliente y el servidor web deben contar con una conexión activa a Internet.	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El operador indica al sistema que se debe iniciar el proceso de sincronización del sistema.	2. El sistema realiza la actividad especificada y muestra un mensaje informativo con los resultados de la operación, habiendo actualizado la base de datos.
Variaciones	No hay una conexión activa a Internet. Mostrar un mensaje informativo y regresar al paso 1.	

Caso de uso	**Administrar clientes**	
Número	8	
Prioridad	Alta	
Objetivo	Ver, agregar, modificar y eliminar información relacionada a un cliente.	
Actores	Operador	
Precondiciones	Ninguna	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El operador ingresa al módulo de administrar clientes. 3. Selecciona una de las opciones. 5. Agrega, modifica o elimina la información necesaria.	2. El sistema presenta las opciones de ver, agregar, modificar y eliminar clientes. 4. Presenta la información correspondiente, de acuerdo a la opción seleccionada. 6. Actualiza la información en la base de datos.
Variaciones	Se ingresa información no válida en alguno de los campos de información a guardar. Mostrar un mensaje de error y regresar al paso 5.	

Caso de uso	**Administrar alquileres**	
Número	9	
Prioridad	Alta	
Objetivo	Registrar el alquiler de una película.	
Actores	Operador	
Precondiciones	Debe haber al menos un ítem disponible en el inventario.	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El operador ingresa al módulo de registrar alquiler de película. 3. Ingresa los detalles del alquiler de la película e indica al sistema que debe realizar el registro.	2. El sistema proporciona un formulario para ingresar los detalles del alquiler de la película. 4. Realiza el registro del alquiler actualizando la base de datos de acuerdo a la información ingresada.
Variaciones	No hay inventario suficiente del ítem a registrar en alquiler. Mostrar un mensaje informativo y regresar al paso 2.	

Caso de uso	**Administrar ventas**	
Número	10	
Prioridad	Alta	
Objetivo	Registrar la venta de una película	
Actores	Operador	
Precondiciones	Debe haber al menos un ítem disponible en el inventario.	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El operador ingresa al módulo de registrar venta de película. 3. Ingresa los detalles de la venta del producto e indica al sistema que debe realizar el registro.	2. El sistema proporciona un formulario para ingresar los detalles de la venta de la película. 4. Realiza el registro de la venta actualizando la base de datos de acuerdo a la información ingresada.
Variaciones	No hay inventario suficiente del ítem a registrar en venta. Mostrar un mensaje informativo y regresar al paso 2.	

Caso de uso	**Actualizar inventario**	
Número	11	
Prioridad	Alta	
Objetivo	Realizar el proceso de actualización de los registros del catálogo de películas desde el servidor web hacia la aplicación cliente.	
Actores	Operador	
Precondiciones	La aplicación cliente y el servidor web deben contar con una conexión activa a Internet.	
Flujo de eventos	**Acción del actor**	**Respuesta del sistema**
	1. El operador indica al sistema que se debe iniciar el proceso de actualizar el catálogo de películas.	2. El sistema realiza la actividad especificada y muestra un mensaje informativo con los resultados de la operación, habiendo actualizado la base de datos.
Variaciones	No hay una conexión activa a Internet. Mostrar un mensaje informativo y regresar al paso 1.	

ANEXO B. DESCRIPCIÓN DE LA BASE DE DATOS DEL SISTEMA

1. CONJUNTO DE TABLAS DE LA BASE DE DATOS

Las siguientes tablas se definieron durante el diseño de la base de datos.

Tabla: genero
Descripción: Contiene información de los géneros por los cuales son clasificadas las películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombre del género.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: clasificación
Descripción: Contiene información de las clasificaciones que recomiendan restricciones de edad para los espectadores de películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombre de la clasificación.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: director
Descripción: Contiene la información de directores de películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombre completo del director.

- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: pais
Descripción: Contiene la información de países de origen de películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombre del país.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: pelicula
Descripción: Contiene la información de las películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(100))*: Nombre de la película.
- *titulo_original (varchar(100))*: Título original de la película.
- *id_genero (integer)*: Identificador del género.
- *id_clasificacion (integer)*: Identificador de la clasificación.
- *sinopsis (varchar(1000))*: Sinopsis de la película.
- *id_director (integer)*: Identificador del director.
- *anio (year)*: Año de lanzamiento.
- *id_pais (integer)*: Identificador del país de origen.
- *url_trailer (varchar(500))*: URL del *trailer* de la película.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: inventario

Descripción: Contiene información del inventario de películas. Estos registros representan existencias físicas de las películas.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *codigo (varchar(50))*: Código único de inventario.
- *id_pelicula (integer)*: Identificador de la película.
- *id_medio (integer)*: Identificador del medio de almacenamiento.
- *disponible (boolean)*: Indicada si el ítem está disponible o no.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: medio

Descripción: Contiene la información de medios de almacenamiento de los ítems de inventario.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombre del medio de almacenamiento.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: cliente

Descripción: Contiene información de los clientes.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(50))*: Nombres del cliente.
- *apellidos (varchar(50))*: Apellidos del cliente.
- *nombre_completo (varchar(105))*: Concatenación del nombre y apellidos del cliente.

- *telefono_fijo (varchar(50))*: Número de teléfono fijo del cliente.
- *telefono_movil (varchar(50))*: Número de teléfono móvil del cliente.
- *direccion (varchar(200))*: Dirección de residencia o contacto del cliente.
- *email (varchar(100))*: Dirección de correo electrónico del cliente.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: alquiler
Descripción: Almacena el encabezado de los registros de alquileres de películas.
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *numero_factura (varchar(50))*: Número de la factura asociada al alquiler.
- *id_cliente (integer)*: Identificador del cliente.
- *id_usuario (integer)*: Identificador del usuario que registra el alquiler.
- *fecha (datetime)*: Fecha y hora del alquiler.
- *fecha_devolucion_programada (date)*: Fecha programada para la devolución del alquiler por parte del cliente.
- *devuelto (boolean)*: Indica si ya se realizó la devolución o no.
- *fecha_devolucion_real (datetime)*: Fecha y hora real de la devolución del alquiler por parte del cliente.
- *id_usuario_recibido (integer)*: Identificador del usuario que recibe el alquiler devuelto.
- *valor_total (currency)*: Valor total pagado por el cliente.
- *notas (varchar(1000))*: Notas opcionales respecto al alquiler.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: detalle_alquiler
Descripción: Almacena el detalle de los registros de alquileres de películas.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *id_alquiler (integer)*: Identificador del alquiler.
- *id_inventario (integer)*: Identificador del ítem de inventario.
- *valor (currency)*: Valor pagado por el cliente específicamente por el ítem referenciado.

Tabla: venta

Descripción: Almacena el encabezado de los registros de ventas de películas.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *numero_factura (varchar(50))*: Número de la factura asociada a la venta.
- *id_cliente (integer)*: Identificador del cliente.
- *id_usuario (integer)*: Identificador del usuario que registra la venta.
- *fecha (datetime)*: Fecha y hora de la venta.
- *valor_total (currency)*: Valor total pagado por el cliente.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: detalle_venta

Descripción: Almacena el detalle de los registros de ventas de películas.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *id_venta (integer)*: Identificador de la venta.
- *id_inventario (integer)*: Identificador del ítem de inventario.
- *valor (currency)*: Valor pagado por el cliente específicamente por el ítem referenciado.

Tabla: usuario

Descripción: Contiene la información de usuarios de la aplicación.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *id_perfil (integer)*: Identificador del perfil de usuario.
- *nombre_usuario (varchar(20))*: Nombre de usuario.
- *contrasena (varchar(50))*: Contraseña del usuario.
- *nombre (varchar(50))*: Nombre real del usuario.
- *apellidos (varchar(50))*: Apellidos del usuario.
- *nombre_completo (varchar(105))*: Concatenación del nombre real y apellidos del usuario.
- *email (varchar(100))*: Dirección de correo electrónico del cliente.
- *fecha_modificacion (timestamp)*: Fecha y hora de la última modificación del registro.

Tabla: perfil

Descripción: Contiene la información de los perfiles de usuarios de la aplicación.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *nombre (varchar(20))*: Nombre del perfil de usuario.

Tabla: sincronizacion

Descripción: Lleva registro de las sincronizaciones realizadas por la aplicación local.

Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *id_usuario (integer)*: Identificador del usuario que inicia la sincronización.
- *fecha (timestamp)*: Fecha y hora de la sincronización.

Tabla: papelera

Descripción: Lleva registro de los elementos eliminados por los usuarios de la aplicación, para soportar la replicación de objetos eliminados en los procesos de sincronización

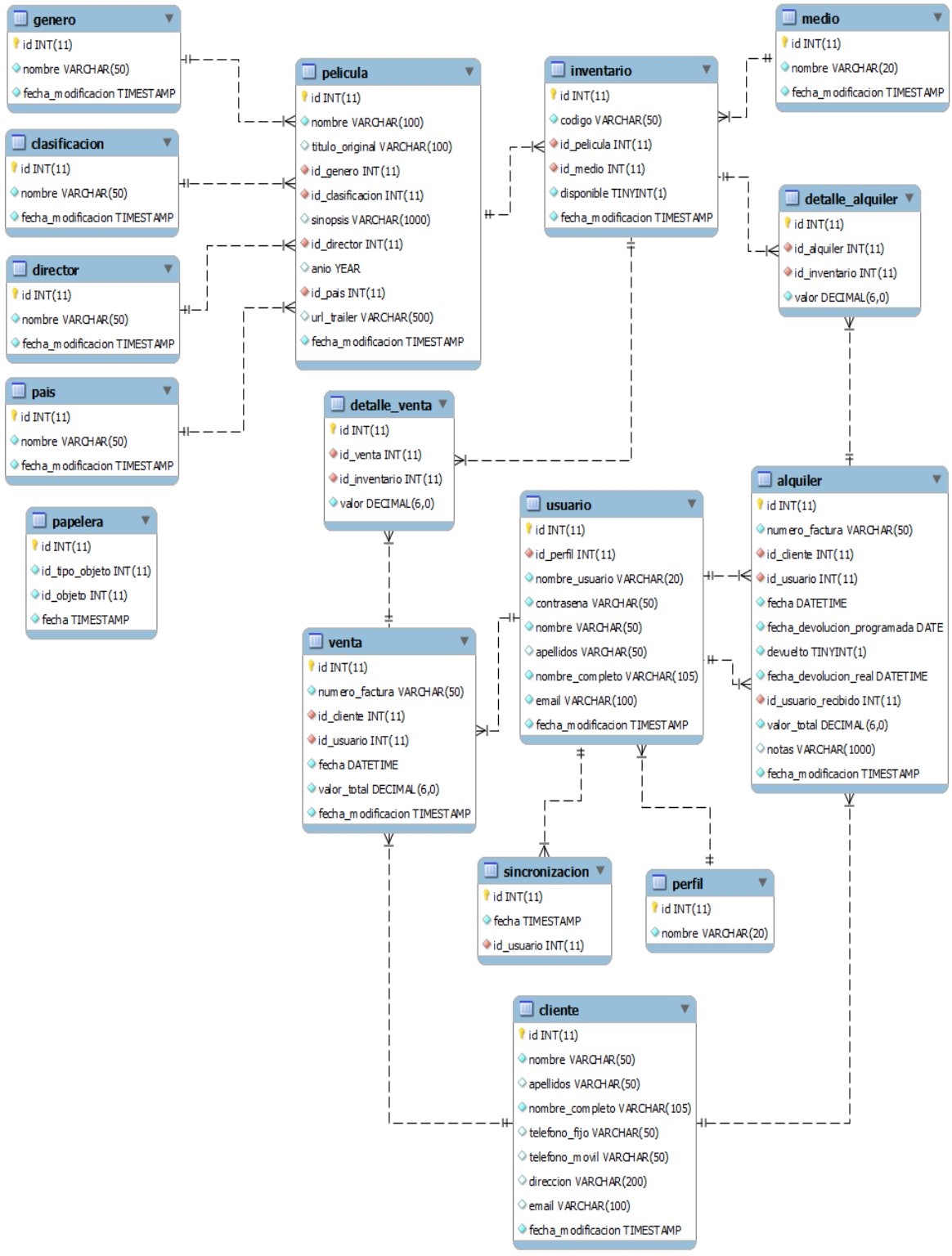
Campos:

- *id (integer)*: Identificador único de los registros de la tabla.
- *id_tipo_objeto (integer)*: Identificador del tipo de objeto eliminado. Cada tipo de objeto representa una tabla de la base de datos que debe ser sincronizada. Los números asignados a cada tipo de objeto están definidos en el código fuente de la aplicación para evitar modificaciones que ocasionarían sincronizaciones incorrectas.
- *id_objeto (integer)*: Identificador del objeto eliminado. Cada objeto representa una fila específica de una tabla de la base de datos.
- *fecha (timestamp)*: Fecha y hora de la eliminación.

2. DIAGRAMA RELACIONAL DE LA BASE DE DATOS

En la etapa del diseño de la aplicación se buscó asegurar la integridad referencial mediante la definición de una base de datos relacional, descrita mediante el diagrama que se muestra en la página siguiente por motivos de presentación del documento.

Figura 1. Diagrama de la base de datos del sistema.



ANEXO C. FORMATO PCL

PH	Test	Program CheckList	Approval	Making	P.
				Ricardo M	1
Type	1	1. Unit 2. Combination 3. Synthesis 4. System	Date	24/05/2012	
	System	DIA			
	Section	Administrative	Category	Test-001	

Check item / Check condition/content of confirmation	TEST-001-001	TEST-001-002	TEST-001-003	TEST-001-004	TEST-001-005	TEST-001-006	TEST-001-007	TEST-001-008	TEST-001-009	TEST-001-010	TEST-001-011	TEST-001-012	TEST-001-013	TEST-001-014
Movies Menu														
Genre manager	○							○						
Rating manager		○							○					
Director manager			○							○				
Countries manager				○							○			
Media manager					○							○		
Movie information						○							○	
Users Menu														
Application users							○							○
Confirmation item														
Object information is displayed in detail view	○	○	○	○	○			○	○	○	○	○		
Object information is displayed in table view	○	○	○	○	○	○	○	○	○	○	○	○	○	○
A new object record should be able to be created	○	○	○	○	○			○	○	○	○	○		
An object record should be able to be deleted	○	○	○	○	○			○	○	○	○	○		
Object information should be able to be edited	○	○	○	○	○			○	○	○	○	○		
Confirmation date / On desk														
Confirmation date / Machine	24/05/2012	24/05/2012	24/05/2012	24/05/2012	24/05/2012	24/05/2012	24/05/2012	05/07/2012	05/07/2012	05/07/2012	05/07/2012	05/07/2012	05/07/2012	05/07/2012
PCL Output type (N:Normal E:Error L:Boundary I:Form)	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Remarks														

Program CheckList

PH	Test		Approval	Making	P.
				Juan N	2
Type	1	1. Unit 2. Combination 3. Synthesis 4. System	Date	04/06/2012	
	System	DIA			
	Section	Operative	Category	Test-002	

Check condition/content of confirmation \ Check ID		TEST-002-001	TEST-002-002	TEST-002-003	TEST-002-004	TEST-002-005	TEST-002-006	TEST-002-007	TEST-002-008	TEST-002-009	TEST-002-010	TEST-002-011	TEST-002-012	TEST-002-013	TEST-002-014	TEST-002-015
Check item																
	Operations Menu															
	Client manager	○					○					○				
	Inventory manager		○					○					○			
	Rental manager			○					○					○		
	Sales manager				○					○					○	
	Sync manager					○					○					○
Confirmation item	Object information is displayed in detail view	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Object information is displayed in table view	○	○	○	○		○	○	○	○		○	○	○	○	
	A new object record should be able to be created	○	○	○	○		○	○	○	○		○	○	○	○	
	An object record should be able to be deleted	○	○	○	○		○	○	○	○		○	○	○	○	
	Object information should be able to be edited	○	○	○	○		○	○	○	○		○	○	○	○	
	Sync process must be performed					○					○					○
	Sync process must show results after finishing					○					○					○
Confirmation date	On desk															
	Machine	04/06/2012	04/06/2012	04/06/2012	04/06/2012	11/06/2012	29/07/2012	29/07/2012	29/07/2012	29/07/2012	29/07/2012	31/07/2012	31/07/2012	31/07/2012	31/07/2012	31/07/2012
PCL Output type (N:Normal E:Error L:Boundary I:Form)		N	N	E	E	E	N	N	N	N	E	N	N	N	N	N
Remarks																

PH	Test	Program CheckList	Approval	Making	P.
				Juan N	3
Type	1	1. Unit 2. Combination 3. Synthesis 4. System	Date	14/06/2012	
	System	DIA			
	Section	Reports	Category	Test-003	

	Check ID	TEST-003-001	TEST-003-002	TEST-003-003	TEST-003-004	TEST-003-005	TEST-003-006
Check item	Check condition/content of confirmation						
	Reports Menu						
	Rental report	○		○		○	
	Sales report		○		○		○
Confirmation item	Report for today's date should be displayed by default	○	○	○	○	○	○
	Report updates its information on date range change	○	○	○	○	○	○
	Total value must be displayed at the bottom of the table	○	○	○	○	○	○
Confirmation date	On desk						
	Machine	14/06/2012	14/06/2012	21/06/2012	21/06/2012	28/06/2012	28/06/2012
PCL Output type (N:Normal E:Error L:Boundary I:Form)		N	N	N	N	N	N
Remarks							

PH	Test	**Program CheckList**	Approval	Making	P.
				Ricardo M	4
Type	1	1. Unit 2. Combination 3. Synthesis 4. System	Date	31/05/2012	
	System	DIA			
	Section	Public modules	Category	Test-004	

	Check ID	TEST-004-001	TEST-004-002	TEST-004-003	TEST-004-004	TEST-004-005	TEST-004-006	TEST-004-007	TEST-004-008	TEST-004-009	TEST-004-010	TEST-004-011	TEST-004-012	TEST-004-013	TEST-004-014
	Check condition/content of confirmation														
Check item															
	Public pages														
	Home page	○							○						
	Error pages		○							○					
	Contact page			○							○				
	Movie search														
	Search and results				○							○			
	Video frame integration					○							○		
	Login page														
	Login as authenticated user						○							○	
	Logout to guest user							○							○
Confirmation item	Introductory information must be displayed on home page	○							○						
	Error pages must not include sensitive information such		○							○					
	as source code or database details														
	Contact form must send an email on submit event			○							○				
	Results must be shown after search is performed				○							○			
	Availability must be specified for every movie on search				○							○			
	A video frame must be embedded on search result					○							○		
	A valid user must be accepted to login						○							○	
	User credentials must be removed on logout							○							○
Confirmation date	On desk														
	Machine	31/05/2012	31/05/2012	31/05/2012	31/05/2012	31/05/2012	31/05/2012	31/05/2012	28/06/2012	28/06/2012	28/06/2012	28/06/2012	28/06/2012	28/06/2012	28/06/2012
PCL Output type (N:Normal E:Error L:Boundary I:Form)		N	N	N	N	N	E	N	N	N	N	N	N	N	N
Remarks															

ANEXO D. FORMATO BUGLIST

Bug No.	Object	Phenomenon	Cause	Measures	Corrected file	Discover date	Discovered by	Correction date	Correction by	Fix commit ID	Code			
											Phenomenon	Cause	Factor	Process stage
001	Login	Verified login credentials are not being accepted on login.	Real user name was being validated instead of symbolic username.	Validate symbolic username instead of the real user full name.	UserIdentity.php	31/05/2012	Ricardo M	31/05/2012	Ricardo M	e3adc470adde	11	23	9	C
002	Rental and sales	Attempt to delete a sales or rental record without items, results in error.	No verifications was made before attempting to set inventory availability status when deleting the record.	Check if the record has related items before attempting to change their inventory availability status.	Multiple files. Check commit.	04/06/2012	Juan N	04/06/2012	Ricardo M	47ead7ce90af	2	1	9	C
003	Rental and sales	Additional null verification test for sales or rental records was failing to update status availability.	Empty arrays were passing the verification as non-null objects, causing error afterwards.	Check for empty arrays instead of null objects.	Multiple files. Check commit.	04/06/2012	Juan N	04/06/2012	Ricardo M	dc073fd16bdc	2	4	9	C
004	Sync operation	Rental records are not being processed on sync operation.	Sales table name was being used on rental sync method.	Use the proper rental table name on rental sync method.	SincronizacionController.php	11/06/2012	Juan N	11/06/2012	Ricardo M	34ed5abf6844	10	22	9	C
005	Sync operation	ID of new records are not processed on sync operation. Massive assigment of object attributes doesn't update their values on the DB.	Unsafe object attributes such as ID are not updated on massive assigment operations.	Create a scenario to allow updating unsafe object attributes on massive assigment operations.	Multiple files. Check commit.	29/07/2012	Juan N	30/07/2012	Ricardo M	54e184f61951	12	1	5	C

Texto original en japonés de los códigos de clasificación del formato Buglist

原因コード					
1	処理抜け	11	処理順序不良	21	同件
2	インターフェイス不良	12	共通モジュール使用誤り	22	タグ使用誤り
3	初期設定不良	13	命令使用誤り	23	変数使用誤り
4	演算処理不正	14	入出力処理誤り	24	ＳＱＬ誤り
5	テーブル処理不正	15	操作誤り	25	メモリ処理不正
6	ポインタ処理不正	16	データ環境誤り	26	ブラウザ設定誤り
7	カウンタ処理不正	17	システム環境誤り		
8	フラグ処理不正	18	ハード異常		
9	判定処理不正	19	仕様通り		
10	編集処理不正	20	再現待ち（資料不足）		

要因コード	
1	仕様不明確
2	ドキュメント不良
3	基本ソフト理解不足
4	業務仕様理解不足
5	共通モジュール理解不足
6	運用面考慮不足
7	規格・基準理解不足
8	修正確認不足
9	単純誤り
10	言語仕様理解不足
11	実行環境考慮不足

現象コード	
1	ウエイト、ループ
2	ＡＢＥＮＤ（異常終了）
3	メモリ（テーブル）破壊
4	ファイル破壊
5	計算値不正
6	リスト出力不正
7	画面表示不正
8	メッセージ不正
9	性能
10	操作性
11	期待値不正
12	ＤＢ更新不正

不良作り込み工程	
UI	ユーザＩ／Ｆ設計
SS	詳細設計
PS	プログラム設計
C	コーディング