Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Arquitectura de software para aplicaciones Web

Tesis que presenta

Juan Tahuiton Mora

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Pedro Mejía Álvarez

México D.F. **Agosto 2011**

Agradecimientos

- A Dios por ser el arquitecto de mi vida, por sustentarme en cada paso que he dado a lo largo de mi vida.
- A mi esposa Alma por su ayuda, comprensión y consuelo en los momentos importantes de mi vida, por enseñarme a soñar y a conquistar los sueños por difíciles que parezcan.
- A mis padres, Juan Tahuiton y Leonor Mora por su gran amor y apoyo durante toda mi vida. A mis hermanos Fide, Juanis, Toño y Leti por sus consejos.
- A mi director de tesis, Dr. Pedro Mejía Álvarez por su ayuda y paciencia durante la realización de este trabajo de tesis.
- A mis revisores de tesis, Dr. Oscar Olmedo Aguirre y Dra. Ana María por sus invaluables contribuciones a este trabajo de tesis.
- A todo el personal del Departamento de Computación, en especial a Sofía Reza, Felipa Rosas y Erika Ríos por su gran apoyo y compromiso con su trabajo.
- A todos mis amigos del CINVESTAV, con quienes pase momentos difíciles pero también agradables.
- Al Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV-IPN) por haberme brindado la oportunidad de ser formar parte de una grandiosa institución.

- Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por otorgarme el apoyo económico para realizar mis estudios de posgrado.

Resumen

La arquitectura de software es una pieza central del desarrollo de sistemas de software modernos. El objetivo de la arquitectura consiste en desarrollar sistemas de software grandes de forma eficiente, estructurada y con capacidad de reuso. La arquitectura forma parte del proceso de diseño de software el cual también forma parte del proceso de desarrollo de software que comprende, requerimientos, diseño, implementación, prueba y mantenimiento.

La investigación en esta área es muy reciente y actualmente existen muy pocos modelos que permiten diseñar arquitecturas de aplicaciones de software. Debido a esto, en la presente tesis tenemos como objetivo el desarrollo de un marco genérico para definir la arquitectura de un sistema de software basado en Web.

Los sistemas de software basados en Web han tenido un gran auge en la última década. Sus principales aplicaciones, los sistemas de comercio electrónico y las redes sociales han visto un crecimiento notable debido también a la mejora de las tecnologías de Internet, de cómputo distribuido, de los lenguajes basados en objetos y las arquitecturas de hardware.

En el presente trabajo de tesis se presenta una arquitectura de software para aplicaciones Web en donde se sigue un proceso de ingeniería de software. En este desarrollo, la arquitectura se descompone mediante distintas vistas o enfoques tales como, la vista lógica, la vista de procesos, la vista de desarrollo, la vista física y la vista de seguridad. Cada vista, en esta tesis, se desarrolla mediante el lenguaje de modelado unificado UML.

Además de presentar la arquitectura genérica de un sistema en Web, en esta tesis presentamos un caso de estudio en el cual se utiliza la arquitectura genérica desarrollada para modelar la aplicación.

Abstract

Software architecture is a central part of the development of modern software systems. The purpose of software architecture is to develop large software systems in an efficient and structured fashion capable reuse. Software architecture is a part of the software design process which is included as in the software development process. This process includes requirements, design, implementation, testing and maintenance.

Research in this area is very recent and currently there are very few models to design software applications architectures. Because of this, in this thesis we aim to develop a generic framework for defining the architecture of Web-based software.

Web-based software systems have boomed in the last decade. Their main applications, e-commerce systems and social networks, have seen significant growth due also to the improvement of Internet technologies, distributed computing, object-based languages and hardware architectures.

In this thesis we introduce a software architecture for Web applications according to a well defined software engineering process. In our development the architecture is broken down by different views or approaches such as the logical view, the process view, the development view, the physical view and the security view. Each view, this thesis is developed using the unified modeling language UML.

Besides presenting the general architecture of a Web system, in this thesis we present a case study in which the generic architecture is used to model the application developed.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Debido al gran éxito que ha tenido la WWW (World Wide Web) el desarrollo de aplicaciones Web ha crecido de forma notable abarcando áreas como comercio electrónico, redes sociales, banca en línea, entretenimiento, etc. La mayoría de los grandes sistemas información han tenido que ser trasladados a ambientes Web como parte de su evolución. Los sistemas de comercio que antes atendían a pequeñas localidades ahora están encargados de atender a todo el mundo.

Debido a que Internet es un mercado muy demandante nos encontramos con la necesidad de construir aplicaciones Web más complejas y en un tiempo muy reducido. Además de que dichas aplicaciones necesitan cumplir con requisitos de calidad como son rendimiento, usabilidad, escalabilidad, mantenimiento, accesibilidad, etc. Esto conduce a que el desarrollo de aplicaciones Web tenga una probabilidad muy alta de fracasar.

Autores como [1] [2] han propuesto metodologías que pretenden mejorar el desarrollo de aplicaciones como lo exige la Web, sin embargo estas sólo se han preocupado de los aspectos funcionales, de la navegación y representación de la aplicación pero han omitido los aspectos de calidad o no funcionales y aspectos que permitan describir formalmente la arquitectura de dichas aplicaciones[3].

Dichas carencias en las metodologías actuales, han motivado el desarrollo de esta tesis en la cual se propone una arquitectura de software para aplicaciones Web

utilizando técnicas de patrones arquitectónicos, patrones de diseño y lenguajes de modelado.

1.1. Motivación y Justificación

La motivación principal de este trabajo de tesis consiste en modelar una arquitectura de software para aplicaciones basadas es Web. En la actualidad no existen modelos de arquitectura de software que permitan representar la estructura y todas las características de distintas aplicaciones de software, en particular aplicaciones basadas en Web. Así mismo, los métodos actuales de modelado y de análisis de software no han madurado lo suficiente como para ser capaces de representar los requerimientos de software y que de este modelado sea posible construir el software de forma confiable.

El estudio de la arquitectura de software se centra en la forma en cómo son diseñados y construidos los sistemas de software. La arquitectura de un sistema de software define la estructura del sistema y está integrada de todos los componentes que involucran al sistema así como sus interconexiones.

Disciplinas como la electrónica utilizan técnicas de diseño que permiten construir placas electrónicas para computadoras de forma ordenada y siguiendo un estándar. Los ingenieros en electrónica construyen computadoras siguiendo una arquitectura previamente definida en la que la mayoría de los componentes ya se encuentran construidos. En esta arquitectura los componentes son el microprocesador, la memoria, los contadores, el controlador de interrupciones, el bus, los drivers y los dispositivos periféricos.

La arquitectura de la mayoría de las computadoras actuales sigue un diseño basado en la arquitectura definida por Von Neuman. Este diseño, se sigue en la actualidad con muy pequeñas variantes y lo que cambia mayormente es la velocidad de procesamiento o la capacidad de memoria de los componentes.

El desarrollo de esta arquitectura de hardware ha permitido la construcción en

serie y de forma industrial de miles de computadoras que actualmente se usan en nuestro entorno tecnológico y de negocios. Así mismo, ha permitido la integración y el reuso de componentes electrónicos dentro de la arquitectura, y ha motivado el desarrollo de herramientas para la construcción y pruebas de las computadoras.

El avance en la tecnología de desarrollo de software es muy pobre en comparación con lo que existe en la tecnología de desarrollo de hardware. El desarrollo actual de software carece de modelos arquitecturales que permitan desarrollar software a gran escala y que permitan la integración y reuso de componentes dentro de estos modelos. El software es intangible y mucho más complejo que el hardware. Esta complejidad se debe a que existen muchos lenguajes de software, compiladores, sistemas operativos, middleware y entornos de ejecución del software.

Lo más parecido en la actualidad a un componente en software, son las librerías de software de los lenguajes. Sin embargo estos componentes son de muy bajo nivel de abstracción y contienen un número muy bajo de líneas de código de lenguaje. Los lenguajes orientados a objetos han permitido definir a los componentes de software elevando su nivel de abstracción, lo cual ha ayudado a construir componentes más complejos. A partir de este tipo de lenguajes ha sido posible construir componentes más complejos como los patrones arquitecturales. Sin embargo, a pesar de los recientes avances en la tecnología de lenguajes orientados a objetos, los sistemas de software grandes, como los que se utilizan en aplicaciones basadas en Web, son muy complejos de diseñar debido a que no se cuenta con modelos arquitecturales que ayuden en su construcción.

Por otro lado, en la actualidad la mayoría de los sistemas de software que utilizamos son diseñados por programadores y no por ingenieros de software. Con la tecnología actual de lenguajes de programación, estos programadores son capaces de construir eficientes sistemas de tamaño pequeño, sin embargo cuando estos programadores diseñan sistemas grandes producen una gran cantidad de errores, poca documentación y software con difícil reutilización. Este problema se debe a que en muchas ocasiones el programador recibe la especificación de un cliente (de forma oral

en muchas ocasiones) y este de inmediato comienza a programar. De hecho esta ha sido la causa por la cual el software presenta tantos fallos durante su ejecución.

Un ingeniero de software debe documentar todo el proceso de diseño. Los requerimientos deben ser capturados en un documento de especificaciones, el cual sirva como entrada para la fase de diseño. En la fase de diseño, se construye la arquitectura del software así como de los datos y se toma en cuenta el entorno sobre el cual el software estará ejecutando. El documento de diseño, permite conocer la estructura del software a construir. Con este documento se puede repartir el trabajo para los programadores, quienes tienen la tarea de la construcción del software. Una vez construido el software, este debe probarse e instalarse. La falta de alguna de las partes de este proceso produce falta de documentación, posibles fallas en el software y principalmente problemas en la etapa del mantenimiento.

1.2. Planteamiento del problema

En la actualidad no existe una representación formal para la descripción de arquitecturas de aplicaciones Web. Además los estilos arquitectónicos propuestos para el desarrollo de este tipo de aplicaciones no brindan una especificación detallada de la arquitectura de dichas aplicaciones, dando como resultado arquitecturas deficientes y que no cumplen con los requerimientos del sistema y en consecuencia representan aplicaciones con poca calidad.

La arquitectura para aplicaciones Web propone la descripción formal de todas las partes que integran la arquitectura de estos sistemas, como son su estructura, componentes, conectores e interfaces haciendo uso de las herramientas existentes (estilos arquitectónicos, patrones arquitectónicos, frameworks, etc.). Esto permitirá obtener arquitecturas que cumplan, de una mejor forma, con los requerimientos del sistema, dado que estos requerimientos se verán reflejados en la descripción de la arquitectura.

1.3. Objetivo

Esta tesis tiene como principal objetivo, la propuesta y desarrollo de una arquitectura de software que permita la especificación arquitectónica de aplicaciones basadas en Web. La arquitectura provee una vista general de la aplicación, permitiendo analizar su estructura detectando errores de diseño. Para cumplir nuestro objetivo general fue necesario cumplir los siguientes objetivos particulares:

- *Analizar* los diferentes estilos arquitectónicos que existen para aplicaciones basadas en Web.
- *Estudiar* el impacto que tiene la arquitectura de software en el desarrollo de aplicaciones Web.
- *Estudiar y Analizar* los patrones de diseño. El objetivo es la integración a arquitecturas Web
- *Utilizar* el lenguaje de modelado unificado (UML) para la documentación de la arquitectura de software.
- *Proponer* de una arquitectura para aplicaciones Web siguiendo un proceso de ingeniería de software.
- *Desarrollar* la arquitectura de software de una aplicación basada en Web tomando como base la arquitectura propuesta.

1.4. Organización de la tesis

El resto del documento está organizado de la siguiente manera. El capítulo 2 presenta los antecedentes teóricos de la arquitectura de software sobre los cuales está fundamentado nuestro trabajo. Después, en el capítulo 3 realizamos un profundo análisis sobre las aplicaciones Web que nos permiten comprender mejor los retos de la tesis. En el capítulo 4 presentamos una propuesta de arquitectura de software

para aplicaciones Web. El capítulo 5 presenta el caso de estudio que realizamos para validar el trabajo realizado en los capítulos anteriores. Por último, en el capítulo 6 presentamos las conclusiones y proponemos el trabajo futuro de nuestro trabajo.

Capítulo 2

Ingeniería de Software

En este Capítulo presentamos los conceptos relacionados a las arquitecturas de software dentro del campo de la ingeniería de software. Así mismo, enfatizamos la necesidad de comprender los requerimientos y planificar el desarrollo para reducir costos y esfuerzos durante y después del desarrollo. Para poder comprender el dominio del problema que estamos atacando damos una pequeña introducción sobre los temas relacionados, con la arquitectura de software y las técnicas utilizadas para el desarrollo de la tesis. Además describimos brevemente los trabajos relacionados con este trabajo de tesis.

2.1. Modelos de la ingeniería de software

La ingeniería de software es la disciplina de la ingeniería que estudia todos los aspectos relacionados con la producción de software desde la especificación del sistema, hasta el mantenimiento y la evolución del software [4].

La ingeniería de software cuenta con un conjunto de actividades y resultados asociados para obtener un producto de software, el cual se conoce como un proceso de software. Un proceso de software considera cuatro actividades fundamentales tales como especificación, desarrollo, validación y evolución.

Cada sistema requiere un tipo de desarrollo diferente que corresponda al dominio

de su aplicación. Por lo tanto se necesita contar con modelos que permitan describir de forma simplificada los procesos de software (mejor conocidos como modelos de procesos de software). Un modelo de proceso de software es una representación abstracta de un proceso de software, el cual en su mayoría, está basado en modelos de desarrollo de software tales como el enfoque en cascada, el desarrollo iterativo y el desarrollo de software basado en componentes.

El primer modelo de software publicado fue el modelo en cascada (Figura 2.1) el cual incluye las actividades fundamentales del proceso de desarrollo de software. En este modelo para comenzar con la siguiente actividad se necesita haber terminado o pasado por una de las etapas anteriores. Este modelo también se conoce como ciclo de vida del software. Una de las desventajas del modelo en cascada es la necesidad de terminar las etapas anteriores para comenzar con la siguiente, dando como resultado un tiempo de desarrollo más largo.

El desarrollo iterativo acorta las etapas del modelo en cascada, de tal forma que se puede desarrollar una versión reducida del sistema en muy poco tiempo, permitiendo utilizar el sistema y analizar las mejoras para las siguientes versiones. Cada iteración está compuesta de un modelo en cascada reducido y contiene pocos requerimientos. En cada iteración el sistema debe evolucionar hasta convertirse en el sistema que cubra las necesidades del cliente. El enfoque iterativo permite una interacción directa con los stakeholders, permitiendo entender de mejor forma los requerimientos del sistema. Por último, el desarrollo basado en componentes sigue el paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen dentro de un gran sistema de software. Un componente es una pieza de software previamente construido que encapsula una funcionalidad específica y cuenta con interfaces estándar que permiten su uso. El desarrollo basado en componentes incorpora muchas de las características del modelo iterativo. Es evolutivo por naturaleza y exige un enfoque interactivo para la creación del software. Este modelo permite la reutilización del software a gran escala, abstrae el diseño del sistema permitiendo a los ingenieros de software enfocarse en la arquitectura y no en los detalles de cada componente.

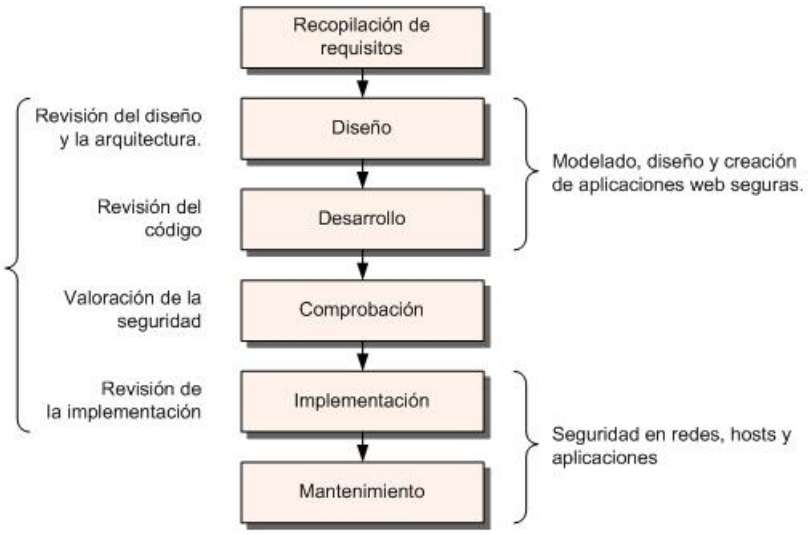

Figura 2.1: Ciclo de vida del software

2.2. Ciclo de vida del software

El ciclo de vida del software (mostrado en la Figura 2.1) es un modelo para la producción del software, el cual incluye cuatro actividades fundamentales:

- Especificación de software: esta actividad es el resultado de la obtención y análisis cada uno de los requerimientos del sistema, en esta actividad se obtiene el documento de especificacioes de todo el sistema. Esta actividad se describe en la Sección 2.2.1.

- Diseño de software: esta actividad consiste en traducir los requerimientos del cliente a un modelo. Este modelo permite definir cada uno de los componentes, subsistemas, interfaces y estructuras que se integran en la aplicación de software, el documento que se obtiene en esta actividad es el diseño de todo el sistema. Esta actividad se describe en la Sección 2.2.2.

- Desarrollo del software: esta actividad consiste en la traducción del diseño en una plataforma tecnológica (lenguajes de programación, servidores web, protocolos de comunicación). El resultado es un producto de software con las características descritas en la actividad anterior, en esta actividad se obtiene un

producto de software que puede ser utilizado por el cliente. Esta actividad se describe en la Sección 2.2.3.

- Validación del software: en esta actividad se analiza la funcionalidad a distintos niveles (clases, componentes, subsistemas, módulos) para comprobar que los requerimientos fueron cumplidos. Esta actividad se describe en la Sección 2.2.4.

- Evolución del software: esta actividad se agregan nuevas características al software desarrollado o corregir los defectos que se hayan detectado durante su uso. Esta actividad se describe en la Sección 2.2.5.

En el ciclo de vida del software estas actividades son realizadas de forma secuencial, de manera que el producto de cada uno de las actividades es la entrada de otra.

A continuación se describe brevemente en qué consiste cada una de las actividades del ciclo de vida del software.

2.2.1. Obtención y análisis de requerimientos

La obtención y el análisis de requerimientos es un proceso que se lleva a cabo para la comprensión y definición de los servicios que se quieren del sistema. En esta etapa también se identifican y definen las restricciones del funcionamiento y desarrollo. La etapa de obtención y análisis de requerimientos también se le conoce como ingeniería de requerimientos.

La obtención y análisis de requerimientos es una etapa crítica en el ciclo de vida del software, ya que uno de los principales problemas es la especificación de los requerimientos que debe cumplir el software. Estos requerimientos permiten entender las principales funciones que deben incluirse en el software. Los errores cometidos en esta etapa darán como resultado inevitables problemas en las etapas posteriores.

El producto de esta etapa será un documento que contenga la especificación del sistema. En este documento los requerimientos son representados en dos niveles de detalle. En el primer nivel, los clientes y usuarios finales requieren una representación

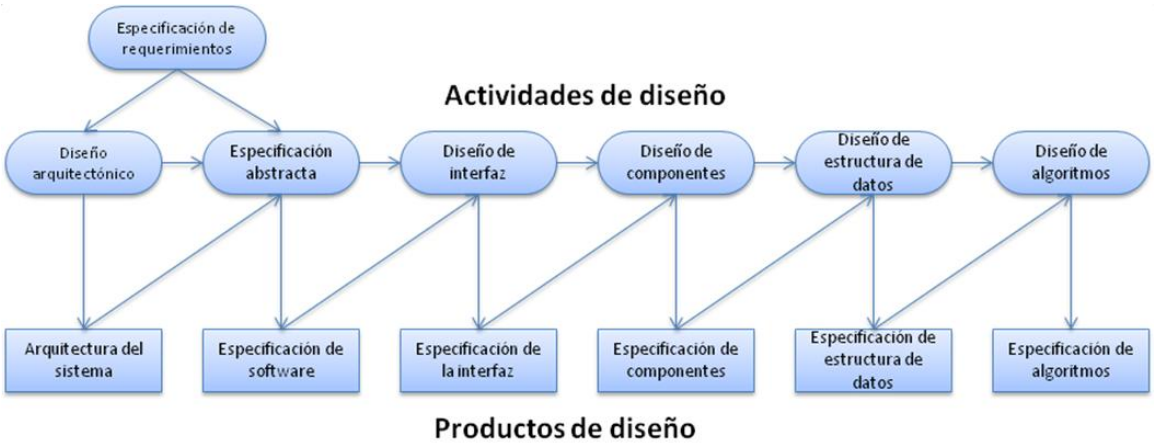

Figura 2.2: Modelo general del proceso de diseño

de alto nivel que les permita comprender los requerimientos. Por otra parte, en el segundo nivel, los programadores y desarrolladores necesitan una representación más detallada de los requerimientos.

2.2.2. Diseño

Una vez terminada la etapa de especificación del software se puede comenzar con la etapa de diseño.

En esta etapa se describe la estructura, los componentes, los conectores, las interfaces y los algoritmos utilizados en el software que se desea implementar. La etapa de diseño se desarrolla de forma iterativa a través de varias versiones, rara vez los diseñadores llegan a un diseño detallado inmediatamente.

En la Figura 2.2 se muestran las fases de la etapa de diseño, teniendo como documento inicial la especificación de los requerimientos y como resultado la especificación de todo el proceso de diseño.

- El *diseño arquitectónico* es una actividad que permite estructurar de forma general al sistema, el objetivo es tener una visión clara del sistema a construir. El resultado de esta fase de diseño es la arquitectura del sistema.

- La *especificación abstracta* es una actividad que toma la arquitectura y la especificación de requerimientos para definir al software en general. El resultado de esta etapa es el documento de especificación del software.

- El *diseño de interfaces* es una actividad que necesita estudiar el flujo de la información para analizar la comunicación entre los distintos componentes del sistema. Como resultado de esta actividad se tiene el documento de especificación de las interfaces.

- La *especificación de componentes* permite identificar grupos de elementos para formar componentes. Cada componente permite abstraer una funcionalidad para poder ser reutilizado. El resultado de esta actividad es el documento de especificación de componentes.

- El *diseño de las estructuras de datos* es la actividad que detalla cada una de las clases y objetos que son necesarios para construir el sistema.

- El *diseño de algoritmos* permite especificar cada algoritmo utilizado en el sistema. En esta actividad se analiza el desempeño del sistema, si existen deficiencias en los algoritmos se corrige evitando errores de implementación.

Autores como [5] [6] [7] han considerado el diseño como una actividad importante que debería ser el centro de todo el desarrollo de software. A este nuevo enfoque se le conoce como desarrollo centrado en la arquitectura [8]. Este enfoque introduce grandes ventajas en comparación con los enfoques tradicionales como el modelo en cascada.

El enfoque consiste en aprovechar todos los beneficios de la arquitectura de software para beneficio del producto. Un sistema con una buena arquitectura está destinado al éxito, por el contrario un sistema con una arquitectura deficiente está condenado al fracaso [9]. Los grandes proyectos de software tienen grandes posibilidades de fallar. El reporte del grupo Standish [9] declara que cerca de un tercio de los proyectos son

cancelados antes de completarse y más de la mitad sufren serios incrementos en su costo.

Los beneficios que nos provee el desarrollo centrado en la arquitectura comprenden a las demás etapas de desarrollo como son obtención y análisis de requerimientos, implementación, pruebas y evolución del software. Esto se debe a que la arquitectura es la representación formal de todo el sistema y puede usarse para localizar errores existentes en el software [7].

Diseño arquitectónico

Debido a la complejidad de los sistemas grandes, se ha hecho necesaria la subdivisión del sistema en subsistemas, los cuales proporcionen servicios relacionados. El diseño arquitectónico es la primera etapa del proceso de diseño en donde se identifican estos subsistemas y se establece un marco de control y comunicación entre ellos.

El diseño arquitectónico juega un papel muy importante en la ingeniería de diseño y la ingeniería de requerimientos, siendo este el enlace entre ambas etapas.

Como se mencionó anteriormente centrar nuestro desarrollo en la arquitectura y más específicamente en el diseño y la documentación de una buena arquitectura traerá ventajas tales como:

- *Comunicación* entre los diferentes *actores (stakeholders)* que participan en el desarrollo. Esto se debe a que la arquitectura es una representación de alto nivel la cual puede ser el punto de partida de intercambio de ideas y opiniones de los diferentes puntos de vista que se tiene del sistema.

- *Análisis del sistema.* Dado que la arquitectura describe la estructura de todo el sistema, es posible realizar un análisis en etapas tempranas del desarrollo permitiendo localizar los errores de diseño y así tomar decisiones de diseño. En este análisis también se evalúa si el sistema puede cumplir con los requerimientos no funcionales como son rendimiento, escalabilidad, confiabilidad, entre otros.

- *Reutilización.* La arquitectura nos muestra los diferentes componentes del sistema y la forma en cómo estos interactúan entre sí. Una buena arquitectura nos permitirá identificar a los componentes que se pueden reutilizar. Como sabemos, existen arquitecturas similares a la de otros sistemas, permitiendo la compatibilidad de sus componentes.

La arquitectura afecta directamente al rendimiento, la confiabilidad, la distribución y la mantenibilidad del sistema, por lo tanto, el diseño de la arquitectura puede depender de los requerimientos no funcionales como la seguridad, el rendimiento, la mantenibilidad o la disponibilidad.

Hasta ahora se ha presentado la etapa de diseño, y la actividad de diseño arquitectónico. A continuación se presentan algunas definiciones que se consideran importantes para la comprensión de la arquitectura de software.

Arquitectura

Las decisiones de diseño de alto nivel de un sistema de software suelen ser más críticas que los detalles de los algoritmos y estructuras de datos utilizados, especialmente cuando su tamaño y complejidad crece. La arquitectura de software ha sido descrita en [10], [11] como un medio para documentar las decisiones de diseño de alto nivel para un software complejo.

Algunos investigadores definen a la arquitectura de un sistema de software como el conjunto de decisiones de diseño tomadas para el sistema [7]. Haciendo una analogía con la construcción de edificios, se puede decir que la arquitectura representa los planos (de diseño, de organización y de construcción) del software.

Con el uso de diagramas simples, los diseñadores pueden documentar y explicar de manera fácil sus decisiones de diseño. Si bien el término “arquitectura de software ha sido propuesto recientemente, muchos investigadores están de acuerdo en que la arquitectura de software de un sistema se refiere a los diferentes componentes del sistema, las interacciones entre ellos y la configuración del sistema.

Para poder tener éxito en el desarrollo del software se necesita comprender sus requerimientos, las reglas del negocio y el comportamiento de la aplicación, para poder proponer una arquitectura acorde al dominio del software. Hoy en día se cuenta con varias arquitecturas de referencia o estilos arquitectónicos que nos pueden ayudar a identificar y diseñar la arquitectura del sistema.

Con el surgimiento de Internet, a mediados de los 90's, surgió un nuevo problema al momento de diseñar software. A medida que las aplicaciones crecen en tamaño y complejidad, es más difícil identificar la arquitectura del sistema debido a que su distribución, comportamiento y seguridad cambian respecto a las aplicaciones locales. Estos problemas fueron abordados por varios investigadores quienes proponen arquitecturas para aplicaciones Web [11], [12], [13].

Vistas arquitectónicas

Una arquitectura de software incluye muchos detalles de la implementación y es utilizada por varios *actores* en la organización. Para evitar sobrellenar el diagrama de arquitectura con muchos detalles, varios investigadores han propuesto varias vistas que abordan las necesidades de *actores* específicos. Por ejemplo, en lugar de especificar el flujo de datos y la concurrencia de un gran sistema, dos diagramas (vistas) separados deben usarse para representar estos aspectos separadamente. En esta sección presentamos el trabajo de Kruchten [14] sobre las vistas arquitectónicas o arquitecturales.

Kruchten propone el modelado de arquitecturas utilizando cuatro diferentes vistas y una vista de casos de uso para ilustrar y validar las otras vistas. Cada vista aborda un enfoque específico de la arquitectura para un conjunto particular de *actores*. En *4+1 View Model of Architecture* (Figura 2.3), Kruchten define las siguientes vistas:

- La *vista Lógica* representa los requerimientos funcionales del sistema, usando abstracciones elaboradas desde el dominio del problema. Esta vista es utilizada por el usuario final para asegurarse que todos los requerimientos funcionales han sido considerados en la implementación del sistema.

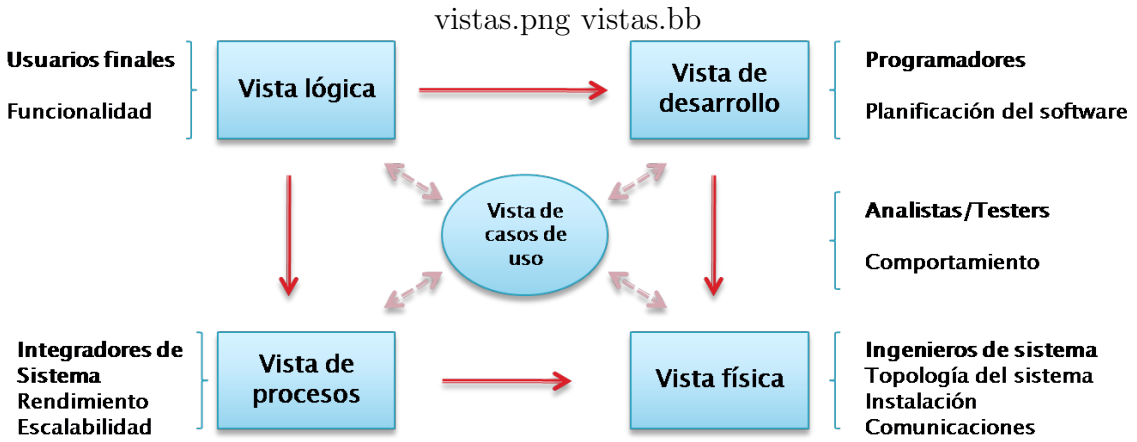

Figura 2.3: Modelo 4+1 vistas de la arquitectura

- La *vista de procesos* describe los mecanismos de concurrencia y sincronización usados en el sistema. Esta vista es utilizada por los integradores del sistema para el análisis del rendimiento y escalabilidad del sistema.

- La *vista de desarrollo* muestra el diseño del código en el ambiente de desarrollo. Esta vista es utilizada por los lideres de proyecto y programadores, y tiene como fin ayudar en la planeación y la evaluación del progreso del proyecto.

- La *vista física* describe el mapeo del software en el hardware y refleja los aspectos de la distribución. Los Ingenieros en Sistemas desarrollan esta vista para determinar la topología del sistema y los requerimientos de comunicación entre los distintos componentes.

- Los *escenarios* son el termino *+1* dentro del *4+1*. Los escenarios ilustran las distintas decisiones que son tomadas a lo largo de las cuatro vistas. En los escenarios se capturan las características generales del sistema.

Cada una de las vistas describen de forma general al sistema desde una perspectiva particular. Cada una de las vistas tiene una notación particular de representación. En nuestro caso usaremos el Lenguaje de Modelado Unificado para describir las vistas.

4+1 Vistas	Arquitectura	Diagramas UML
Vista lógica	Lógica	Clases, de estados y colaboración
Vista de procesos	Procesos	Actividad, estados y de secuencia
Vista de desarrollo	Implementación y datos	Componentes
Vista física	Despliegue	Despliegue
Escenarios	Requerimientos y QoS	Casos de uso

Tabla 2.1: Mapeo de vistas a diagramas UML

En la Tabla 2.1 se muestra el mapeo de las vistas, propuestas por Kruchten, de la Arquitectura a diagramas UML.

Soni [15] también propuso una descomposición similar para comprender mejor la arquitectura de grandes sistemas. Basado en estudio de la construcción de grandes sistemas, Soni propone la siguiente clasificación de la arquitectura:

- *La arquitectura conceptual* abstrae el diseño del sistema utilizando elementos de diseño de alto nivel. Esto permite mostrar la relación que existe entre los elementos de alto nivel.

- *La arquitectura de módulos* presenta la implementación ideal sin depender de las características particulares de un determinado leguaje. Esta arquitectura permite descomponer el sistema utilizando descomposición en capas funcionales.

- *La arquitectura de ejecución* se refiere a la topología del sistema desplegado. Muestra la localización de los distintos componentes y la comunicación que existe entre ellos.

- *La arquitectura de código* muestra la organización del código fuente, los binarios y las librerías en el entorno de desarrollo.

En su estudio, Soni señala que cada arquitectura abarca diferentes etapas a lo largo del proceso de desarrollo como la etapa de diseño, de implementación, de construcción y de ejecución. También señala que el grado de cambio en cada arquitectura varía considerablemente. Por ejemplo, la arquitectura conceptual raramente cambia, mientras que la arquitectura de ejecución sigue cambiando durante el tiempo de vida

del producto para soportar los nuevos requerimientos de rendimiento, aprovechando al máximo los avances tecnológicos en tecnologías de hardware y software.

Componentes

Las decisiones tomadas para la arquitectura de un sistema de software comprenden una amplia composición e interacción de varios elementos diferentes. Estos elementos se encargan de los problemas clave del sistema, incluyendo: (a) procesamiento, que también puede ser referenciado como funcionalidad o comportamiento, y (b) estado, que también puede ser referenciado como información o datos.

Los componentes que encapsulan al procesamiento y los datos en una arquitectura de software son llamados "componentes de software. Un componente de software es una entidad arquitectural que encapsula un subconjunto de funcionalidades del sistema y/o datos, restringe el acceso a ese subconjunto a través de una interfaz definida explícitamente, y define explícitamente las dependencias requeridas en su contexto de ejecución [7].

Un componente de software puede ser muy simple como una sola operación o muy complejo como todo un sistema, dependiendo de la arquitectura, la perspectiva de los diseñadores y las necesidades del sistema. Otra característica de los componentes es que pueden ser vistos por sus usuarios desde el exterior y únicamente por sus interfaces que fueron elegidas para hacerlo público. Por otro lado se puede ver al componente como una caja negra que recibe datos por medio de su interfaz y devuelve una salida. Los componentes son ejemplos claros de encapsulación, abstracción y modularidad.

Conectores

Los componentes son los encargados del procesamiento o los datos, o ambos a la vez. Otro aspecto fundamental de los sistemas de software es la interacción de los bloques que integran al sistema.

Muchos sistemas modernos están desarrollados para soportar el uso de componentes distribuidos. Los conectores distribuidos pueden encapsular todas las librerías

necesarias para la comunicación de los componentes, permitiendo así su reutilización en varios tipos de aplicaciones.

Un conector puede resultar un elemento crítico ya que es necesario para la integración de funcionalidad de varios componentes y obtener el resultado esperado. Entonces, podemos decir que un conector de software es el encargado de gestionar las interacciones entre los componentes.

En otras palabras se define a un conector de software como un elemento arquitectural encargado de efectuar y regular las interacciones entre componentes.

Nótese que mientras la mayoría de los componentes proveen servicios específicos a la aplicación, los conectores son típicamente independientes de la aplicación. En la mayoría de los casos el conector es construido sin un propósito específico, por ejemplo tenemos métodos como publicación, suscripción, notificación de eventos asíncronos, y las llamadas de procesos remotos.

Estilos arquitectónicos

En la sección anterior se mencionaron cada uno de los elementos que son indispensables para modelar una arquitectura y que dependiendo del dominio del software a construir será la interacción entre dichos elementos. Podemos definir a un estilo arquitectónico como el conjunto de elementos (componentes y conectores) que pueden ser usados para describir una familia de sistemas y el conjunto de restricciones requeridas para poder combinarlos.

2.2.3. Desarrollo del software

Una vez que se ha diseñado la arquitectura del software, la etapa siguiente es la implementación. En esta etapa los componentes, conectores e interfaces de la arquitectura comienzan a ser expresados en código de un cierto lenguaje. Los encargados de esta etapa son los programadores quienes convierten el diseño en un programa ejecutable. El avance de la etapa de desarrollo será reflejado en la funcionalidad que está siendo agregada al software.

En la etapa de desarrollo se define el método para construir el software. La elección del método depende en gran manera del dominio de la aplicación. Actualmente se utilizan métodos ágiles que permiten un desarrollo rápido [16],[17].

En la mayoría de los casos, los programadores comienzan a desarrollar los componentes del software que comprenden mejor, dejando al último los componentes que no comprenden. Para verificar que cada componente cumple su función o especificación, los programadores hacen pruebas unitarias de cada componente y al final realizan pruebas de todo el sistema, ya que un componente puede funcionar correctamente de forma individual, pero puede presentar problemas al interactuar con otros componentes.

2.2.4. Validación del software

La validación es una actividad que permite analizar el comportamiento del software teniendo como métricas los requerimientos del software especificados en el documento de requerimientos.

La validación debe llevarse a cabo para todas las etapas del proceso de desarrollo de software. Esta comienza con la revisión de los requerimientos y continúa con revisiones de diseño e inspecciones de código (clases y objetos). También permite comprobar que el producto de software desarrollado cumple con su especificación y que satisface los requerimientos funcionales y no funcionales. Al final, la validación más importante es la del cliente el cual tiene que dar su aprobación de que el software cumple son sus expectativas.

Debido a que la validación es un proceso costoso es necesario comprender mejor los requerimientos en etapas tempranas con el fin de poder garantizar que la validación sea más rápida y menos costosa.

2.2.5. Instalación del software

Una vez que se ha terminado la etapa de desarrollo y que se han realizado las pruebas necesarias para verificar que el producto de software cumple con las especificaciones, se inicia la etapa de instalación. Esta etapa consiste en crear las condiciones necesarias para que el software sea instalado y que pueda funcionar sin ningún problema.

En algunos casos es necesario cumplir requisitos tanto físicos (espacio, temperatura, ventilación, etc.) como de software (servidores de aplicaciones y manejadores de base de datos) que permitan un buen funcionamiento del producto de software. A veces estos requerimientos ocasionan que el software no cumpla los requerimientos o, en el peor de los casos, que el software falle. Una vez que el software ha sido instalado se realizan pruebas para verificar que su comportamiento es correcto.

2.2.6. Evolución del software

El mundo en que vivimos cambia constantemente y eso se ve reflejado también en el software. El software que se construyó hace un año puede no cumplir con las necesidades actuales del cliente, por que se necesita adaptar el software a las nuevas necesidades del cliente. Por lo tanto, la etapa de evolución del software consiste en mejorar el software existente, corrigiendo las deficiencias y agregando nuevas funcionalidades. Comúnmente esto se logra a través de versiones, cada una de las cuales permitirá obtener un software con mejor funcionalidad, ya sea agregando nuevos requerimientos o corrigiendo los existentes.

2.3. Trabajos relacionados

En esta sección se describen las propuestas de arquitecturas para aplicaciones Web.

2.3.1. Transferencia de Estado Representacional (REST)

REST (*Representational State Transfer*) [13]es un estilo arquitectónico para aplicaciones de hipermedia distribuidas. REST está basado en la descripción de un conjunto de restricciones arquitectónicas centradas en el dominio de las aplicaciones Web. REST hace uso de tres vistas (Proceso, conector y datos) para especificar la arquitectura Web. Laespecificación de REST ignora los detalles de la implementación del componente y la sintaxis del protocolo para enfocarse en los roles de los componentes y las interacciones que se tienen entre los componentes.

Las restricciones de REST se fundamentan en los conectores, componentes y datos, quienes definen la base de la arquitectura Web.

REST ayuda al diseño de aplicaciones Web y se enfoca en definir requisitos no funcionales de la aplicación, como son escalabilidad, accesibilidad, seguridad y soporte para sistemas legados.

2.3.2. Recuperación de la arquitectura de software de aplicaciones Web

Hassan y Holt [2] argumentan que en un futuro las aplicaciones Web serán grandes sistemas legados difíciles de mantener. De acuerdo a esa suposición proponen un enfoque que realice la recuperación de la arquitectura de una aplicación Web partiendo de su implementación.

El objetivo principal de hacer ingeniería inversa para obtener la arquitectura es mejorar el mantenimiento y no el desarrollo como tal.

El enfoque define un conjunto de extractores que analizan el código fuente y binario de las aplicaciones Web y obtienen diferentes diagramas de arquitectura de diferentes niveles de abstracción.

En la aproximación se han definido un conjunto de cinco componentes identificados como comunes dentro de la arquitectura Web. Estos componentes son Static Pages (páginas estáticas), Active Pages (páginas activas de servidor), Web Objects

(componentes de servidor), Multimedia Objects (objetos multimedia como imágenes, video y sonido) y Databases (Bases de datos).

La obtención de la arquitectura se basa en un conjunto de tres modelos, que de forma progresiva reducen los detalles obtenidos en la recuperación de los datos. En el modelo ELS (*Entity-Level Schema*) se definen las relaciones que existen entre los componentes que viven dentro de las aplicaciones Web (objetos, tablas, variables, etc.) y a partir de este modelo se define un conjunto de transformaciones necesarias para subir el nivel de abstracción hasta llegar al modelo CLS (*Component-Level Schema*).

CLS es un modelo que representa las relaciones entre los componentes de la aplicación Web (páginas estáticas, páginas de servidor y bases de datos) y nuevamente se aplican una serie de transformaciones para obtener el último modelo ALS (*Abstract-Level Schema*).

Finalmente, ALS es un modelo que muestra las relaciones que existen entre los componentes de mayor tamaño, como los subsistemas, y componentes que los contienen.

Para la representación de la arquitectura el esquema utiliza como lenguaje de modelado diagramas de entidad relación conocidos como Schemas.

2.4. Resumen

La ingeniería de software y las técnicas de modelado ayudan a los diseñadores a construir grandes sistemas de software. En la última década el campo de la arquitectura de software ha hecho grandes contribuciones [11], [13], [9], [32]. Sin embargo, el concepto no ha madurado y en la actualidad es una actividad bastante compleja. La arquitectura de software propone ser la pieza clave para el éxito de los desarrollos de sistemas de software.

El uso de metodologías de desarrollo centrada en la arquitectura modificaría el flujo de las etapas del desarrollo hacia el diseño. Este cambio de enfoque permite a los diseñadores visualizar todos los aspectos del software sin necesidad de construir

el sistema.

En el siguiente Capítulo se describen las características de las aplicaciones Web, se presenta el modelado de las vistas de la arquitectura de acuerdo a la taxonomía de la aplicación.

Capítulo 3

Aplicaciones Web

Con la llegada de Internet han surgido las aplicaciones Web. Este tipo de aplicaciones permiten usar la infraestructura de la Web para desarrollar aplicaciones globales. Reportes recientes indican que las aplicaciones Web representan más de la mitad del total de todas las aplicaciones de la industria de software. Esa cifra indica que las aplicaciones Web siguen creciendo y ganando popularidad en un mercado muy competitivo, y que se perfilan a ser las aplicaciones del futuro.

En este capítulo, se presenta la definición de una aplicación Web. Así mismo se analiza el dominio de las aplicaciones Web y se presenta una taxonomía de las diferentes aplicaciones Web que existen. Por último se describe cada una de las vistas arquitecturales de una aplicación Web, siguiendo las 4+1 vistas de Kruchten [14]

3.1. Definición de una aplicación Web

Las aplicaciones Web usan la infraestructura de la Web (protocolos, lenguajes, etc.) para su funcionamiento. Hoy en día las aplicaciones Web han crecido hasta convertirse en grandes sistemas distribuidos complejos y que pueden atender a millones de usuarios de forma simultánea.

Las aplicaciones Web utilizan tecnología basada en Web como son los navegadores y los servidores Web. Esto permite tener el acceso a un mundo de aplicaciones por

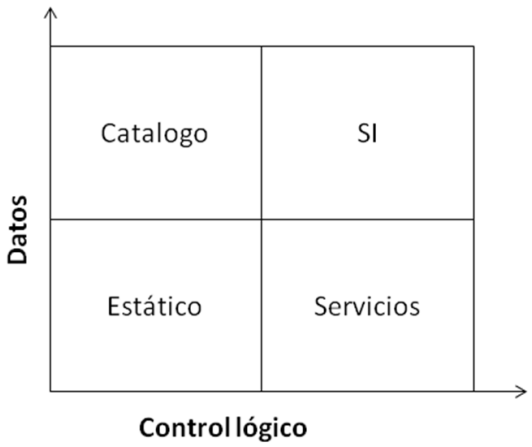

Figura 3.1: taxonomía de aplicaciones Web

medio de una sola interfaz que es el navegador.

Las aplicaciones Web han ganado terreno sobre las aplicaciones tradicionales (de escritorio, locales, etc.) por las siguientes razones:

- Las aplicaciones Web son más accesibles: gracias al uso de protocolos estándar como HTTP toda aplicación Web puede ser usada por toda la Web sin importar ubicación y plataforma.

- Las aplicaciones Web son más fáciles de mantener: desde el momento en que una aplicación se despliega en el servidor, los clientes puede acceder a ella, sin necesidad de una aplicación cliente. Esto permite que cualquier cambio sólo se haga en el servidor y no en el cliente, ahorrando tiempos de desarrollo.

3.2. Taxonomía de aplicaciones Web

Las aplicaciones Web no están limitadas a un dominio, se puede tener aplicaciones para todo tipo de necesidades. A continuación presentamos la taxonomía de las aplicaciones Web.

Como se puede observar en la Figura 3.1 existen cuatro tipos principales de aplicaciones Web:

- Estáticas: este tipo de aplicaciones representan a la primera generación y están compuestas por páginas Web estáticas, imágenes y texto, pero no cuentan con una lógica de negocio. Dentro de este tipo de aplicaciones se encuentran las páginas personales.

- Orientadas a servicios: este tipo de aplicaciones intentan ofrecer un servicio especializado, por lo cual implementan una lógica de negocio acorde al servicio ofrecido. Durante su mantenimiento los desarrolladores necesitan comprender claramente la lógica de negocio. Un ejemplo de este tipo de aplicaciones son los servidores de correo electrónico.

- De datos: este tipo de aplicaciones está enfocado a proveer una interfaz para acceder a una gran cantidad de datos y no en la lógica de negocio, por lo tanto los desarrolladores necesitan comprender el flujo de datos. Un ejemplo de este tipo de aplicaciones son los catálogos en línea de las bibliotecas.

- Sistemas de información: combinan las aplicaciones orientadas a servicios y de datos. Los desarrolladores necesitan comprender claramente el flujo de datos y la lógica de negocio (especialmente en la manipulación de los datos). Ejemplos de este tipo de aplicaciones son la banca en línea y los portales de comercio electrónico.

3.3. Vistas arquitectónicas de una aplicación Web

Las aplicaciones Web son aplicaciones distribuidas que usan las tecnologías Web como su infraestructura. Usan un navegador Web como clientes, el protocolo HTTP para comunicarse entre clientes y servidores, y el lenguaje HTML para expresar el contenido transmitido entre clientes y servidores. Hoy en día existen muy diversas tecnologías para desarrollar aplicaciones Web, por tal motivo una sola vista arquitectónica es complicada e insuficiente para describir todas las necesidades del sistema. En esta sección describimos las vistas arquitectónicas de una aplicación Web y nos

centramos en las vistas propuestas de Kruchten [14] (*Vista Lógica, de Procesos, Física* y la *Vista de Desarrollo*). Estas vistas son descritas en la Figura 2.3. Cada vista captura decisiones especificas de diseño y para una mejor comprensión de de la aplicación pueden observarse todas las vistas en conjunto.

Es importante hacer notar que una aplicación Web tiene diferentes necesidades con respecto a otro tipo de aplicaciones, por lo cual es necesario complementar el modelo de las 4+1 vistas. Dado que una aplicación Web puede estar compuesta por varios componentes distintos y que a su vez estos componentes pueden comunicarse por medio de Internet, la aplicación puede volverse más vulnerable y blanco fácil de ataques.

La seguridad en las aplicaciones Web es un tema muy complejo por lo cual es necesaria otra vista en donde se especifique las decisiones de diseño de acuerdo a la seguridad. Esta vista complementa el modelo de las 4+1 vistas.

3.3.1. Vista lógica

La vista lógica se centra principalmente en los requerimientos funcionales (por ejemplo, los servicios que el sistema debe de proporcionar a sus usuarios). El sistema se descompone en una serie de abstracciones clave, tomadas (principalmente) del dominio del problema y en la forma de los *objetos* o las *clases de objetos*.

Sobre estos se aplican los principios de abstracción, encapsulamiento y herencia. Esta descomposición no sólo se hace para potenciar el análisis funcional, sino también sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema. Los diagramas se usan para representar los distintos componentes del sistema y la interacción que existe entre estos.

A nivel arquitectural las aplicaciones Web se componen de tres capas: *la capa de presentación*, *la de lógica de negocio* y *la capa de persistencia*. En la Figura 3.2 se muestra la descomposición de las capas de una aplicación Web y cuál es la relación que existe entre estas.

Como se puede observar, el número de capas varía de acuerdo a la complejidad de

la aplicación, sin embargo describiremos sólo las capas mencionadas anteriormente.

La *capa de presentación* se encarga de mostrar a todos los componentes involucrados en la generación de la interfaz de usuario y la relación que existe entre estos. Ejemplos de estos componentes son los botones, las imágenes, el texto y las formas. Los componentes de esta capa sólo se pueden comunicar con la capa de lógica de negocio, aunque esto depende de las restricciones de diseño que se deseen adoptar, ya que al incluir patrones de arquitectura esta restricción puede desaparecer.

La *capa de lógica de negocio* contiene todas las reglas de negocio necesarias para modificar los componentes de datos contenidos en la capa de persistencia. Las reglas de negocio permiten al sistema realizar una función que cumpla los requerimientos de los usuarios. La capa de lógica de negocio sólo se encarga de modificar los datos contenidos en la capa de persistencia y enviarlos a la capa de presentación para mostrarlos al usuario. En una aplicación Web para comercio electrónico esta capa es la encargada de validar usuarios, verificar la existencia de los productos, validar los pagos electrónicos, etc.

La *capa de persistencia* contiene todos los componentes necesarios para dotar de persistencia a los datos de la aplicación, por ejemplo, una base de datos que contenga los datos de los clientes y proveedores.

Una arquitectura basada en tres capas provee una muy buena separación de conceptos (código fuente, interfaz y reglas de negocio) para las aplicaciones Web, pero otros tipos de aplicaciones no requieren la separación de la capa de lógica de negocio y la capa de persistencia. Por ejemplo en una aplicación orientada a servicios no tiene clara la separación entre dichas capas, por lo tanto podríamos usar una arquitectura en 2 capas, en donde una capa que se encargue de la interfaz y otra capa permita el acceso a los datos y las reglas para modificarlos.

3.3.2. Vista de procesos

La vista de procesos presenta la concurrencia y distribución de los procesos de una aplicación. Un usuario de una aplicación Web interactúa con el navegador Web pero

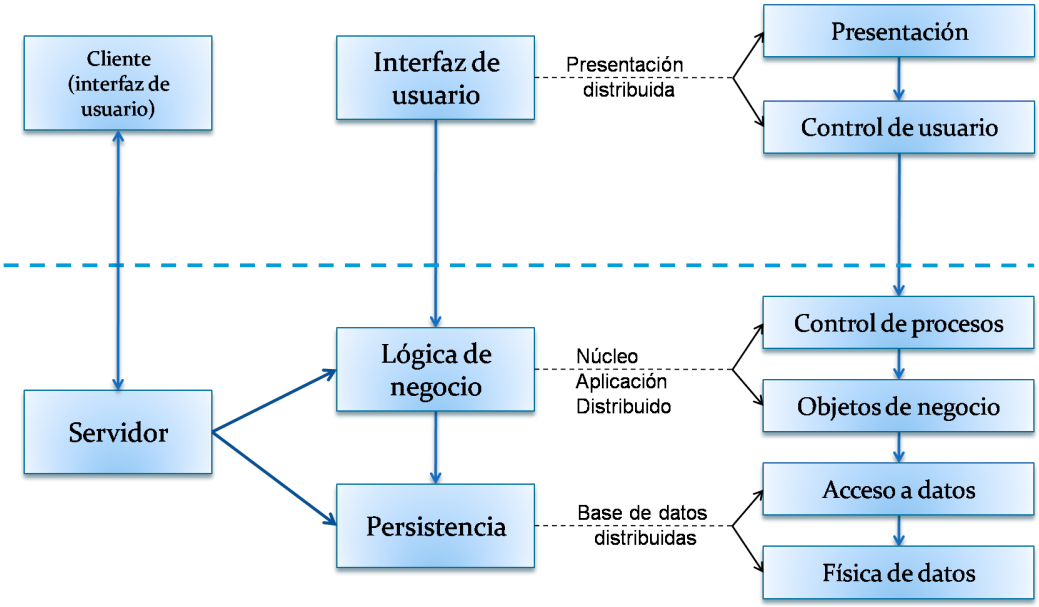

Figura 3.2: Vista lógica

el usuario nunca interactúa directamente con el servidor. El navegador Web traduce los eventos señalados (clics) en ligas e imágenes en peticiones HTTP al servidor. Posteriormente el servidor responderá a las peticiones del navegador con respuestas HTTP. En la interfaz de una aplicación Web el navegador Web siempre estará a cargo de mostrar las respuestas del servidor por medio de páginas HTML.

Hoy en día las aplicaciones son demasiado robustas por lo tanto hay que considerar también los procesos que pueden existir por la parte del cliente.

Dada la naturaleza de las aplicaciones Web se separan en dos tipos fundamentales de procesos, (1) los procesos que van a ser ejecutados por el servidor, que son la gran mayoría; y (2) los procesos que van a ser ejecutados por el cliente (procesos ligeros).

3.3.3. Vista física

La vista física muestra el mapeo de los componentes del sistema en los componentes físicos del mismo. Esta vista nos permite observar la distribución del software en el hardware, dado que el entorno de una aplicación Web consta de muchos componentes

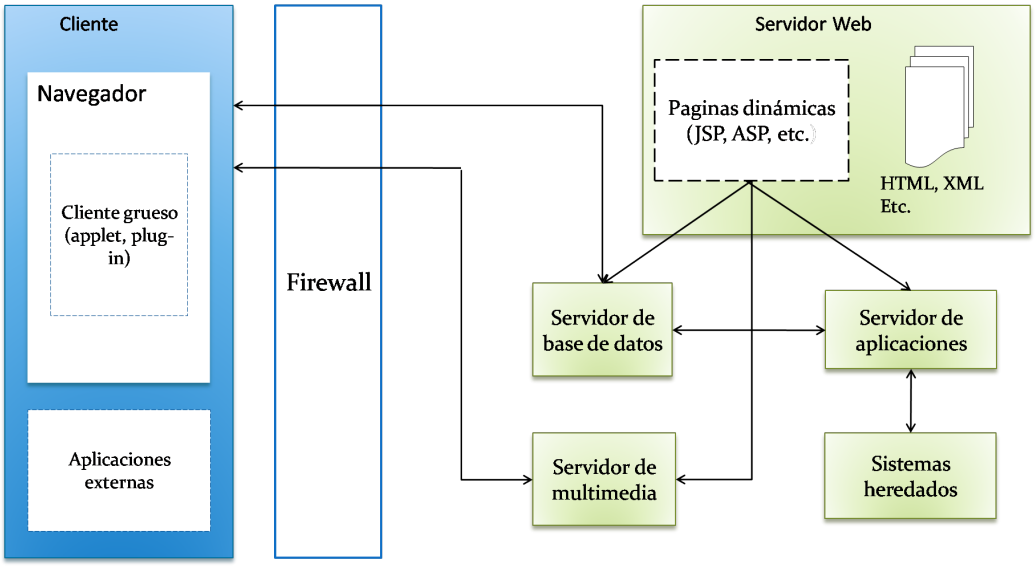

Figura 3.3: Vista física de una aplicación Web

que están interrelacionados entre sí para brindar la funcionalidad necesaria.

Hoy en día las aplicaciones Web son aplicaciones muy grandes que no pueden residir en un solo lugar. Por tal motivo se necesita conocer los recursos hardware para evaluar los requerimientos no funcionales (rendimiento, disponibilidad, escalabilidad).

La Figura 3.3 muestra los distintos componentes de una aplicación Web, así como la relación que existe entre estos.

Componentes del cliente

- *Navegador Web*: es una aplicación que se ejecuta por parte del cliente la cual permite la visualización de archivos en lenguaje HTML.

- *Cliente Grueso*: se define como la aplicación que reside dentro del cliente y que se encarga de procesar la mayoría de los datos para la aplicación Web. La mayoría de clientes gruesos usan al navegador Web como plataforma. Los ejemplos más comunes de clientes gruesos son los *Applet´s* de Java.

- *Aplicaciones externas*: este tipo de aplicaciones son ajenas a la aplicación pero son necesarias para complementar su funcionalidad. Ejemplos de este tipo de

aplicaciones pueden ser los editores de texto y los visualizadores de documentos PDF.

- *Firewall*: en este caso el firewall puede estar instalado en la parte del cliente o puede ser un componente independiente. Un ejemplo de un firewall es una empresa que desea proteger su red interna de usuarios que intentan acceder a esta red.

Componentes de la parte del servidor

- *Servidor Web*: un servidor Web es un programa que permite la transmisión de datos en lenguaje HTML por medio del protocolo HTTP. La información transmitida consta de elementos como paginas HTML que a su vez pueden contener elementos multimedia como fotos, videos y audio. Entre los ejemplos mas conocidos de servidores Web podemos mencionar al servidor Apache.

- *Recursos HTML*: este tipo de recursos se encuentran escritos en lenguaje HTML y son enviados al cliente para ser visualizados en un navegador Web.

- *Servidor de Base de datos*: este servidor será el encargado de administrar los recursos de persistencia así como el acceso y manipulación de los datos.

- *Servidor Multimedia*: este servidor se encarga de administrar todos los recursos multimedia. En ocasiones muchos sitios web (p.ej. youtube.com) necesitan almacenar grandes cantidades de archivos multimedia haciendo compleja la administración.

- *Sistemas Heredados*: son sistemas existentes que se integran a la aplicación Web. Estos pueden ser sistemas administrativos u otros sistemas Web.

La vista física varía de acuerdo al contexto de la aplicación y podemos ver que las vistas físicas de dos aplicaciones pueden ser totalmente diferentes. Por ejemplo una aplicación Web de catálogos hace mayor uso de los servidores de bases de datos y no

así de los objetos distribuidos. Por otro lado una aplicación de banca en línea hace mayor uso de componentes especializados para la seguridad mientras que no hace uso de las aplicaciones externas.

3.3.4. Vista de desarrollo

La vista de desarrollo se centra en la organización real de los módulos del software en el ambiente de desarrollo del mismo. El software se empaqueta en partes pequeñas (componentes, bibliotecas, subsistemas) que a su vez se relacionan entre sí. En el desarrollo de las aplicaciones Web se utilizan una gran variedad de tecnologías para poder satisfacer los requerimientos (funcionales y no funcionales) de la aplicación. En la vista de desarrollo para una aplicación Web hay que destacar ciertos detalles, tales como:

- *Mapa de navegación*: permite tener una vista general de todas las páginas que contiene la aplicación y la relación que existe entre estas.
- *Técnicas de administración de sesiones de usuario*: permiten tener un control sobre el acceso del usuario a la aplicación.
- *Tecnologías para le generación de páginas de aplicación*: son tecnologías utilizadas para la generación de páginas dinámicas por medio de un lenguaje de programación.
- *Técnicas de seguridad en los distintos niveles*: son técnicas que permiten la protección de los datos del usuario y limitan el acceso a usuarios no autorizados.

Varias páginas Web son relacionadas para formar una aplicación Web. Es necesario verificar periódicamente las ligas entre diferentes páginas para evitar la falta de disponibilidad de las diferentes partes de la aplicación Web. Estos problemas son considerados como *bugs* o errores en el sistema.

Como se mencionó anteriormente, las aplicaciones Web usan el protocolo HTTP como medio de comunicación. El protocolo HTTP es un protocolo sin estado, en

donde el cliente se conecta al servidor Web por medio de peticiones HTTP. Una vez que el servidor ha contestado la petición del cliente la conexión termina. Si el cliente necesita acceder a otra página, la conexión puede ser restablecida.

Para el servidor Web cada petición es independiente de las demás, por tal motivo, el servidor no puede determinar si las peticiones le corresponden a un mismo cliente, esto permite al servidor atender a un gran número de clientes. Dado que el cliente consume recursos del servidor sólo en el momento que hace una petición, al finalizar la conexión los recursos del servidor son liberados. Sin embargo, en las aplicaciones Web esto es distinto ya que una aplicación necesita saber si las peticiones pertenecen o no a un mismo cliente. Por ejemplo, en una página de Login necesitamos determinar si un usuario se ha identificado por medio de su login y su password, de lo contrario mostraremos la interfaz de acceso para que el usuario se autentifique.

Además, las aplicaciones Web deben ser capaces de determinar si un usuario ha salido de la aplicación. En las aplicaciones tradicionales una vez que el usuario cierra la página de la aplicación, la conexión con el servidor se termina y la aplicación está enterada de que el usuario ha salido. En las aplicaciones Web no sucede lo mismo, ya que el cliente puede salir de la página de la aplicación y visitar otras páginas, no hay señales que se envíen al servidor de que el usuario ha salido. Por tal motivo, es necesario el uso de técnicas que permitan tener conocimiento sobre las acciones del usuario. Algunas aplicaciones Web utilizan el tiempo de espera para determinar si el usuario ha salido de la aplicación.

En resumen, la propia naturaleza del protocolo HTTP sin estado, impide a una aplicación Web reconocer si dos peticiones, que se originaron durante el mismo periodo de sesiones, pertenecen a un mismo usuario. Este problema se conoce como el problema de administración de sesiones. Existes muchas técnicas que ayudan a resolver este problema. El uso de cookies o de campos ocultos para almacenar la sesión del usuario permiten asociar las peticiones a los usuarios. En una aplicación Web se puede tener una combinación de varios métodos que nos permitan dotar de una mayor seguridad a la aplicación.

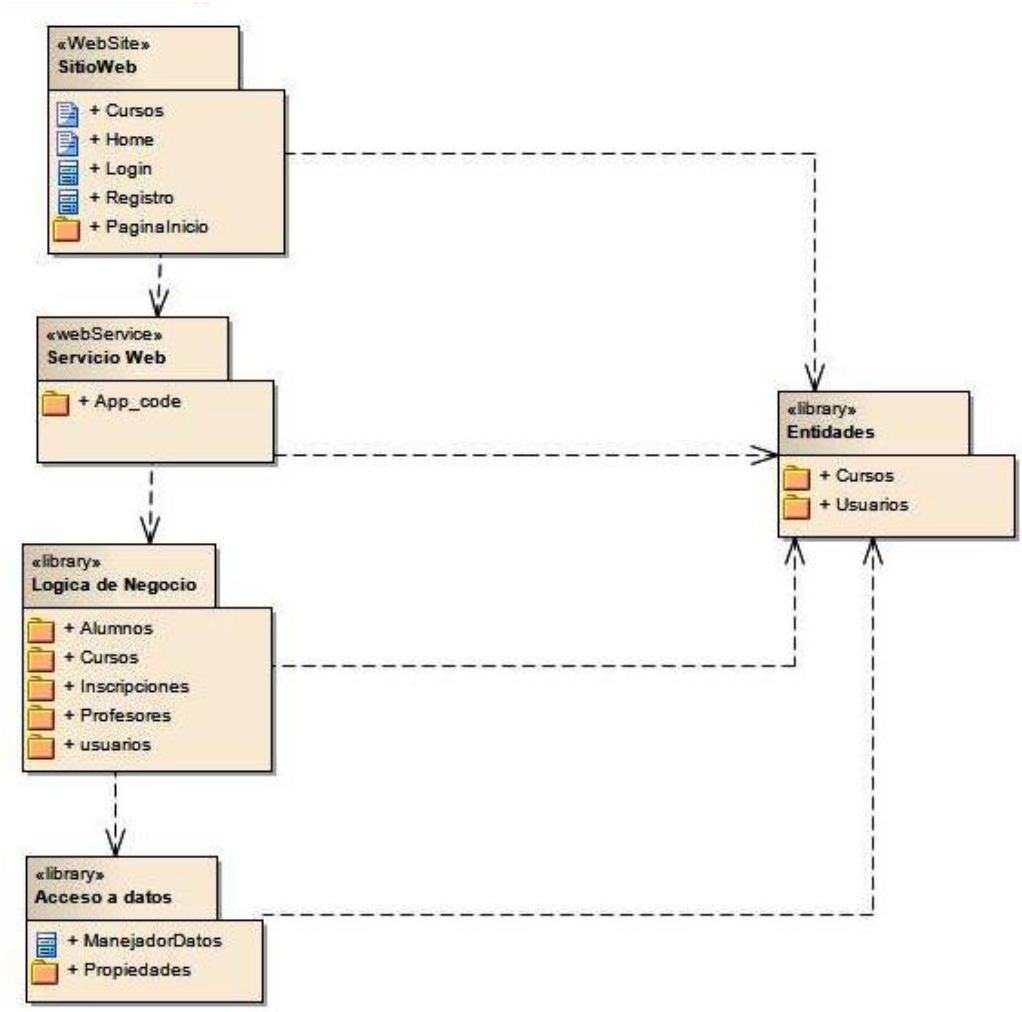

Figura 3.4: Vista de desarrollo de una aplicación Web

En la Figura 3.4 se observa la vista de desarrollo de una aplicación Web. En esta vista se pueden observar las capas que se utilizan para desarrollar la aplicación y la relación que se tienen entre estas.

La vista de desarrollo puede contener las mismas capas de la vista lógica. Las capas de sitio Web y servicio Web corresponden a la capa de presentación de la vista lógica. La capa de lógica de negocio y entidades corresponden a la capa de la lógica de negocio, mientras que la capa de acceso a datos corresponde a la capa de persistencia.

La vista de desarrollo depende en gran manera del entorno de desarrollo. En nuestro caso se trata de una aplicación Web, pero podría ser una aplicación de escritorio

o un servicio.

3.3.5. Vista de seguridad

Las aplicaciones Web están compuestas en la mayoría de los casos de un gran número de componentes distribuidos. Todos estos componentes usan una red no segura como Internet para su comunicación, la cual garantiza su accesibilidad desde cualquier parte, si es que no existe un método de protección. La vista de seguridad refleja la seguridad de la aplicación y los aspectos sobre el control de acceso, permitiendo mostrar los niveles de seguridad necesarios para acceder a los diferentes componentes de la aplicación.

El nivel de seguridad que se desea agregar a una aplicación Web será de acuerdo a los datos que se manejen. Por ejemplo, una aplicación para banca en línea necesita un nivel de seguridad mayor que una aplicación para noticias. La seguridad en una aplicación Web pretende garantizar que la información sólo puede ser consultada por los usuarios que están autorizados para dicha tarea.

La vista de seguridad especifica los métodos de autenticación utilizados, que pueden ser por medio de login y password o los certificados digitales. Además la vista detalla cómo y cuándo se puede usar técnicas criptográficas para proveer seguridad. Por ejemplo, para acceder a los recursos de una compañía, el empleado debe de proporcionar su login y password. Cabe mencionar que se necesita un canal seguro para enviar dicha información.

La vista de seguridad también debe detallar los métodos de seguridad para la protección de los diferentes componentes. Se debe de contar con componentes de hardware y de software que permitan resguardar la integridad de la aplicación. Para lograr obtener una seguridad aceptable dentro de nuestras aplicaciones Web debemos de contar con infraestructura necesaria durante el desarrollo de la aplicación. Muchos diseñadores ignoran la seguridad al momento del diseño, obteniendo aplicaciones que pueden cumplir con los requerimientos funcionales pero que suelen ser demasiado vulnerables a los ataques de terceros.

Los componentes que podemos utilizar para brindar seguridad son los siguientes:

- *Firewalls* (físicos y de software) para la restricción de acceso de terceros a nuestros recursos.
- *Certificados de seguridad* para una confiable y fácil distribución de llaves públicas.
- *Protocolos seguros* para garantizar la seguridad del canal de comunicación.
- *Políticas de seguridad* tales como respaldos, niveles de acceso, restricción de servicios.
- *Técnicas criptográficas* para el envío y recepción de información de forma segura.

Al diseñar una aplicación de software debemos de considerar estas herramientas que nos ayudaran a brindar seguridad, pero también necesitamos definir políticas de seguridad que nos ayuden tener un mejor control del acceso a los recursos.

3.4. Resumen

Las aplicaciones Web son sistemas complejos que utilizan una combinación de plataformas tecnológicas para su funcionamiento y se caracterizan de tener a un navegador Web como su interfaz. Debido a la creciente complejidad de los sistemas de software basados en Web se ha hecho necesario el uso de técnicas, como la reutilización de componentes, que permitan garantizar el éxito de los desarrollos de estas aplicaciones.

Desafortunadamente, las herramientas actuales para desarrollo de sistemas basados en Web se han enfocado en las etapas de desarrollo y pruebas dejando a un lado el diseño. El modelo de 4+1, descrito en la Sección 3.3, permite modelar a una aplicación Web por medio de sus vistas. Debido a que las necesidades de un sistema

basado en Web son muy distintas entre sistemas, la correcta definición de estas vistas es crucial para el entendimiento de los requerimientos.

En el siguiente Capítulo proponemos una arquitectura genérica que sirve como base para el diseño de sistemas basados en Web, mediante el análisis de los requerimientos y el uso de los estilos arquitectónicos y patrones de diseño. En cada una de las etapas se define y justifica cada uno de los componentes utilizados en la arquitectura así como las interfaces que utilizan.

Capítulo 4

Arquitectura de software para aplicaciones Web

En los Capítulos anteriores se mencionó que la arquitectura de software nos ayuda a comprender mejor las necesidades que debe de cumplir el producto de software. De la misma forma también se explicó que las aplicaciones Web tienen diferencias con respecto a otro tipo de aplicaciones. En este Capítulo proponemos utilizar el modelo de las 4+1 vistas [14] para describir el modelo arquitectónico de aplicaciones en Web. Este modelo es adaptado para soportar aplicaciones Web agregando una vista de seguridad.

En este Capítulo describimos el proceso requerido para poder desarrollar la arquitectura de software de aplicaciones Web. Así mismo, describimos cada una de las vistas arquitectónicas, los componentes y las relaciones que existen entre estos. Por último, utilizamos los diagramas del Lenguaje de Modelado Unificado (UML) para documentar la arquitectura de software.

4.1. Páginas Web y aplicaciones Web

En 1994 las páginas Web eran simples páginas HTML que contenían ligas a otras páginas. Las páginas Web proveían un fácil y libre acceso a la información a través del mundo. Estas páginas web se conocen como la primero generación de aplicacio-

nes Web. Sin embargo sus funcionalidades eran limitadas ya que consistían de una combinación de imágenes y texto de manera estática.

Con el nacimiento de nuevas tecnologías como los lenguajes de *script* se podía explotar de una mejor manera el potencial de la Web, pero el desarrollo de aplicaciones retrocedió debido al mal uso de estas tecnologías. Las empresas solo se dedicaron a utilizar las nuevas tecnologías para mejorar el aspecto, introducir animaciones, fotos, videos, etc. Sin embargo con el paso del tiempo las empresas se dieron cuenta del gran potencial de las nuevas tecnologías y fue así como comenzó el desarrollo de aplicaciones Web.

Desde entonces se produjo una evolución, ahora las aplicaciones Web son capaces de contener información dinámica. Se comenzó a utilizar de forma más inteligente los conceptos de cliente y servidor. Esto produjo una evolución de los mecanismos para vincular los interfaces con bases de datos y esencialmente hacer todas las aplicaciones accesibles desde un navegador Web. Hoy en día podemos ver aplicaciones Web que aprovechan al máximo los beneficios de la Web.

Como se ha descrito anteriormente, una aplicación Web utiliza una página Web para introducir datos (por ejemplo formularios), enviarlos al servidor, procesar dichos datos (de acuerdo a las políticas del negocio) y mostrar los resultados por medio de otra página Web.

El desarrollo de aplicaciones Web resultó ser de gran ayuda para las organizaciones, pero a la vez requirio la solución de problemas asociados tales como la integración con sistemas legados, la evolución constante y la capacidad de manejo de grandes cantidades de usuarios.

Otros puntos críticos que se tienen que considerar son la seguridad y la presentación como parte de la experiencia de usuario.

Cada aplicación Web tiene distintas necesidades. Por ejemplo, algunos sistemas Web pueden no tener interfaz de usuario porque se comunican con otros sistemas por medio de protocolos. Ejemplo de estos sistemas son los servicios Web. Sin embargo, para otros sistemas el tener una buena interfaz de usuario va a determinar el éxito

a fracaso de la aplicación. Ejemplos de estos sistemas son las tiendas en línea, los periódicos o los sistemas de reservación.

4.2. Patrones de diseño

Como se mencionó en el Capítulo 2, los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y también para otros ámbitos referentes al diseño de interfaces.

El desarrollo de software siempre ha sido comparado con la construcción de edificios. Hemos adoptado ciertas metodologías que nos permiten construir grandes aplicaciones de igual forma que la ingeniería civil lo hace con los grandes edificios. Esto hasta cierto grado es posible debido a que los ambientes son totalmente diferentes. En la Ingeniería Civil se tiene que lidiar con problemas físicos como la tensión y la compresión. Por otro lado, en el desarrollo de software las leyes de la física no son aplicables, sin embargo el equipo de desarrollo tiene que lidiar con nuevas restricciones como son la calidad, la confiabilidad, la portabilidad y el rendimiento.

Lo que nos ha costado entender como desarrolladores de aplicaciones de software es que todas las características anteriores se pueden analizar en etapas tempranas del desarrollo como lo es la etapa de diseño. El diseño es el gran aporte de la ingeniería civil, el cual nos permite contar con una descripción de todo el sistema y saber cuál es el comportamiento del mismo sin necesidad de construirlo. El diseño permite estimar costos, tiempos, detectar errores, dividir tareas y ayuda a hacer las modificaciones necesarias, en el futuro, para mantener un desarrollo en buen estado y mejorar su funcionalidad.

Los patrones de diseño son propuestos por Chistopher Alexander [18] en 1979. En principio los patrones fueron utilizados para la construcción de edificios de mayor calidad y en un menor tiempo y costo. Según Alexander, cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, y provee una solución al mismo. De tal modo que podemos utilizar esta solución muchas veces más adelante

sin tener que volver a pensarla otra vez.

Esta idea fue identificada para sistemas de software en el libro Designs Patterns [19], con la construccion de 23 patrones para el diseño de software.

De acuerdo a lo anterior, un patrón de diseño se describe como una solución a un problema de diseño. Para que una solución sea considerada un patrón debe esta debe de poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Las características de los patrones de diseño son las siguientes:

- *Proporcionar* catálogos de elementos reusables en el diseño de sistemas software.
- *Evitar* la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- *Formalizar* un vocabulario común entre diseñadores.
- *Estandarizar* el modo en que se realiza el diseño.
- *Facilitar* el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Es necesario aclarar que los patrones de diseño no pretenden imponer ciertas alternativas de diseño frente a otras ni eliminar la creatividad inherente al proceso de diseño.

Los patrones abstraen el comportamiento de un determinado problema en una configuración de componentes y estan clasificados de acuerdo con el nivel de abstracción que tenga cada patrón. Los patrones se clasifican en:

- *Patrones de arquitectura*: Son patrones de alto nivel que se utilizan en la definición de la estructura y organización de un sistema de software.

- *Patrones de diseño*: Son patrones de bajo nivel que se utilizan para definir de la mejor manera los componentes de un sistema de software.

- *Patrones de interacción*: Son patrones para el diseño de interfaces de usuario. Su uso es muy común en el diseño de interfaces web.

4.3. Patrones de Arquitectura

Sommerville [4] hace énfasis en que las aplicaciones Web no tienen una definición formal para llevar a cabo su desarrollo. El desarrollo de este tipo de aplicaciones está basado en la experiencia, en el conocimiento del dominio, y en la gente involucrada en el desarrollo de estas aplicaciones. Es por eso que se ha hecho el esfuerzo por documentar los casos de éxito en el desarrollo de este tipo de aplicaciones y se han propuesto patrones arquitectónicos [20], [21], [22] que proporcionan una guía de diseño. A diferencia de un patrón de diseño, un patrón de arquitectura intenta abstraer el comportamiento de un conjunto de componentes, lo cual permite ver el diseño a un nivel más alto (mayor nivel de abstracción)

Un patrón de arquitectura encapsula los elementos y las relaciones que existen entre ellos permitiendo abstraer su comportamiento para que sea posible tener una configuración de componentes que satisfaga ciertas necesidades.

El uso de patrones arquitectónicos en el desarrollo de software ha traído grandes ventajas. Ahora se cuenta con sistemas más robustos y de mayor calidad, los cuales son fáciles de mantener y por esto permiten un desarrollo más rápido. Sin embargo, el uso de patrones hace que el diseño de los sistemas de software sea más complejo, por lo cual se necesita abstraer el comportamiento de un conjunto de componentes y que a su vez este pueda ser utilizado de forma independiente de los demás componentes.

La reusabilidad se vuelve cada vez mas importante, ya que nos permite reducir los tiempos de desarrollo y por lo tanto el costo del software. Las empresas que perfeccionan sus prácticas de reusabilidad obtienen mayor calidad en sus productos. Una vez que se tienen los componentes el reto es ensamblarlos de forma coherente.

Existen una gran variedad de patrones arquitectónicos y cada uno de ellos está pensado para un determinado dominio. Existen patrones para sistemas distribuidos, para sistemas en capas, para sistemas basados en componentes, etc. y cada uno de ellos garantiza resolver un determinado problema utilizando una determinada configuración de los componentes.

El caso que presentamos en este trabajo de tesis se enfoca a las aplicaciones Web, por lo tanto nos limitaremos a mencionar solamente a los patrones de este dominio.

4.3.1. Patrón multicapa

El patrón multicapa descompone una aplicación monocapa en varias capas. El objetivo principal es separar los componentes de acuerdo a su función, por ejemplo en las aplicaciones hay componentes encargados de la presentación, otros de la lógica de negocio y otros de la persistencia de los datos.

En ocasiones se confunde el termino "capas" con el termino "nivel". El primer término se utiliza para referenciar a las distintas "partes" en las que un aplicación se divide desde el punto de vista lógico. Mientras que el segundo término corresponde a la forma física en que una aplicación se organiza. Un ejemplo muy simple (descrito mediante la Figura 4.1) y que es muy común encontrar, es una aplicación que tiene dos niveles (nivel de aplicación y nivel de datos), en donde cada nivel puede tener varias capas. En este caso, el nivel de aplicación puede estar constituido por la capa de presentación y por la capa de lógica de negocio y el nivel de datos puede contener sólo la capa de datos. Como podemos ver en el nivel de aplicación las capas interactúan entre sí por medio de una interfaz.

Esta interfaz permite a la capa de lógica de negocio proveer los recursos que necesita la capa de presentación. Las capas inferiores se encargarán de brindar sus servicios a las capas superiores por medio de sus interfaces.

Al separar una aplicación en capas y niveles permitimos modificar de forma independiente cada capa. De acuerdo al nivel de complejidad de la aplicación se pueden seguir incorporando capas o niveles de acuerdo a las necesidades. En nuestro caso

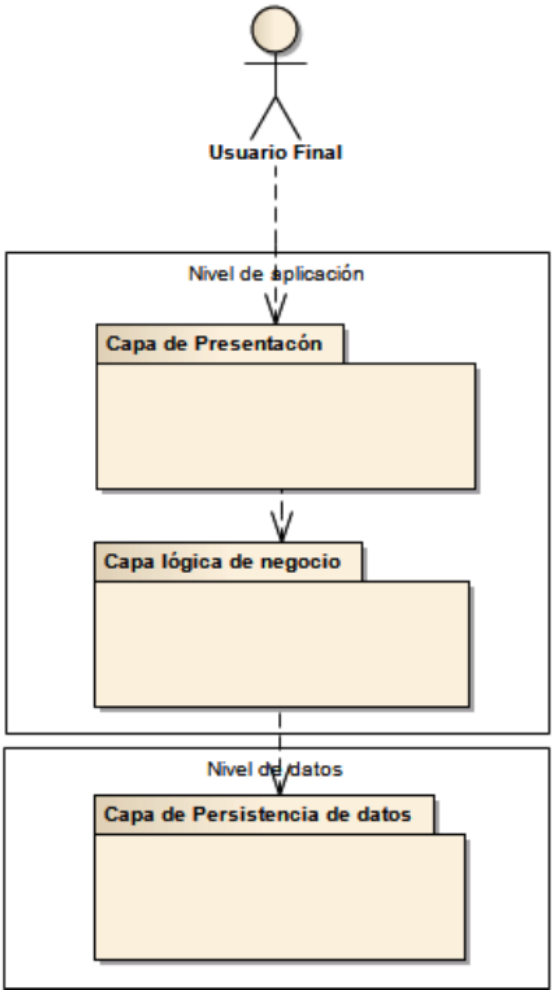

Figura 4.1: Arquitectura dividida en dos niveles y tres capas

requerimos una aplicación que pueda interactuar con diferentes gestores de bases de datos. Para solucionar este problema agregamos otra capa en el nivel de aplicación que se encargue del acceso a los datos (Figura 4.2) independientemente del tipo de gestor de la base de datos.

Si en algún momento se necesita incorporar soporte para otro gestor de base de datos, sólo necesitamos modificar la capa de acceso a datos haciendo transparente los cambios para las otras capas. Para poder agregar otra capa a nuestra arquitectura necesitamos identificar que componentes se pueden aislar de los demás. Este proceso lo podemos seguir hasta obtener una arquitectura que satisfaga las necesidades de la

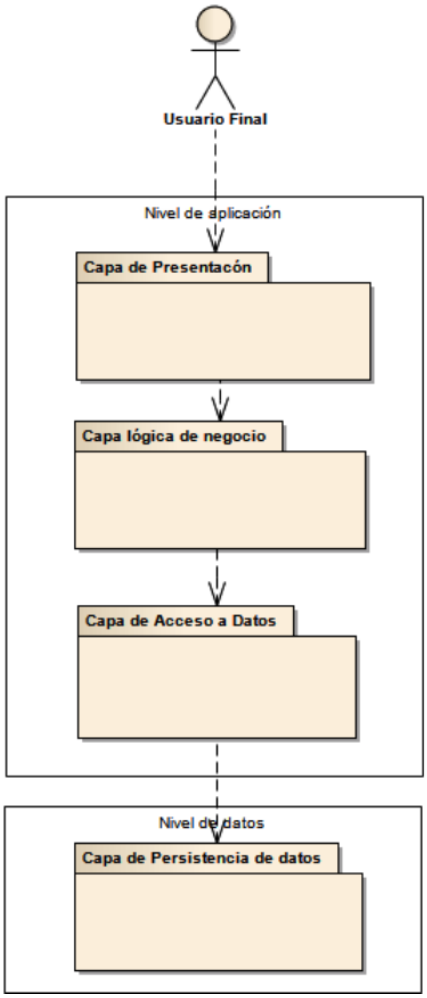

Figura 4.2: Arquitectura dividida en dos niveles y cuatro capas

aplicación. La cual en nuestro caso son las aplicaciones Web.

En la arquitectura descrita en la Figura 4.2 es necesario recalcar que los componentes de la capa de lógica de negocios necesitan referenciar a instancias de las "clases del dominio"(las que representan las entidades del negocio). De la misma forma, los componentes de la capa de acceso a datos también tienen que referenciarlas para poder "rellenar"tales instancias con los datos que obtienen de las capas inferiores. Por tal motivo vemos como una nueva necesidad la capacidad de tener acceso a las clases de dominio desde cualquier capa de la aplicación.

Ahora necesitamos proponer una arquitectura que satisfaga las nuevas necesida-

des, para ello identificamos que las entidades se pueden poner en una capa independiente de las demás.

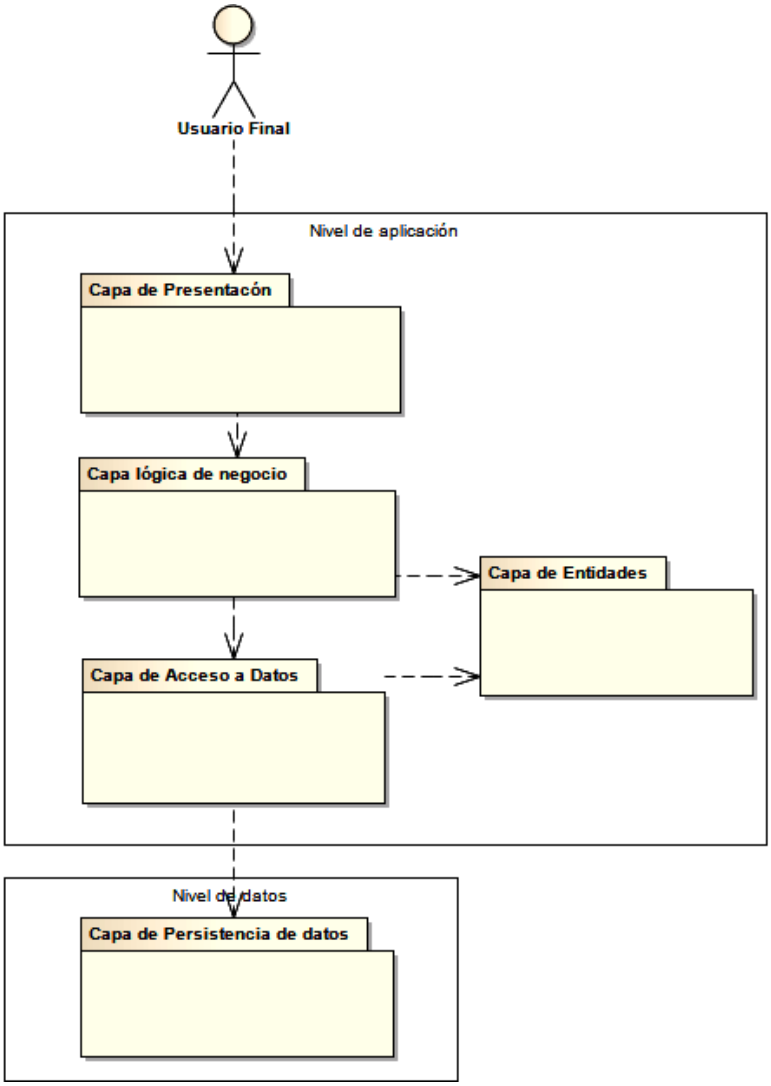

Figura 4.3: Arquitectura dividida en dos niveles y cinco capas

La arquitectura propuesta se muestra en la Figura 4.3, se agrega una nueva capa de entidades que corresponde al dominio de la aplicación. En esta capa se encuentra la declaración de las entidades de la aplicación de manera que se puede tener acceso a ellas desde cualquier otra capa. Ahora tenemos una capa independiente de la capa de la lógica de negocio y de la capa de acceso a datos pero que mantiene las relaciones por medio de sus interfaces.

En la arquitectura cliente-servidor podemos apreciar 2 niveles, (1) cliente y (2) servidor y cada nivel puede tener un número diferente de capas. Las aplicaciones Web son aplicaciones basadas en la arquitectura cliente-servidor y por naturaleza siempre serán aplicaciones con 2 niveles como mínimo, aunque no está limitada a solo 2 niveles.

Recordemos que uno de los principales indicadores del éxito de las aplicaciones Web ha sido la presentación que tienen. En la actualidad existen tecnologías que nos permiten enriquecer las interfaces de presentación de las aplicaciones. Debido a lo importante que resulta la presentación, se necesita adecuar la arquitectura para soportar interfaces ricas y complejas.

La complejidad en el nivel de presentación resulta crucial para el buen rendimiento de la aplicación, la gran mayoría de las aplicaciones necesita hacer un gran número de peticiones al servidor, en ocasiones se necesita mostrar información que no ha sido modificada y, sin embargo se envía una petición para reenviarla. Este problema causa que el tráfico en la red y en el servidor aumente de forma considerable, siendo solo una pequeña parte la información de suma relevancia.

Para atacar estos problemas se necesita dotar a la capa de presentación con la capacidad de decidir qué información se necesita solicitar. Ayudando a enfocar la carga del servidor en peticiones de suma relevancia y aprovechando al máximo el tráfico en la red.

En la Figura 4.4 se muestra el nivel de presentación que se divide en 2 capas, (1) la presentación contiene todos los elementos para presentar de una mejor forma la información y (2) el control de presentación que contiene una pequeña lógica de negocio para determinar los elementos utilizados en la presentación.

El nivel de presentación no necesita de muchos recursos y es independiente de los demás niveles. El nivel de aplicación es el encargado de proporcionar lo necesario para que el nivel de presentación funcione correctamente. A nivel físico podemos incluir el nivel de presentación en la parte del cliente.

Debido a la gran variedad de aplicaciones Web, no podemos proponer una arquitectura genérica que satisfaga las necesidades de toda aplicación Web. Sin embargo

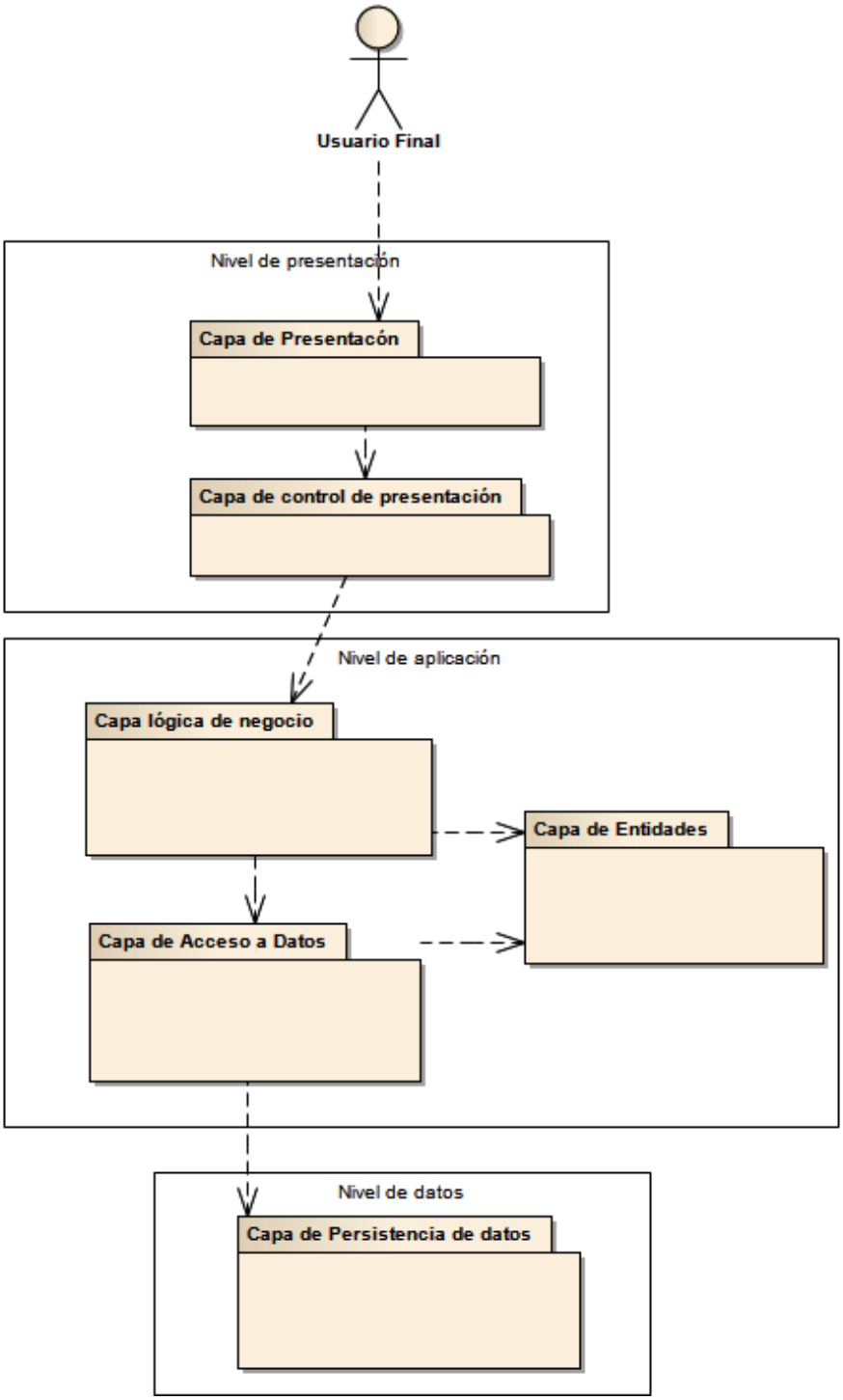

Figura 4.4: Arquitectura dividida en tres niveles y seis capas

haremos una propuesta lo más genérica posible.

4.3.2. Patrón modelo vista controlador

Uno de los patrones de arquitectura que más se utilizan para el desarrollo de aplicaciones Web es el Modelo-Vista-Controlador (MVC) [23]. MVC es un patrón que fue utilizado para construir las interfaces de usuario en el lenguaje Smalltalk-80 [24], siendo la aportación más importante de este patrón la separación de los componentes relacionados con los datos de la aplicación de los componentes de la interfaz de usuario. La separación de las capas permite tener, a nivel de desarrollo, un código más claro, flexible y reusable.

Durante el desarrollo de un patrón multicapa se puede observar que cada capa tiende a encapsular elementos que comparten ciertas características, creando capas que contienen componentes para una determinada función.

El patrón MVC descompone la aplicación en capas permitiendo tener una separación entre la lógica de negocio de la aplicación, la representación y la persistencia. El patrón MVC identifica tres capas que son importantes para cualquier aplicación las cuales son:

- *Modelo* encapsula los datos de la aplicación y la lógica para interactuar con ellos.
- *Vista* maneja la interacción con el usuario y la representación del modelo.
- *Controlador* es el intermediario entre el modelo y la vista ante las peticiones generadas por el cliente en la vista. El controlador se encarga de seleccionar el modelo solicitado por el usuario y la vista adecuada para representarlo.

La separación que propone MVC por medio de las distintas capas se puede observar a nivel de diseño, ayudando a los diseñadores a identificar los componentes de cada capa y la comunicación que existe con los demás componentes.

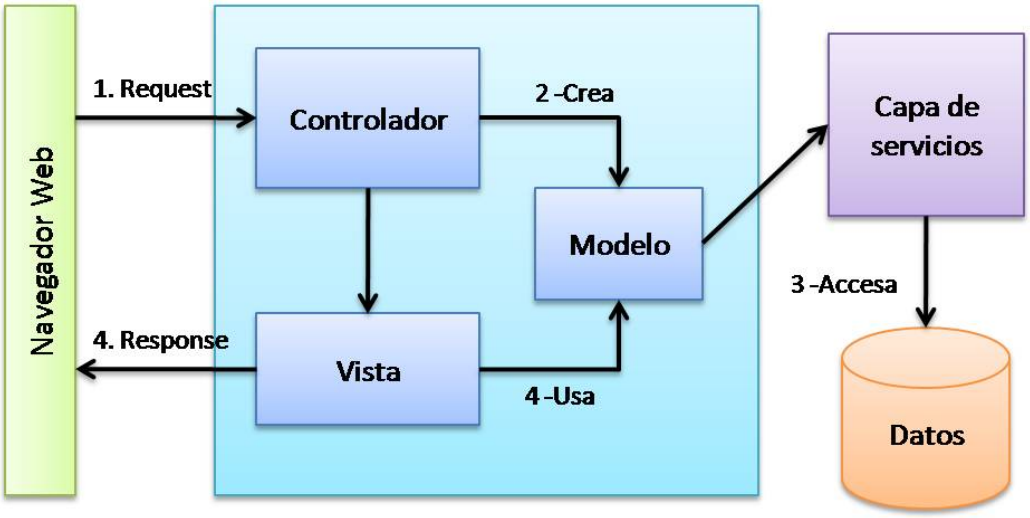

Figura 4.5: Patrón Modelo-Vista-Controlador

En la Figura 4.5 se puede apreciar una arquitectura simple para una aplicación Web utilizando el patrón MVC como pilar para la definición de la arquitectura. En esta arquitectura se puede ver como se utiliza MVC para interactuar con componentes ya definidos tales como el navegador Web y las bases de datos.

Al separar la presentación, los datos y la lógica de negocio se tiene una idea más clara de lo que necesita la aplicación, se desarrollan los componentes y se establecen las relaciones necesarias. La utilización de MVC permite tener menor acoplamiento, modificando solo las partes involucradas (se modifica solo lo que se necesita), siendo transparente para las demás.

Al utilizar MVC tenemos la capacidad de representar la información de varias formas sin necesidad de modificar la fuente, en la Figura 4.6 se muestra a un modelo que puede tener varias vistas. En las aplicaciones Web nos permite crear interfaces personalizadas sin necesidad de hacer un cambio mayor.

El gran nivel de abstracción del patrón MVC ha permitido desarrollar exitosamente complejas aplicaciones Web. Para el desarrollo de aplicaciones Web no resulta muy obvia la aplicación de este patrón (recordemos que la interfaz por defecto de una aplicación Web es el navegador Web). Sin embargo, necesitamos considerar el diseño

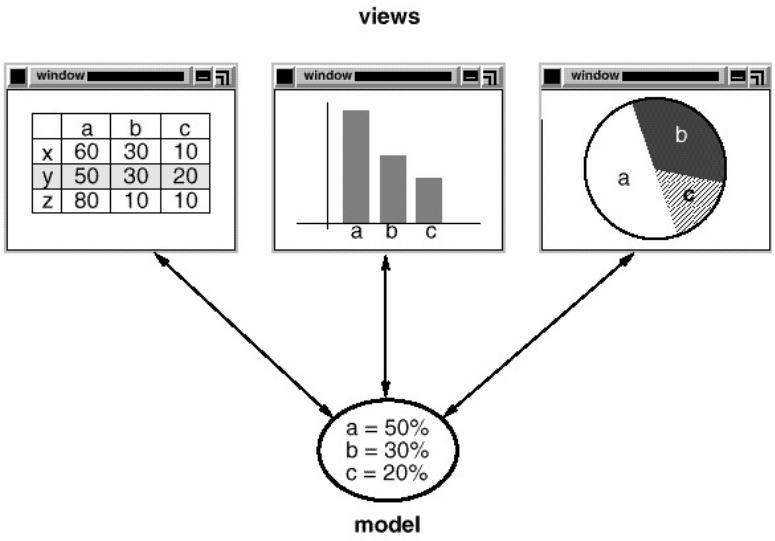

Figura 4.6: Varias vistas para un sólo controlador

de los dos componentes. Esta tarea, en ocasiones, resulta ser muy complicada debido a que entre el cliente y servidor puede existir una gran cantidad de componentes. Supongamos que tenemos una aplicación en donde la persistencia de los datos es muy compleja y que se necesita tener varios servidores en diferentes partes de la organización. En estos casos podemos tener otro subsistema encargado solo de la persistencia, esto implica que cada componente puede tener varios componentes internos.

4.4. Arquitectura genérica para aplicaciones Web

En esta sección propondremos una arquitectura genérica que pueda utilizarse para modelar la mayoría de las aplicaciones Web existentes. Esta arquitectura genérica contiene todos los componentes necesarios para el desarrollo de aplicaciones del dominio Web. A diferencia de las arquitecturas de aplicaciones locales, esta arquitectura debe contener componentes basados en Web que permiten aprovechar al máximo la infraestructura de la Web obteniendo aplicaciones robustas y de gran desempeño, con alcance global.

Para poder definir nuestra arquitectura genérica, lo primero que necesitamos hacer

es identificar los componentes necesarios para las aplicaciones Web.

- **Cliente-Servidor.**

  Lo que podemos observar en primera instancia es que una aplicación Web sigue un modelo cliente-servidor. Esto se debe a que se requiere un componente que hospede a la aplicación y a todos sus componentes (servidor Web) y un componente cliente, en nuestro caso un navegador Web. Una vez que tenemos estos dos componentes necesitamos un sistema de archivos en donde se van a almacenar todos los recursos para la generación de contenido. El sistema de archivos puede almacenar una gran cantidad archivos multimedia (fotos, videos, audio, etc.), archivos XML y HTML, necesarios para el funcionamiento de la aplicación.

  La primera propuesta de arquitectura para aplicaciones Web se puede observar en la Figura 4.7.

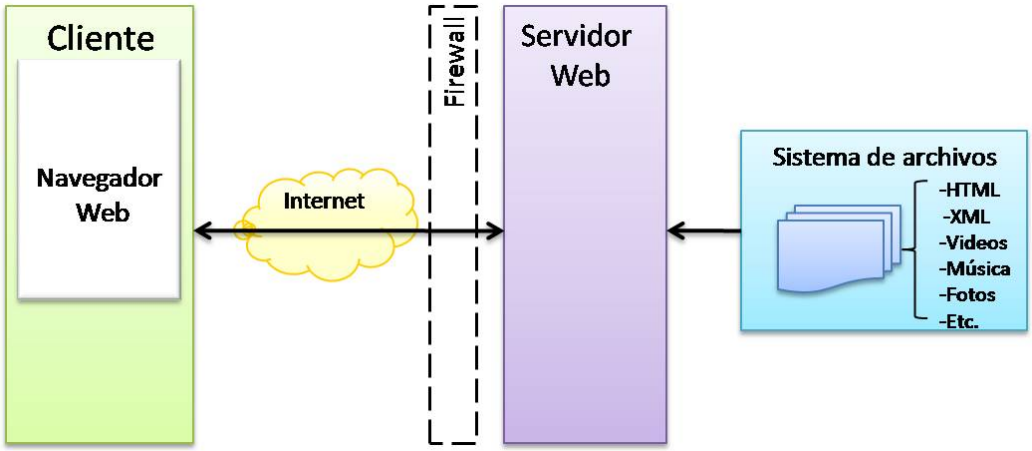

Figura 4.7: Arquitectura para aplicaciones Web estáticas

- **Servidor de aplicaciones.**

  La arquitectura propuesta en la Figura 4.7 satisface los requerimientos de aplicaciones web estáticas que basan su contenido en archivos HTML estáticos. Sin embargo, para que podamos modelar aplicaciones Web dinámicas necesitamos agregar componentes adicionales tales como un servidor de aplicaciones para

permitir la generación contenidos dinámicos tomando como base la arquitectura que estamos proponiendo. Al agregar este nuevo componente necesitamos considerar las interfaces que se necesitan incluir así como la relación con los componentes existentes.

La Figura 4.8 muestra los componentes y las interfaces que se han agregado para complementar la arquitectura y poder soportar las demandas de las aplicaciones Web con contenido dinámico.

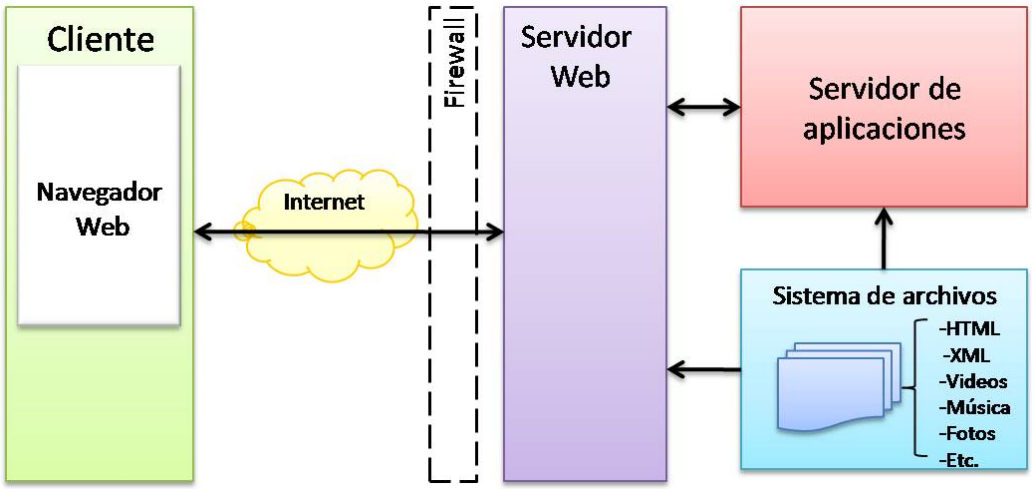

Figura 4.8: Arquitectura para aplicaciones Web dinámicas

- **Servidor de Bases de Datos.**

  Hasta ahora, con la arquitectura desarrollada podemos generar aplicaciones Web con contenido dinámico utilizando plantillas, archivos multimedia y recursos estáticos. Sin embargo, apegándonos a las necesidades de las aplicaciones Web que se desarrollan actualmente, necesitamos considerar la gran cantidad de datos que se necesitan manejar. Considerando esta necesidad agregamos a nuestra arquitectura un servidor de base de datos. Este componente nos permitirá almacenar toda la información que la aplicación necesita.

- **Interfaz con sistemas externos.**

Otra necesidad actual de los sistemas en Web es la comunicación con sistemas externos. La mayoría de los sistemas Web necesitan comunicarse con otro tipo de sistemas. Por ejemplo, la aplicación Web de una tienda electrónica necesita comunicarse con el sistema de inventarios para verificar la existencia de un producto. Para solucionar este problema se agrego un componente que tiene como objetivo ser la interfaz con los sistemas externos.

En la Figura 4.9 se puede observar la evolución de nuestra arquitectura propuesta.

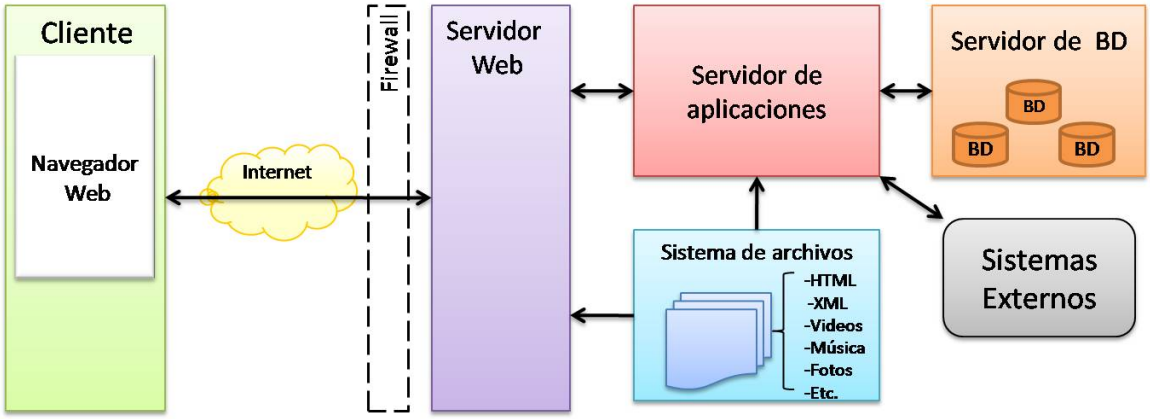

Figura 4.9: Arquitectura para aplicaciones Web con soporte de grandes cantidades de datos

La arquitectura presentada en la Figura 4.9 nos permite desarrollar aplicaciones Web muchos más robustas en comparación con las primeras arquitecturas. Hasta el momento consideramos que la arquitectura puede soportar aplicaciones Web que necesitan ser escalables.

Además de los componentes anteriormente añadidos, incluiremos otros que nos permitirán reforzar la arquitectura propuesta.

- **Servidor Multimedia.**

  Una de las características que consideramos importante es el manejo de contenidos multimedia en las aplicaciones Web. Con el manejo de interfaces ricas

y una mejora en la presentación de las aplicaciones, se ha generado una enorme demanda de recursos multimedia. Esta necesidad de las aplicaciones se ha convertido en un reto, debido a que demandan el manejo de grandes cantidades de recursos multimedia sin presentar degradación en el rendimiento de la aplicación.

Nuestra propuesta (descrita en la Figura 4.9) ha consistido en añadir a la arquitectura un servidor multimedia que permita a las aplicaciones basadas en Web manipular grandes cantidades de datos y de recursos multimedia.

- **Servidor de Componentes.**

  Por otro lado, existen aplicaciones Web que necesitan permitir el acceso de terceros a sus componentes Web de forma segura. La razón para permitir el acceso a los componentes se debe a que muchas aplicaciones Web utilizan componentes desarrollados por terceros.

  Un Ejemplo muy común de este tipo de aplicaciones son los servicios de pronósticos del clima. Una aplicación central es la encargada de obtener dichos datos y después distribuirla a las demás aplicaciones por medio de un componente. Otro ejemplo es el mercado de divisas, en donde solo la aplicación del banco central tiene dicha información que posteriormente será distribuida a los demás por medio del acceso a dicho componente.

  La forma que hemos considerado más adecuada para cumplir con la necesidad de brindar acceso a los componentes, es por medio de un servidor de componentes que se encargara de hospedar solo los componentes que son utilizados por un tercero.

  En la Figura 4.10 se muestran los nuevos componentes que fueron agregados y las relaciones que existen entre ellos. La relación del servidor multimedia va a ser directamente con el servidor Web. De la misma forma, el servidor de componentes va tener una relación directa con el servidor Web, debido a que el servidor Web es el que va a solicitar los recursos para enviarlos al cliente.

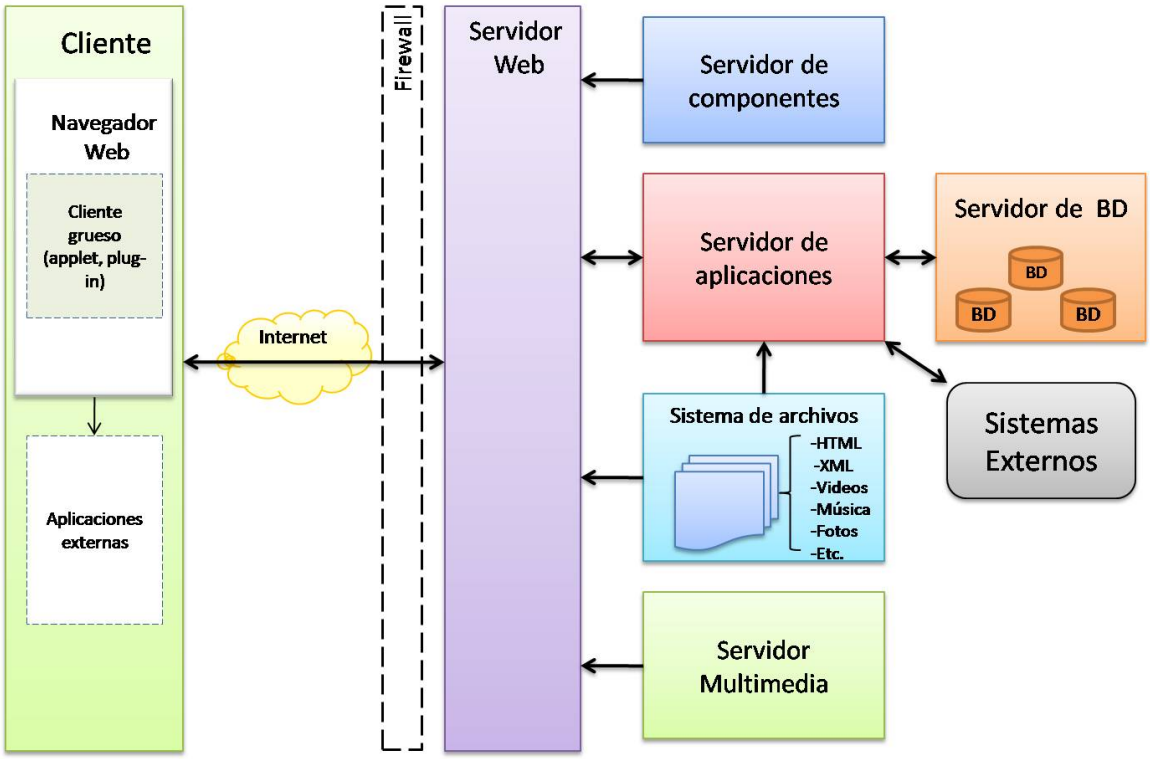

Figura 4.10: Arquitectura Genérica para aplicaciones Web

- **Componentes del Cliente.**

  La arquitectura de la Figura 4.10 se puede considerar una arquitectura completa para este tipo de aplicaciones, debido a que cuenta con varios componentes que permiten desarrollar una gran variedad de aplicaciones Web.

  Del lado del cliente se agrego un modelo de cliente grueso, esto ayudara al servidor de a liberar un poco de carga de trabajo al momento de trabajar con aplicaciones ricas en Internet (Rich Internet Applications) y una interfaz para trabajar con aplicaciones externas (controladores para cámaras Web, micrófonos, aplicaciones multimedia).

  Agregando estos componentes de parte del cliente podemos tener una mejor experiencia de usuario sin que esto se vea reflejado en el desempeño del servidor. Por la parte del servidor también se han agregado otros componentes con la intención de aligerar la carga de trabajo para el servidor Web y de aplicaciones.

Muchos desarrolladores creen que una aplicación se debe de adaptar a la arquitectura, cuando la realidad es la arquitectura la que se tiene que ir adaptando a las necesidades de cada aplicación. La arquitectura que se ha propuesto en un inicio, ha ido evolucionando para poder satisfacer los requerimientos genéricos de las aplicaciones Web tradicionales.

La Figura 4.11 muestra la arquitectura genérica final como resultado del proceso de diseño arquitectural. En esta arquitectura se muestra cada uno de los componentes que existen dentro de cada servidor y cuál es la relación que existe entre estos

Como se puede observar se han hecho cambios en ambas capas (cliente y servidor) tratando de soportar las demandas de una aplicación Web robusta.

Los servidores que se incorporan a esta arquitectura tienen como objetivo enfocarse en tareas específicas como se describe a continuación:

- **Proxy**: es el encargado de trasmitir las peticiones de los clientes al servidor Web pero con una característica particular, tener un registro de todos los recursos que se han solicitado recientemente. Cuando un recurso es muy solicitado en una aplicación Web, el proxy ayudará a responder a la petición con mayor rapidez. Por ejemplo, si nuestra aplicación tiene una página para la autenticación de usuarios, el proxy guardará esta página para la próxima vez que sea solicitada disminuyendo el tiempo de espera. En dado caso que tal recurso no exista en el proxy la petición pasará al servidor Web quien sabrá que hacer para atender petición.

- **Servidor Web**: es el encargado de recibir todas las peticiones de la aplicación, y es el componente que se encargará de atender la petición y enviarla en formato HTML por medio de los demás componentes. Como vemos, es el componente que mantiene una comunicación con la mayoría de los componentes.

- **Servidor de Aplicaciones**: es el encargado de hospedar a la aplicación y de proporcionar lo necesario para que la aplicación pueda funcionar de forma correcta. Este servidor es el encargado de transformar la petición proveniente

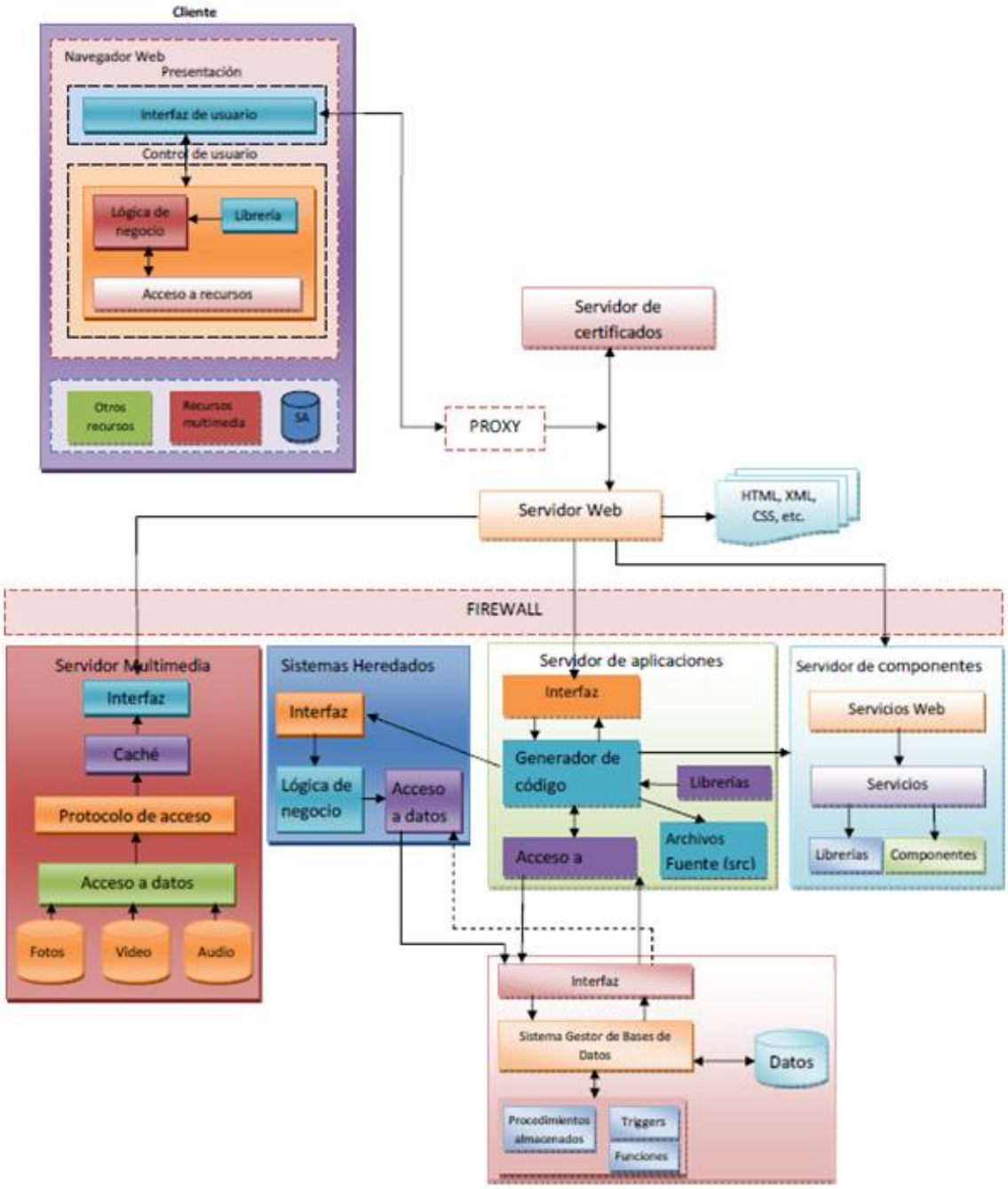

Figura 4.11: Arquitectura Genérica para aplicaciones Web

del servidor Web, y transformarla en un recurso (archivo HTML) utilizando todas las reglas de negocio establecidas.

La relación que existe entre el servidor de aplicaciones y el servidor de base de datos se debe a la gran demanda de recursos que existe entre ellos.

- **Servidor de Base de Datos**: es el encargado de proporcionar la persistencia de los datos de la aplicación por medio de un Sistema Gestor de Base de Datos (SGBD). Este servidor se encargará de almacenar cualquier dato que la aplicación necesite. A veces se confunde un servidor de Base de Datos con un SGBD. Recordemos que la principal diferencia es que el servidor es un componente que provee todos los recursos para el almacenamiento de los datos (discos duros, conexiones de banda ancha) y el SGBD es el software para administrar Bases de datos relacionales. Un Servidor de Base de Datos puede tener múltiples SGBD.

- **Servidor Multimedia**: es el encargado de almacenar todos los recursos multimedia de la aplicación. A diferencia del Servidor de Base de Datos el servidor Multimedia no utiliza un SGBD para la administración de los recursos debido al tamaño de los mismos. Los recursos son almacenados en un sistema de archivos lo que permite su fácil localización y un mejor desempeño.

- **Servidor de componentes**: es el encargado de proporcionar todas las herramientas para la publicación de componentes. Los servidores de componentes permiten el acceso a los recursos de una aplicación Web por medio de estándares bien definidos tales como XML.

Regresando a la arquitectura anterior, se puede observar que el servidor Web era el encargado de realizar cualquier tarea, lo cual aumentaba la carga de trabajo y por la tanto la probabilidad de que la aplicación fallara.

Con la adición de estos servidores, la tarea fundamental del servidor Web será ahora de ser el orquestador de los demás componentes. Cada que se recibe una petición, será por medio de este servidor el cual su vez direccionará la petición al componente indicado, reduciendo notablemente su carga de trabajo.

Sin embargo, existen otros temas que la arquitectura debe abordar debido a diferencia entre los mecanismos de comunicación que unen a los componentes.

Un problema que se considero en la arquitectura propuesta fue la falta de estado del protocolo HTTP. Se sabe que desde el navegador Web al servidor Web la comunicación no tiene estado. Para poder acceder a un recurso Web se necesita hacer una petición al servidor Web, una vez que se ha entregado el recurso la conexión termina. Existen varias técnicas para resolver este tema que consisten en el uso de Cookies o la comunicación a través de IIOP (Internet Inter-ORB Protocol). En nuestra arquitectura existen componentes (servidor de componentes, acceso a recursos de parte del cliente) que pueden ayudar a resolver este problema.

Otro desafío arquitectónico es la integración de la lógica de negocio de la aplicación. Sabemos que la lógica de negocio puede estar en la parte del servidor (cliente delgado) o en la parte del cliente (cliente pesado / cliente ligero). Cada modelo tiene sus ventajas y desventajas.

El enfoque de cliente ligero permite tener un mayor control de la lógica de negocio, brinda mayor seguridad, y no existen problemas de distribución (plug-ins). Por otro lado las aplicaciones carecen de interfaces ricas en la presentación. Un ejemplo de aplicaciones basadas en este enfoque serian las aplicaciones de banca en línea.

En contraste, las aplicaciones con enfoque de cliente grueso intentan trasladar una parte de la lógica de negocio al cliente. Esto permite tener aplicaciones más interactivas pero menos seguras, en donde la carga del servidor disminuye con el consecuente aumente en el tiempo de distribución. Un ejemplo de aplicaciones basadas en este enfoque serian los portales multimedia (videos, fotos, música).

Hay ocasiones en donde los requerimientos de una aplicación con respecto de otra van a tener sentidos opuestos. Mientras una aplicación necesite mayor seguridad otra podría no necesitarlo, cuando una necesité tener una mayor interacción con el usuario la otra podría limitarse a tener una interfaz básica.

Al final necesitamos evaluar constantemente los requerimientos de nuestra aplicación para evitar elegir componentes equivocados que puedan ocasionar problemas a

futuro.

La propuesta de esta arquitectura permite sentar las bases para que cualquier aplicación Web pueda funcionar correctamente. La arquitectura queda abierta a la agregación de nuevos componentes que sean necesarios para satisfacer las necesidades de la aplicación que se desea desarrollar.

4.5. Lenguaje de modelado unificado (UML)

Durante la propuesta de la arquitectura se observó que uno de los principales problemas en el desarrollo de software es la especificación de la arquitectura. Para resolver este problema en [25, 26, 27] se han propuesto lenguajes de descripción de arquitecturas (ADL's). Sin embargo no se ha resuelto del todo el problema debido a que no existe un estándar para este tipo de leguajes.

El problema que ahora surge es cómo representar la arquitectura de un sistema de software, de tal manera que sea entendible por la mayoría de las personas que están involucradas en el desarrollo. Los ADL's son lenguajes que permiten describir la arquitectura, son fáciles de comprender para los arquitectos o diseñadores pero resultan complicados para el resto de los miembros del equipo. Para hacer frente a este problema se propone el uso del lenguaje de modelado unificado (UML) para la representación de las arquitecturas. UML es un lenguaje estándar que nos permite modelar los componentes de un sistema de forma que sea entendible [1, 28] para todos los integrantes del equipo de desarrollo.

En la Sección 3.3 se analizo el trabajo de Philippe Kruchten sobre el modelo de 4+1 vistas (the 4+1 model view) para describir la arquitectura de software. Este enfoque utiliza diferentes vistas para separar los conceptos de cada *Stakeholder*. El enfoque de 4+1 vistas ha sido ampliamente por la comunidad de la industria de software para representar el diseño de la arquitectura de software de las aplicaciones.

En la Figura 4.12 se muestra los diagramas de UML que corresponden a cada vista del modelo de 4+1 vistas.

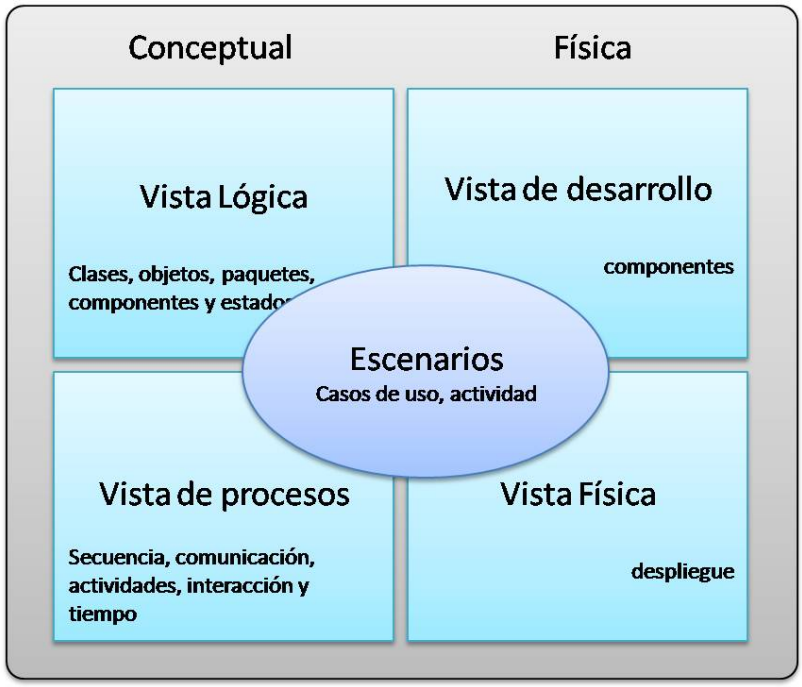

Figura 4.12: Modelo de 4+1 vistas con UML

UML es un lenguaje de modelado que utiliza conceptos orientados a objetos y tiene una sintaxis y semántica bien definidas lo que nos permite utilizarlo en todas las etapas de desarrollo y no solo en el proceso de diseño arquitectónico [29].

De acuerdo con la especificación de UML 2 se encuentra divido en 13 tipos de diagramas básicos dentro de dos categorías clave:

1. *Diagramas estructurales*: estos diagramas son utilizados para definir la arquitectura estática. Estos comprenden la construcción estática como las clases, objetos y componentes, y la relación entre estos elementos. Existen seis diagramas estructurales: Diagrama de paquetes, diagrama de clases, diagrama de objetos, diagrama de componentes, diagrama de despliegue y diagrama de estructura compuesta.

2. *Diagramas de comportamiento*: estos diagramas son usados para representar la arquitectura dinámica. Estos comprenden la construcción dinámica como actividades, estados, líneas de tiempo, y los mensajes que ocurren entre diferentes

objetos. Estos diagramas son usados para representar la interacción entre varios elementos del modelo y estados instantáneos sobre un periodo de tiempo.

Existen siete diagramas de comportamiento: Diagramas de casos de uso, diagramas de actividades, diagramas de estados, diagramas de comunicación, diagramas de secuencia, diagramas de tiempo y diagramas de interacción.

La organización fundamental de un sistema de software puede ser representada por:

- *Elementos* estructurales y sus interfaces que comprenden o forman un sistema.
- *Comportamiento* representado por la colaboración entre los elementos estructurales.
- *Composición* de elementos estructurales y de comportamiento dentro de los grandes sistemas.

Tales composiciones son guiadas por las capacidades deseadas (requerimientos no funcionales) como la usabilidad, resistencia, rendimiento, reusabilidad, limitaciones económicas y tecnológicas etc. Además existen temas transversales (como la seguridad y el manejo de transacciones) que aplican para todos los elementos funcionales.

Cada uno de los elementos y relaciones tiene una representación gráfica que, a su vez, se puede complementar con la especificación del elemento o relación; la especificación no es visible del todo ya que corresponde a los datos o propiedades adicionales que complementan la semántica del elemento o relación.

La arquitectura puede tener diferentes significados para cada stakeholder. Por ejemplo un ingeniero de redes podría estar únicamente interesado en el hardware y la configuración de la red que utiliza el sistema; un administrador de proyecto en los componentes clave a desarrollar y sus líneas de tiempo; un desarrollador en clases que conforman un componente; y un tester en escenarios para localizar los posibles errores del sistema.

Por lo tanto necesitamos múltiples puntos de vista para las distintas necesidades de los *actores* involucrados en el desarrollo de un sistema de software.

4.6. Resumen

En este Capítulo presentamos una arquitectura de software para aplicaciones basadas en Web. La arquitectura es el resultado de un estudio de estilos arquitectónicos, patrones de arquitectura y patrones de diseño, tomando como base el modelo de cliente-servidor para el desarrollo de esta arquitectura.

Le proceso de diseño arquitectónico comenzó con el modelo cliente-servidor, posteriormente se utilizo el patrón diseño multicapa para definir las capas de la arquitectura a nivel lógico. Por último se utilizo el patrón Modelo Vista Controlador para separar correctamente las capas de la presentación, la lógica de negocio y datos. En cada etapa se valido que la arquitectura fuera apegándose a las necesidades del domino Web, que permitió identificar y corregir los errores de diseño.

Se describieron los diagramas de UML que ayudaran a modelar la arquitectura en un lenguaje que permita una fácil interpretación

El siguiente Capítulo presentamos un caso de estudio en donde se resalta la especificación de software y diseño. Se describen todos los requerimientos funcionales y no funcionales que posteriormente se verán reflejados en cada una de las vistas arquitectónicas.

Capítulo 5

Caso de estudio

Para poder evaluar la arquitectura propuesta en el Capítulo 4 necesitamos diseñar una aplicación Web para observar el papel de la arquitectura. Dado que la arquitectura se encuentra en la etapa de diseño centraremos todos los esfuerzos en explicar con gran detalle la arquitectura de una aplicación Web real.

5.1. El desarrollo de aplicaciones Web

Se conoce que el desarrollo de aplicaciones Web es diferente debido a que los requerimientos (funcionales y no funcionales) y el dominio de estas aplicaciones se enfocan en el dominio Web. Para el desarrollo de aplicaciones Web se requiere definir claramente los alcances de la aplicación y considerar todos los posibles problemas a lo que se puede enfrentar. Esta tarea resulta complicada debido a que cada persona involucrada en el desarrollo puede tener una idea distinta sobre la funcionalidad de la aplicación.

En los inicios de Internet el desarrollo de aplicaciones Web era sencillo debido a que eran páginas Web estáticas sin ningún comportamiento o dinamismo. Sin embargo al momento que las aplicaciones son dotadas de comportamientos (paginas personalizadas) estas comienzan a ser más complicadas de desarrollar y ahora necesitan la definición de una capa encargada de las reglas del negocio (lógica de negocio). La

capa de lógica de negocio permite a una aplicación comportarse de diferentes formas ante las peticiones de los usuarios.

Las aplicaciones web han crecido de tal forma que es necesaria una metodología que soporte la fabricación de grandes sistemas de software tomando en cuenta las restricciones del dominio de a aplicación. En el desarrollo de aplicaciones Web necesitamos tener claro el ciclo de vida del software para definir, comprender y acotar cada una de las etapas de este ciclo.

Para el desarrollo de aplicaciones Web no basta el enfoque de cliente-servidor (como se menciono en el capítulo 4) debido a la naturaleza de la aplicación. Por tal motivo se necesita combinar las técnicas de desarrollo de software existentes con las técnicas de desarrollo Web.

5.2. Caso de estudio: Sistema de Administración Escolar (SAE)

Las aplicaciones Web han sido una herramienta clave para el desarrollo de las organizaciones. Hoy en día las aplicaciones Web son un pilar fundamental en la administración de grandes cantidades de usuarios y por consiguiente de datos.

Las organizaciones educativas son un ejemplo de comunidades de usuarios que necesitan el acceso a la información continua de sus actividades escolares. Además de tener una gran cantidad de alumnos, sabemos que se tiene una gran diversidad de empleados (directores, profesores, secretarias, jardineros, etc.).

El caso de estudio planteado en esta tesis consiste en desarrollar la arquitectura para una aplicación Web para la administración, SAE (Sistema de Administración Escolar). Para este caso de estudio seguiremos un enfoque en cascada enfocando la atención en las etapas de diseño. La decisión de usar el enfoque en cascada se debe a que permite ver terminada cada etapa del ciclo de vida del software antes de comenzar con la siguiente etapa.

Para complementar el enfoque en cascada utilizaremos el modelo de 4+1 vistas,

descrito en el capítulo 4, y los distintos diagramas de UML para documentar la arquitectura.

5.3. Obtención de requerimientos

La primera etapa para la construcción de un sistema de software es la obtención de requerimientos. Los requerimientos se dividen en dos grupos (1) los requerimientos funcionales y (2) los requerimientos no funcionales. En esta etapa se muestran cada una de las necesidades que se han detectado, de cada uno de los usuarios involucrados en el sistema.

En esta sección se enlistan cada uno de los requerimientos del SAE de forma natural, como el cliente lo expreso, para después mostrarlo formalmente. Cada uno de estos requerimientos es analizado y expresados de forma técnica y en términos del sistema, para conocer cuáles son las características que el sistema debe de tener.

Así mismo, en esta sección se analizan los diferentes escenarios que el sistema puede tener, así como el comportamiento del sistema ante un determinado escenario. Para la documentación de los requerimientos utilizaremos los diagramas de caso de uso de UML.

5.3.1. Requerimientos del usuario

En la Tabla 5.1 se detallan cada uno de los requerimientos del usuario del SAE. Dichos requerimientos son expresados en lenguaje natural, de forma que puedan ser comprendidos fácilmente por los clientes o usuarios del sistema.

5.3.2. Requerimientos del sistema

En esta sección se describirá mediante tablas cada uno de los servicios y restricciones que el sistema debe de cumplir para satisfacer los requerimientos del usuario del SAE. Esta sección está orientada al personal encargado de su desarrollo (programadores, líderes de proyecto y desarrolladores de software).

#	Requerimientos de usuario
1	El *sistema de Administración Escolar (SAE)* debe de autentificar a cada uno de los usuarios (alumno, coordinador y personal administrativo), mediante su *Login* y *password*.
2	El SAE debe permitir el uso de los servicios del sistema de acuerdo a los permisos de cada uno de los usuarios.
3	El SAE permitirá el alta, baja y modificación de cursos, solo para los usuarios autorizados (coordinadores académicos y personal administrativo).
4	El SAE permitirá el alta, baja y modificación de los usuarios del SAE.
5	El SAE permitirá, a los usuarios registrados, la consulta detallada (horario, cupo y profesor) de los cursos disponibles.
6	El SAE permitirá la asignación y modificación de profesores a cada uno de los a cursos.
7	El SAE permitirá a los alumnos inscritos la inscripción y la baja de cursos
9	El SAE deberá contar con interfaces amigables e intuitivas para facilitar el uso.
10	El SAE deberá garantizar la disponibilidad e integridad de la información en todo momento así como el acceso a ella desde cualquier parte del mundo.

Tabla 5.1: Requerimientos del usuario del SAE

En las siguientes Tablas, se identificarán los requerimientos del sistema que corresponden a cada requerimiento del usuarios. Todos los requerimientos están descritos en leguaje natural de acuerdo al método descrito por Sommerville [4].

Requerimientos del usuario	
1	El SAE debe autentificar a cada uno de los usuarios (alumno, coordinador y personal administrativo), mediante su Login y password.
Requerimientos del sistema	
1.1	El SAE deberá de contar con una base de datos de usuarios donde se almacenen los usuarios que tienen derecho al uso del sistema, así como una pequeña descripción acerca de ellos y los privilegios con los que cuenta.
1.2	El SAE debe incluir una interfaz de identificación, para que el usuario introduzca sus datos
1.3	El SAE debe verificar que el Login y password sean validos comparándolos con los existentes en la base de datos de usuarios.
1.4	De acuerdo con el tipo de usuario (alumno, coordinador y personal administrativo) el SAE deberá permitirle el acceso a los diferentes servicios del sistema.
1.5	En caso de que el usuario se equivoque se deberá mostrar un cuadro de dialogo que indique en donde ocurrió el error.

Tabla 5.2: Requerimientos del sistema correspondiente al requerimiento de usuario 1

Requerimientos del usuario	
2	El SAE debe permitir el uso de los servicios del sistema de acuerdo a los permisos de cada uno de los usuarios.

Requerimientos del sistema	
2.1	El SAE debe mostrar un menú con todos los servicios disponibles para el usuario.
2.2	El SAE debe restringir al usuario el acceso no autorizado a los servicios de acuerdo al tipo de usuario

Tabla 5.3: Requerimientos del sistema correspondiente al requerimiento de usuario 2

Requerimientos del usuario	
3	El SAE permitirá el alta, baja y modificación de cursos, solo para los usuarios autorizados (coordinadores académicos y personal administrativo).
Requerimientos del sistema	
3.1	Si se trata de un curso nuevo, el SAE proporcionará una interfaz con los campos necesarios para su registro.
3.2	El SAE guardará en la base de datos los nuevos registros asociados con el curso.
3.3	Si se trata de una baja de curso, el SAE proporcionará una interfaz en donde el usuario proporcionará el código del curso que desea dar de baja.
3.4	El SAE validará que el código pertenezca a un curso existente.
3.5	El SAE verificará que se cumplan los requisitos necesarios para poder dar de baja al curso. En caso de que no sea así el SAE debe de notificar al usuario las causas que impiden el proceso.
3.6	El SAE eliminará los registros de la base de datos relacionados con el curso.

3.7	Si se trata de una modificación de la información de un curso, el SAE deberá proporcionar una interfaz en donde el usuario pueda acceder a los datos del curso.
3.8	El SAE verificará que los datos nuevos son correctos, en caso contrario se mostrará un aviso al usuario con los datos incorrectos.
3.9	El SAE actualizará los registros de la base de datos con la información proporcionada por el usuario.

Tabla 5.4: Requerimientos del sistema correspondiente al requerimiento de usuario 3

Requerimientos del usuario	
4	El SAE permitirá el alta, baja y modificación de los usuarios del sistema.
Requerimientos del sistema	
4.1	Si se trata del registro de un nuevo usuario, el SAE proporcionará una interfaz con los campos necesarios para su registro.
4.2	El SAE guardará en la base de datos los nuevos registros asociados con el usuario.
4.3	Si se trata de dar de baja a un usuario, el SAE proporcionará una interfaz para capturar los datos del usuario a dar de baja.
4.4	El SAE validará que el identificador de usuario pertenezca a un usuario registrado.
4.5	El SAE verificará que se cumplan los requisitos necesarios para poder dar de baja al usuario. En caso de que no sea así el SAE debe de notificar cuales son las causas que impiden el proceso.

4.6	El SAE eliminará los registros de la base de datos relacionados con el usuario.
4.7	Si se trata de una modificación de la información de un usuario, el SAE deberá proporcionar una interfaz en la que se pueda acceder a la información del usuario, brindando la opción de modificación de dichos datos.
4.8	El SAE verificará que los nuevos datos son correctos, en caso contrario se mostrará un aviso al usuario con los datos incorrectos.
4.9	El SAE actualizará los registros de la base de datos con la información proporcionada por el usuario.

Tabla 5.5: Requerimientos del sistema correspondiente al requerimiento de usuario 4

Requerimientos del usuario	
5	El SAE permitirá, a los usuarios registrados, la consulta detallada (horario, cupo, profesor, etc.) de los cursos disponibles.
Requerimientos del sistema	
5.1	El SAE mostrará un catalogo de todos los cursos disponibles en el cuatrimestre actual.
5.2	El SAE mostrará a detalle cada uno de los cursos que se encuentran en el catálogo.

Tabla 5.6: Requerimientos del sistema correspondiente al requerimiento de usuario 5

Requerimientos del usuario	
6	El SAE permitirá la asignación y modificación de profesores de cada uno de los cursos
Requerimientos del sistema	
6.1	Si se trata de una asignación, el SAE debe proporcionar una lista con todos los cursos que se imparten y que no cuentan con profesor.
6.2	El SAE deberá proporcionar una lista con todos los profesores disponibles para impartir el curso.
6.3	El SAE verificará que la asignación sea válida. En caso contrario, el SAE mostrará un aviso que indique las causas que impiden el proceso.
6.4	El SAE guardará en la base de datos la información relacionada con los cursos.

Tabla 5.7: Requerimientos del sistema correspondiente al requerimiento de usuario 6

Requerimientos del usuario	
7	El SAE permitirá la inscripción a cursos a los alumnos inscritos
Requerimientos del sistema	
7.1	El SAE proporcionara una interfaz con campos para inserción del código del curso a inscribir
7.2	El SAE validará que el código pertenezca a un curso que se esté impartiendo
7.3	El SAE validará que el curso se imparta en el cuatrimestre actual
7.4	El SAE validará cada uno de los prerrequisitos del curso para poder seguir con la inscripción

7.5	El SAE verificará los lugares restantes en el curso
7.6	El SAE validará que el usuario no se haya inscrito anteriormente al curso solicitado.
7.7	En caso de la ocurrencia de algún error el SAE mostrará al usuario la causa del error.
7.8	El SAE debe de llevar a cabo la inscripción del alumno al curso, guardando la información en la base de datos.

Tabla 5.8: Requerimientos del sistema correspondiente al requerimiento de usuario 7

Requerimientos del usuario	
8	El SAE permitirá la baja de cursos de los alumnos inscritos
Requerimientos del sistema	
8.1	El SAE proporcionará una interfaz con campos para inserción del código del curso a inscribir
8.2	El SAE validará que el código pertenezca a un curso que se esté impartiendo
8.3	El SAE verificará que se encuentra en periodo de bajas
8.4	El SAE verificará que el curso haya sido inscrito anteriormente por el alumno

Tabla 5.9: Requerimientos del sistema correspondiente al requerimiento de usuario 8

Requerimientos del usuario	
9	El SAE deberá contar con interfaces amigables e intuitivas para facilitar el uso.
Requerimientos del sistema	
9.1	El SAE deberá contar con interfaces interactivas con el usuario a base de ventanas, botones y menús.
9.2	Las interfaces del SAE deberán ser lo más sencillas y claras para el usuario, deben de evitar procedimientos largos y difíciles.

Tabla 5.10: Requerimientos del sistema correspondiente al requerimiento de usuario 9

Requerimientos del usuario	
10	El SAE deberá garantizar la disponibilidad e integridad de la información en todo momento así como el acceso a ella, por medio de Internet desde cualquier parte del mundo.
Requerimientos del sistema	
10.1	El SAE debe de contar con una capa de administración de los datos, para evitar cualquier anomalía en el manejo de la base de datos.
10.2	El SAE contará con un mecanismo de acceso que permita múltiples usuarios conectados en un mismo momento, dotándolo de capacidad para atender peticiones de los usuarios.
10.3	El SAE debe de ser compatible con la mayoría de navegadores Web para el acceso externo.

Tabla 5.11: Requerimientos del sistema correspondiente al requerimiento de usuario 10

5.4. Especificación de los requerimientos

La especificación de requerimientos es la manera de detallar cada uno de los requerimientos para su aclaración y su fácil rastreo en caso de detectar errores o ambigüedades en el diseño o implementación.

La continuación se especifica cada uno de los requerimientos del usuario siguiendo una estructura de tablas. La especificación de requerimientos se hace de siguiendo el metodo de Sommerville [4].

Requerimiento # 1
Función: identificar usuarios
Descripción: El sistema identificará sólo a tres tipos de usuarios (Alumnos, coordinadores académicos y personal administrativo), y solamente a ellos se les permitirá el acceso y uso de los servicios.
Entradas: Login y password.
Salidas: Acceso al sistema o mensaje de error.
Requerimientos: que el usuario este registrado en el sistema
Precondiciones: Que existan registros de alumnos, coordinadores y personal administrativo en las bases de datos.
Postcondiciones: Uso de los servicios del sistema disponibles para el tipo de usuario.
Efectos colaterales: Ninguno

Tabla 5.12: Especificación del requerimiento #1

Requerimiento # 2

Función: uso de servicios

Descripción: El sistema permitirá el uso de los diferentes servicios con los que el sistema cuenta, distinguiendo al tipo de usuario. Le mostrará un menú con los servicios que el usuario está autorizado a usar.

Entradas: tipo de usuario.

Salidas: Listado de servicios disponibles para el usuario.

Requerimientos: que el usuario esté registrado en el sistema

Precondiciones: que el usuario haya identificado en el sistema

Postcondiciones: Listado y uso de los servicios del sistema.

Efectos colaterales: Restringir el uso de los servicios a usuarios no autorizados.

Tabla 5.13: Especificación del requerimiento #2

Requerimiento # 3 **Función:** alta, baja y modificación de cursos **Descripción:** esta función permite agregar, eliminar y modificar la información de cada uno de los cursos, en la base de datos del sistema. Además de verificar que la información proporcionada sea la correcta. **Entradas:** código del curso **Salidas:** alta, baja o modificación de la información de los cursos. **Requerimientos:** que el usuario esté registrado en el sistema **Precondiciones:** en caso de alta, que se introduzcan los datos necesario para el alta del curso. En caso de baja o modificación de curso, que el curso exista en la base de datos. **Postcondiciones:** en caso de alta, el curso será dado de alta. En caso de baja, el curso será eliminado de la base de datos y en caso de modificación, se actualizara la información relacionada al curso. **Efectos colaterales:** Si no se cumple con los requisitos para la eliminación se producirá un error en el sistema.

Tabla 5.14: Especificación del requerimiento #3

Requerimiento # 4

Función: alta, baja y modificación de usuarios

Descripción: esta función permite agregar, eliminar y modificar la información de cada uno de los usuarios registrados en el sistema. Además de verificar que la información proporcionada sea la correcta.

Entradas: código de usuario

Salidas: alta, baja o modificación de la información de los usuarios del sistema.

Requerimientos: que el usuario esté registrado en el sistema.

Precondiciones: en caso de alta, que se introduzcan los datos necesario para el alta del usuario. En caso de baja o modificación de un usuario se necesita que el usuario esté registrado en el sistema.

Postcondiciones: en caso de alta, el usuario será dado de alta. En caso de baja, el usuario será eliminado de la base de datos y en caso de modificación de la información del usuario, se actualizará la información relacionada en la base de datos.

Efectos colaterales: Si no se cumple con los requisitos para el alta, baja y modificación de la información del usuario se producirá un error en el sistema y no se podrá llevar a cabo la operación.

Tabla 5.15: Especificación del requerimiento #4

Requerimiento # 5 **Función:** Consulta de catálogo de cursos. **Descripción:** Se permitirá la consulta del catálogo de cursos que se imparten en un cierto periodo. **Entradas:** ninguna. **Salidas:** características de cada uno de los cursos. **Requerimientos:** que el usuario esté registrado en el sistema. **Precondiciones:** que exista una base de datos para los cursos, así como una pequeña descripción de cada uno. **Postcondiciones:** se mostrará cada uno de los cursos disponibles en el sistema. **Efectos colaterales:** ninguno.

Tabla 5.16: Especificación del requerimiento #5

Requerimiento # 6

Función: asignación y modificación de profesores.

Descripción: esta función está encargada de asignar un profesor a un curso. También se encarga de modificar la asignación realizada a un curso.

Entradas: código del curso.

Salidas: que un curso tenga asignado un profesor o la actualización de la información de la asignación.

Requerimientos: que el usuario esté registrado en el sistema y que sea un coordinador o personal administrativo.

Precondiciones: que exista una base de datos para los cursos, y un catálogo de profesores.

Postcondiciones: en caso de asignación, se asignará un profesor a un curso. En caso de modificación se actualizará la información contenida en la base de datos.

Efectos colaterales: si el usuario no cuenta con privilegios para llevar a cabo la operación se producirá un error en el sistema.

Tabla 5.17: Especificación del requerimiento #6

Requerimiento # 7
Función: inscripción de alumnos.
Descripción: esta función está encargada de inscribir a los alumnos a un curso que se imparte.
Entradas: código del curso y alumno.
Salidas: inscripción del alumno al curso solicitado.
Requerimientos: que el usuario esté registrado en el sistema y que esté inscrito.
Precondiciones: que exista una base de datos para los cursos y un catálogo de alumnos.
Postcondiciones: el alumno quedará inscrito en el curso.
Efectos colaterales: si el usuario no cuenta con privilegios para llevar a cabo la operación se producirá un error en el sistema.

Tabla 5.18: Especificación del requerimiento #7

Requerimiento # 8
Función: baja de cursos
Descripción: esta función está encargada de dar de baja a los alumnos de un curso en el que se encuentra inscrito.
Entradas: código del curso y alumno.
Salidas: baja del alumno del curso solicitado.
Requerimientos: que el usuario este registrado en el sistema, que esté inscrito y que sea periodo de bajas.
Precondiciones: que exista una base de datos para los cursos, un catálogo de alumnos y que el usuario este inscrito al curso que desea dar de baja.
Postcondiciones: el alumno quedará dado de baja del curso.
Efectos colaterales: si el usuario no cuenta con privilegios para llevar a cabo la operación se producirá un error en el sistema.

Tabla 5.19: Especificación del requerimiento #8

5.5. Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que no se refieren directamente a las funciones específicas que entrega el sistema sino a las capacidades de este como un todo. Asimismo, estos requerimientos definen las restricciones que tendrá el sistema durante su desarrollo y operación. En la Tabla 5.20 se muestran los requerimientos no funcionales para el sistema SAE.

#	**Descripción**
1	El tiempo de respuesta del sistema a cualquier transacción no debe sobrepasar los 15 segundos.
2	Debe contarse con un dispositivo de almacenamiento de por lo menos 50 GB, reservados específicamente para el sistema.
3	Para garantizar el funcionamiento del sistema se debe de contar con conexión de banda ancha hacia Internet.
4	Las páginas del SAE deben de contener únicamente la información que es necesaria para una operación específica, para disminuir el tiempo de respuesta y mejorar la experiencia del usuario
5	El funcionamiento del SAE a través de Internet debe ser independiente del tipo de usuario al que se atiende (PC, PDA, móvil) y del navegador utilizado por este último.
6	El tamaño de memoria necesario para garantizar el funcionamiento óptimo del SAE debe de ser mínimo de 2 GB.
7	La tasa de fallas para que el sistema sea aceptable debe ser del 3 %.
8	Cada página del SAE debe realizar validaciones de entrada de datos del lado del cliente, para garantizar la integridad de la información contenida en la base de datos.

Tabla 5.20: Requerimientos no funcionales del SAE

5.6. Escenarios

Los escenarios ayudan a modelar el comportamiento del sistema. El comportamiento del sistema va a estar definido por la interacción con el usuario u otros sistemas. Después de tener la especificación de cada uno de los requerimientos en lenguaje natural, se necesita traducir los requerimientos a un modelo que permitá a los desarrolladores entenderlos. Para una descripción grafica de cada uno de los requerimientos del sistema utilizamos los diagramas de casos de uso y de secuencia del lenguaje de modelado unificado (UML, por sus siglas en inglés, *Unified Modeling Language*).

En la Figura 5.1 se muestra el diagrama de casos de uso del sistema, con sus respectivos actores directos, note que aunque el administrador tiene acceso a todo el sistema, en el diagrama sólo se muestran los casos de uso con los que se relaciona directamente.

Los actores representan al tipo de usuario que va a tener acceso al sistema. En nuestro caso consideramos cuatro tipos principales: Administrador, Coordinador Académico, Personal Administrativo y Alumno.

5.6.1. Diagramas de casos de uso del administrador.

En la Figura 5.2 se muestran sólo los casos de uso del actor administrador y las acciones que son validas para este tipo de usuario. Para el SAE, el administrador es el encargado de la administración de usuarios (altas, bajas y cambios). El actor administrador es un usuario que lleva el control de los usuarios del SAE.

5.6.2. Diagramas de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en la aplicación a través del tiempo y se modela para cada caso de uso. El diagrama de secuencia contiene detalles de implementación, incluyendo los objetos, clases y mensajes intercambiados entre los objetos. En estos diagramas se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación.

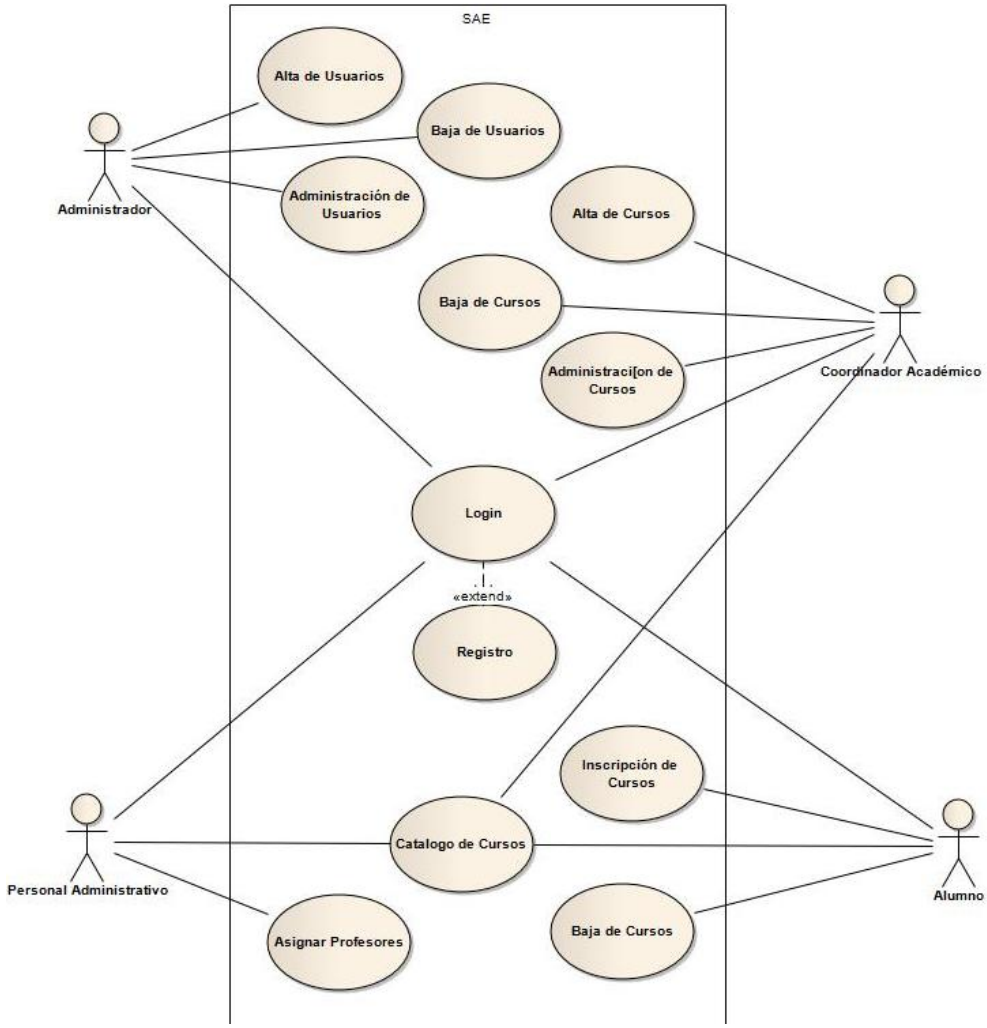

Figura 5.1: Casos de uso del SAE

Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes enviados entre los objetos como flechas horizontales.

En las Figuras 5.3, 5.5 y 5.6 se muestran los diagramas de secuencia asociados al usuario administrador describiendo a cada uno de los casos de uso de la Figura 5.2. Cada diagrama de secuencia muestra los procedimientos que el sistema debe de seguir para poder lograr una determinada tarea.

En la Figura 5.3 se describe el procedimiento para poder dar de alta a un usuario en el sistema. El resultado de tal operación es que el usuario pueda tener acceso al

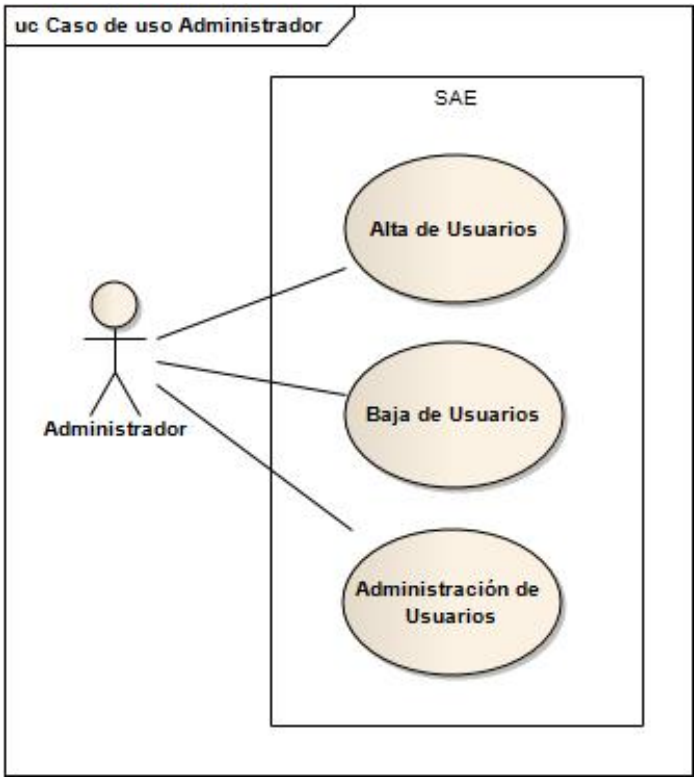

Figura 5.2: Casos de uso del actor Administrador

sistema.

En la Figura 5.5 se muestra el diagrama de secuencia que describe los pasos que el sistema debe de seguir para poder dar de baja a un usuario del sistema. El resultado de dicha operación es que el usuario no pueda tener acceso al sistema.

En la Figura 5.5 se muestran los pasos que el sistema debe de seguir para poder actualizar los datos de un usuario registrado. La salida de este proceso consiste en la actualización de datos del usuario, como podría ser una nueva descripción o tan solo el cambio de Login y password.

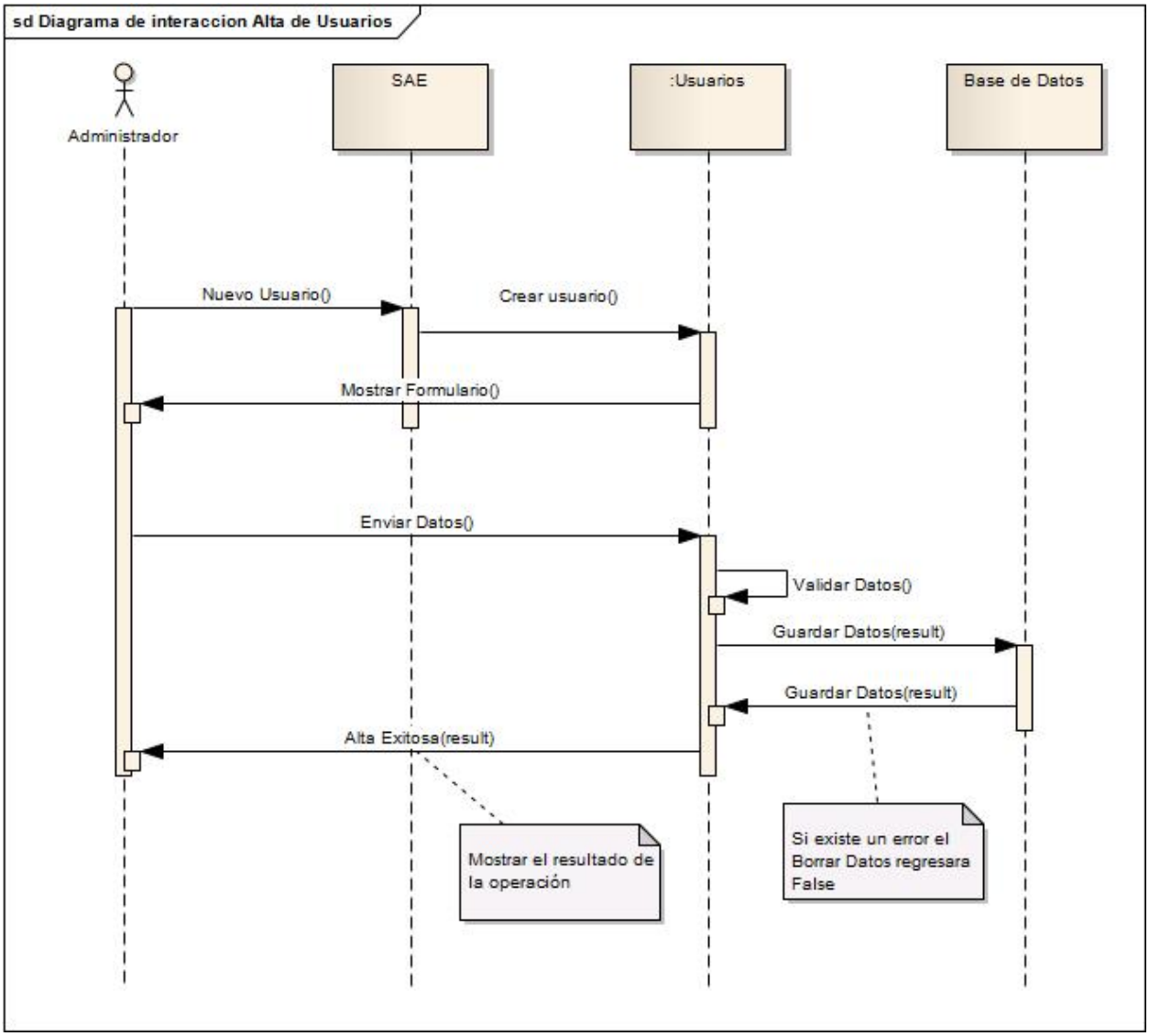

Figura 5.3: Diagrama de secuencia del caso de uso Alta de usuario

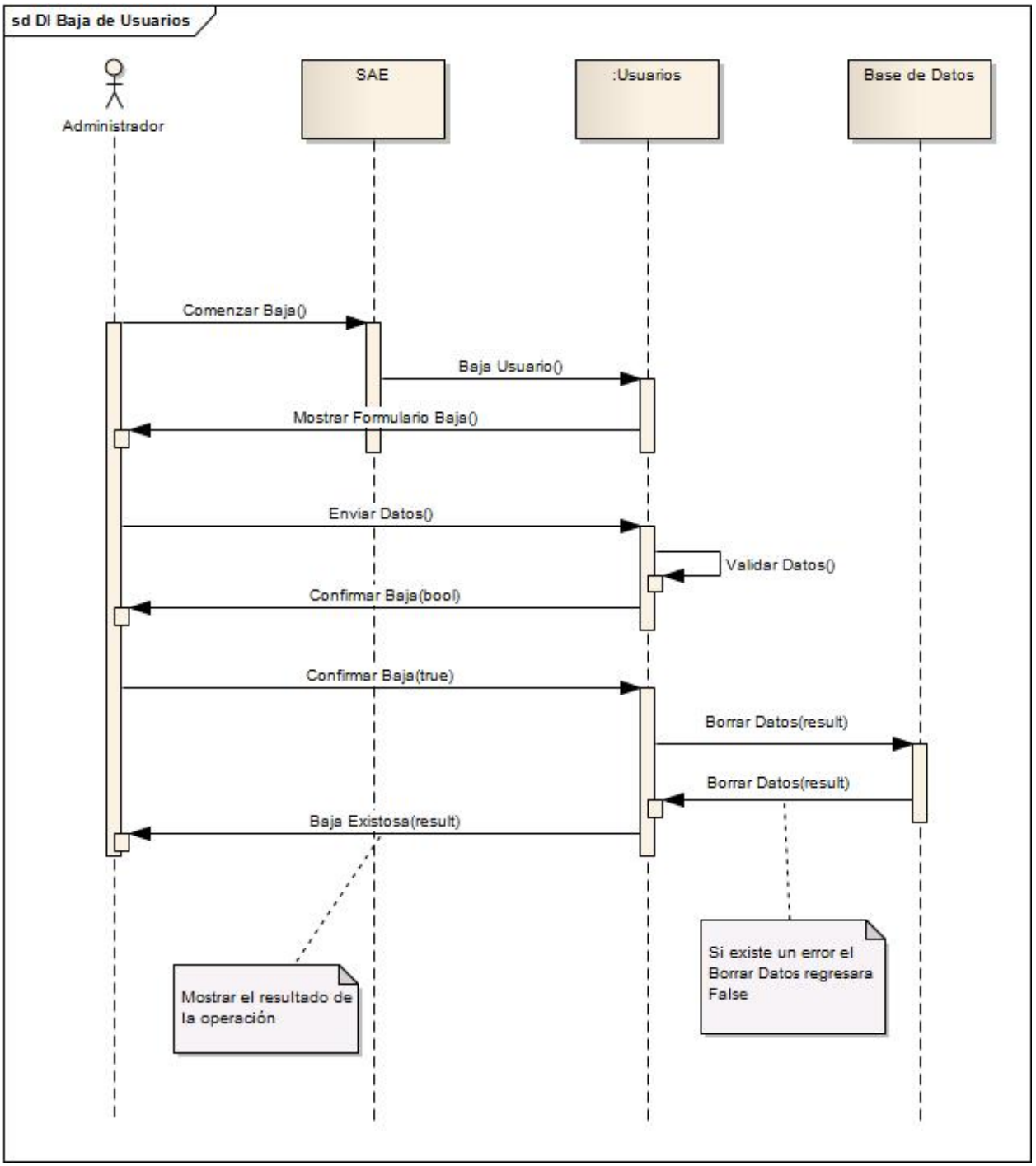

Figura 5.4: Diagrama de secuencia del caso de uso Baja de usuario

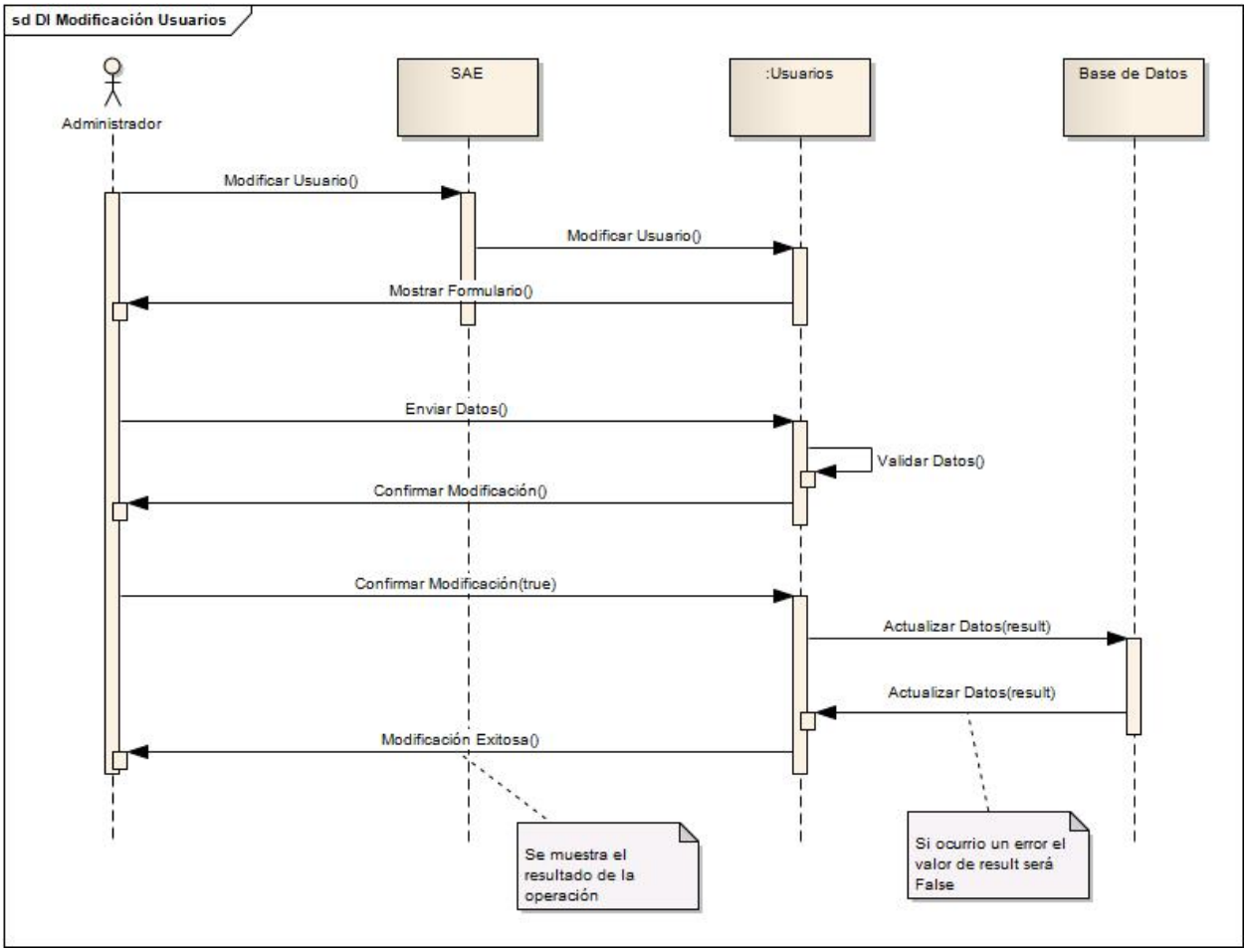

Figura 5.5: Diagrama de secuencia del caso de uso Administración de usuario

5.6.3. Diagramas de casos de uso del coordinador académico

En la Figura 5.6 se muestran los casos de uso del tipo de usuario coordinador académico. Los casos de uso representan las tareas o acciones que son validas para este tipo de usuario. Para el SAE el coordinador académico es el encargado de la gestión de los cursos y sus principales tareas son el alta de cursos, baja de cursos y la administración de los cursos.

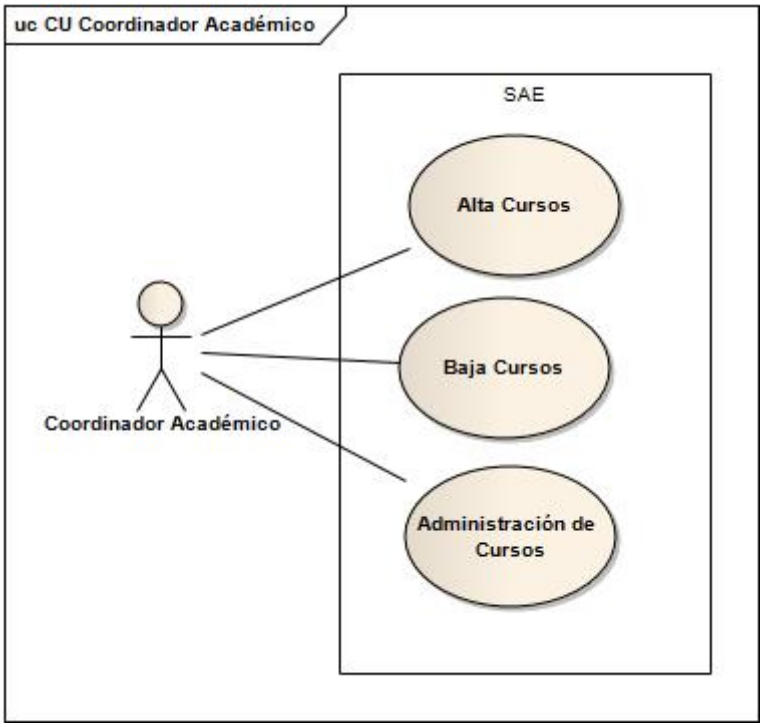

Figura 5.6: Diagrama de casos de uso del usuario coordinador académico

5.6.4. Diagramas de secuencia

El diagrama de secuencia de la Figura 5.7 muestra el proceso que el sistema debe de seguir para poder dar de alta un nuevo curso. Como se puede apreciar esta tarea es exclusiva del coordinador académico. El resultado de esta operación es la creación de un nuevo curso.

El proceso que se describe en el diagrama de secuencia de la Figura 5.8 es el de

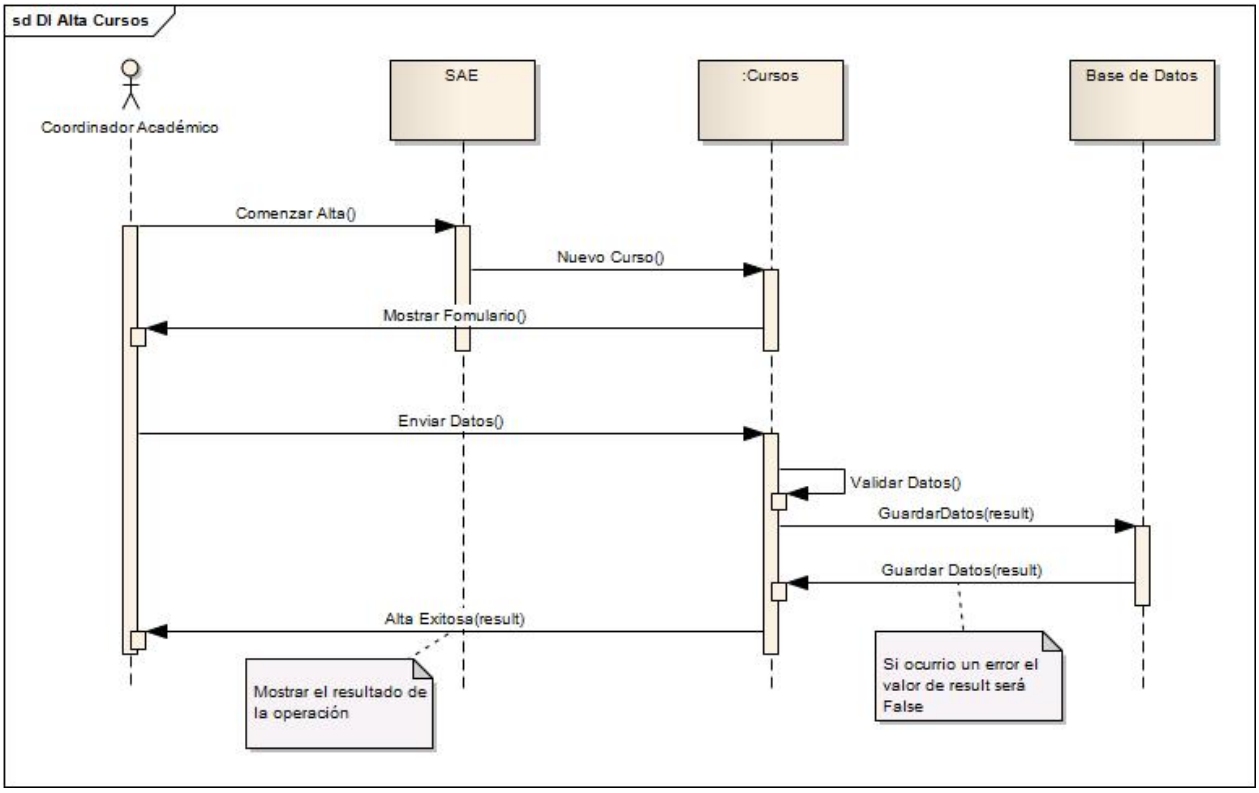

Figura 5.7: Diagrama de secuencia para el caso de uso alta cursos

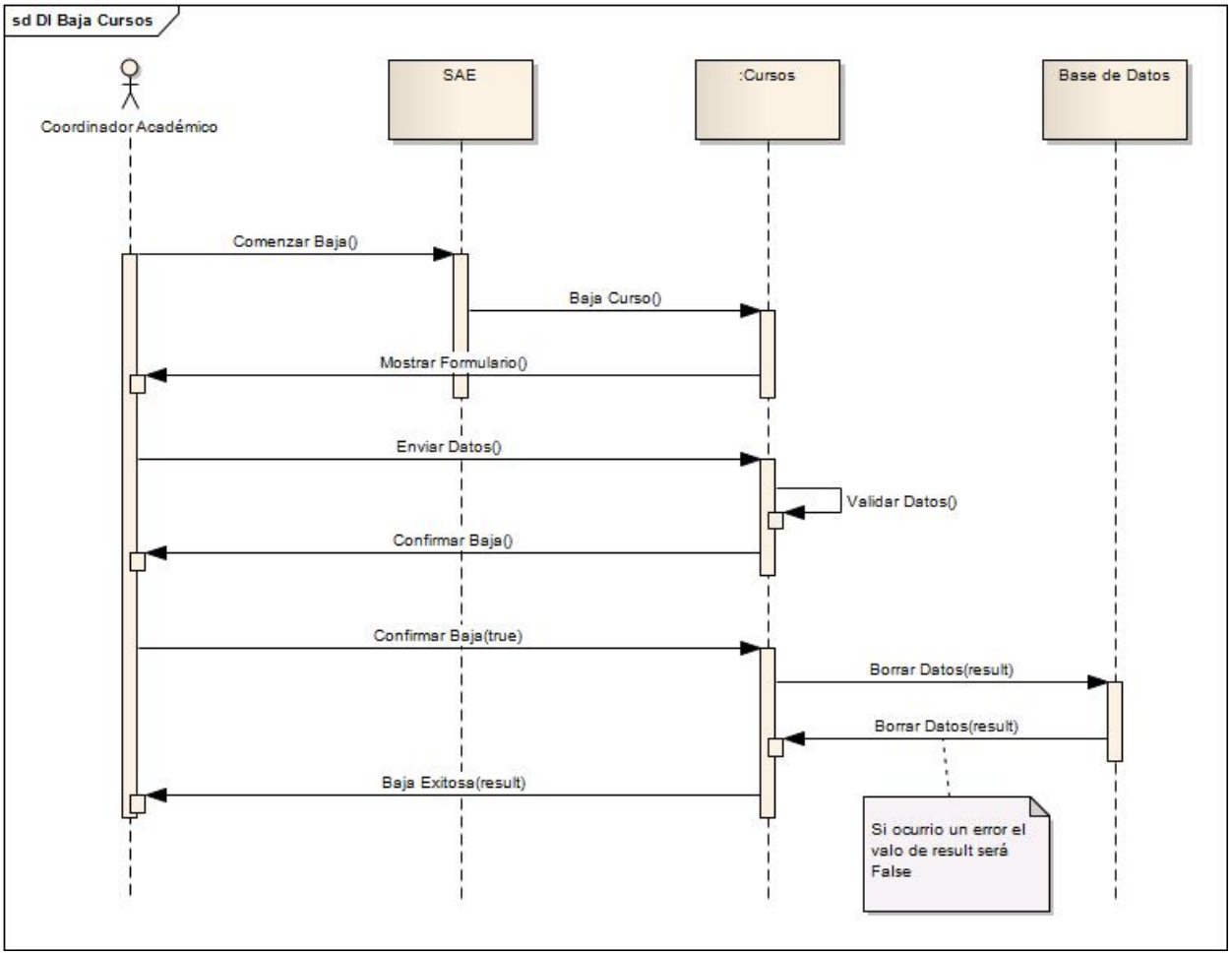

Figura 5.8: Diagrama de secuencia para el caso de uso baja cursos

dar de baja un curso del plan de estudios. La salida de este proceso es que el curso quede eliminado del sistema.

Si no se necesita eliminar el curso se puede optar por actualizar la información perteneciente al curso. Para lograr esto se necesita que el sistema siga el procedimiento que se muestra en la Figura 5.9.

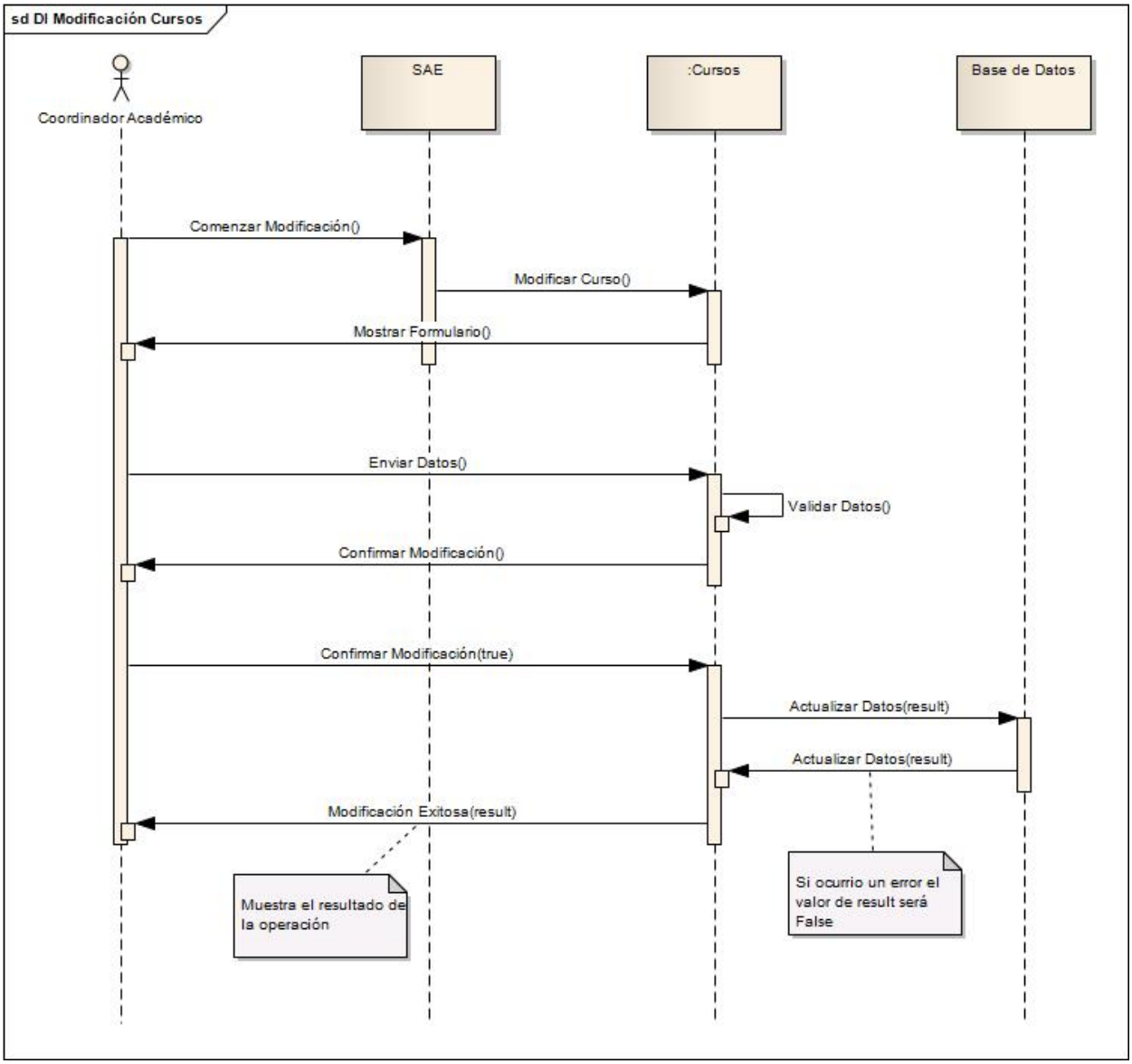

Figura 5.9: Diagrama de secuencia para el caso de uso administración cursos

5.6.5. Diagramas de casos de uso del Personal Administrativo

En el Diagrama de casos de uso de la Figura 5.10 se muestran los casos de uso del personal administrativo. Para el SAE, el personal administrativo es el tipo de usuario acreditado para la asignación de un profesor a un curso y a su vez la modificación de dicha asignación.

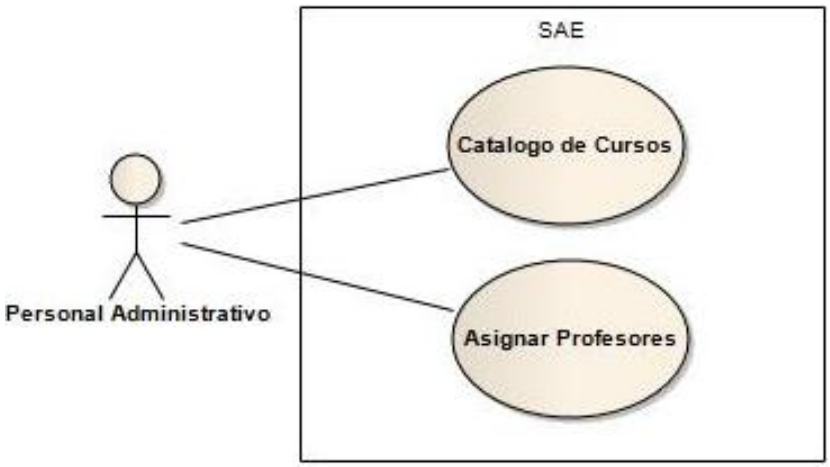

Figura 5.10: Diagramas de casos de uso del Personal Administrativo

5.6.6. Diagramas de secuencia

En la Figura 5.11 se describe el procedimiento que sigue el sistema para consultar el catalogo de cursos, mientras que en la Figura 5.12 se describe el procedimiento el sistema debe segui para que el personal académico pueda asignar un profesor a un cursor

.

En la Figura 5.13 se describe el proceso para cambiar al profesor asignado a un curso

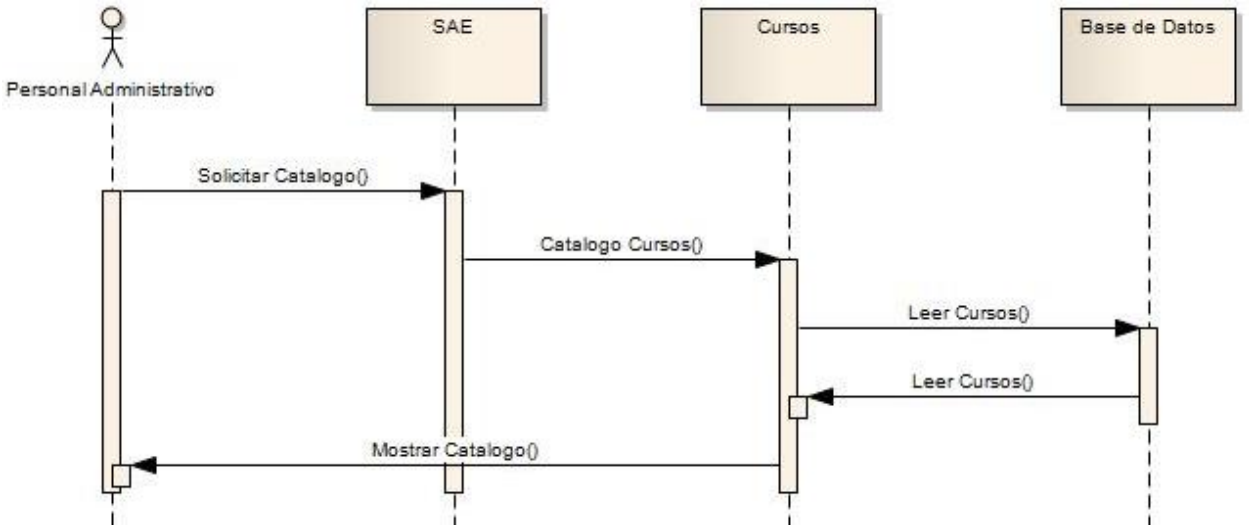

Figura 5.11: Diagramas de casos de uso catalogo de cursos

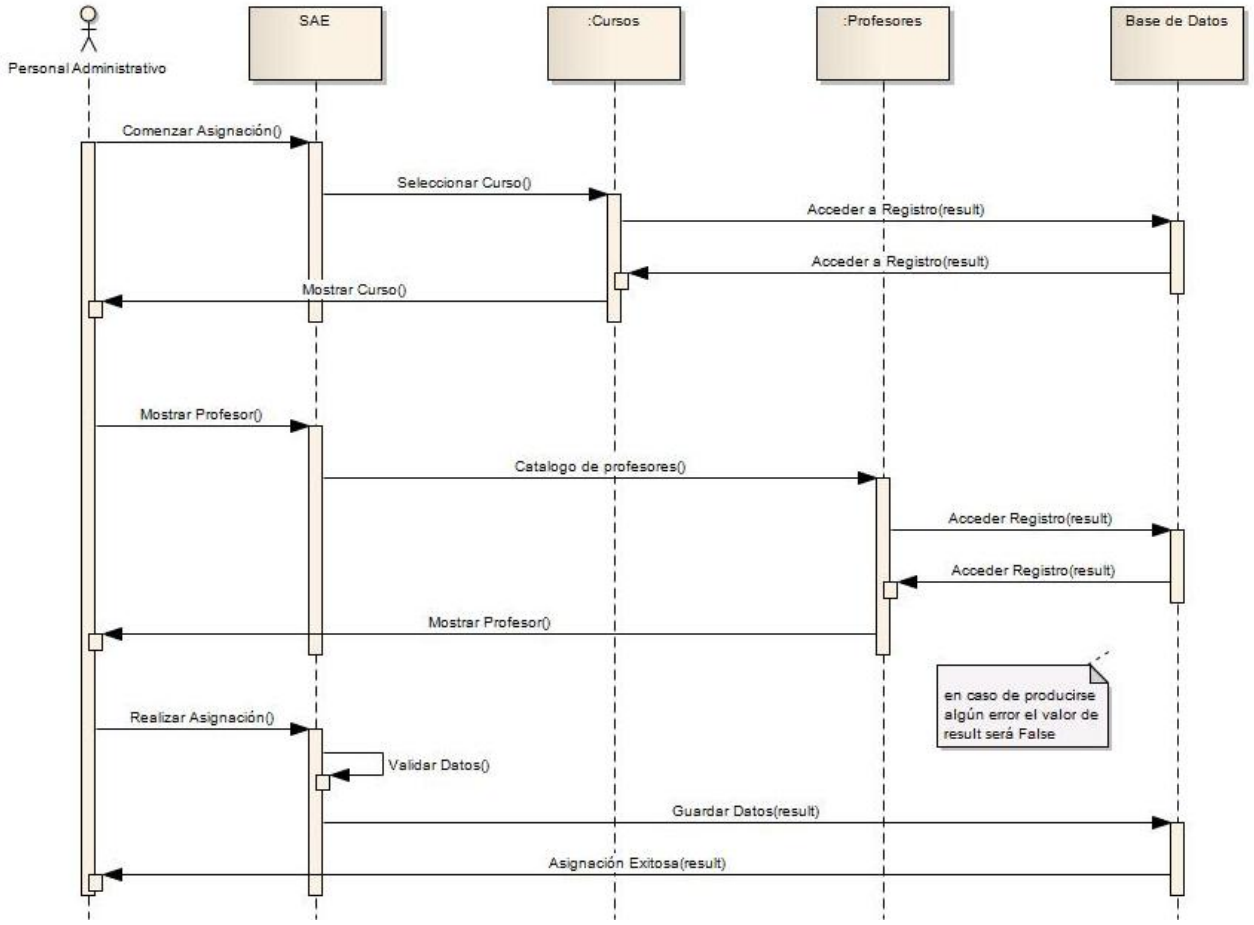

Figura 5.12: Diagramas de casos de uso asignar profesores

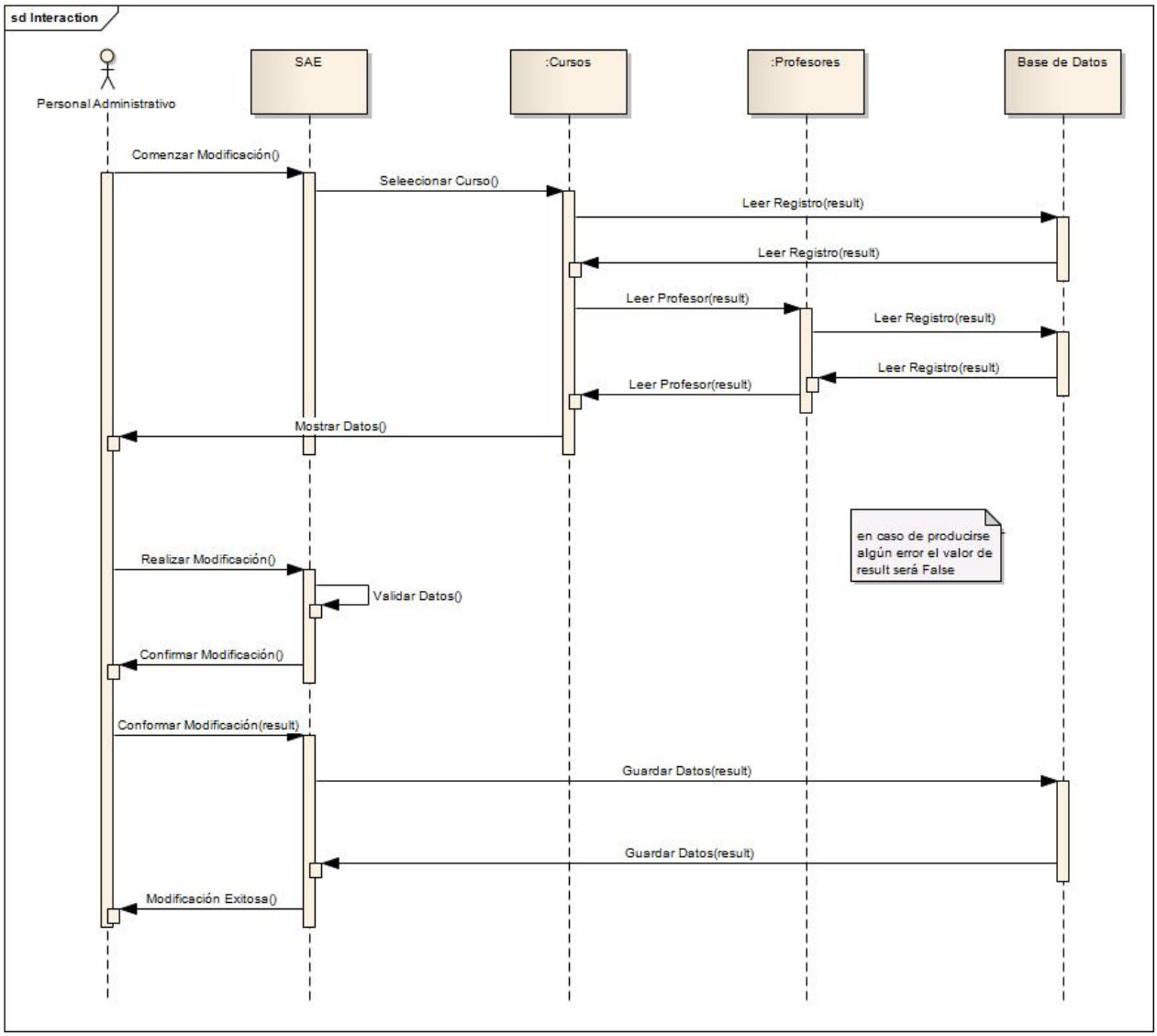

Figura 5.13: Diagramas de casos de uso Modificación profesores

5.6.7. Diagramas de casos de uso del Alumno

En el diagrama de secuencia de la Figura 5.14 se muestra solo los casos de uso del alumno. En el SAE el usuario alumno tiene la capacidad de realizar una inscripción a un curso o darse de baja de un curso.

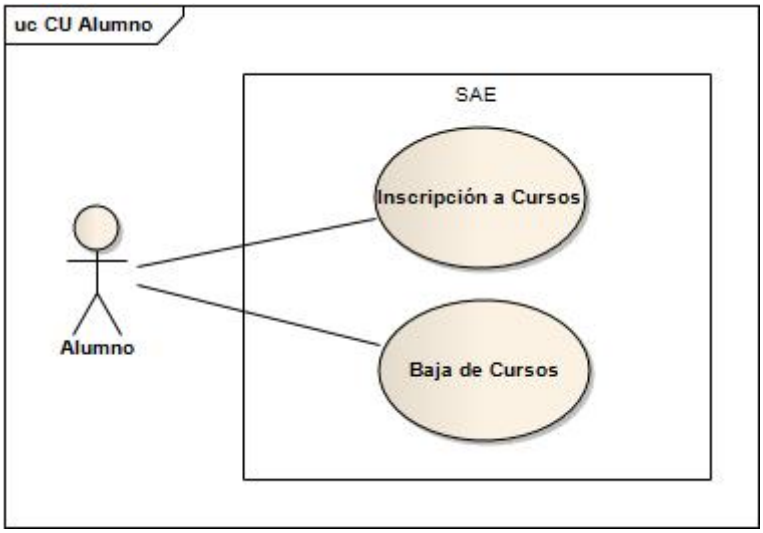

Figura 5.14: Diagramas de caso de uso de Alumnos

5.6.8. Diagramas de secuencia

En la Figura 5.15 se describe el proceso de inscripcion para el alumno. En el proceso de inscripción es un proceso de suma importancia, por tal motivo se necesita la interacción de la mayoría de los módulos del sistema. El proceso de inscripción puede incluir acciones repetitivas, representardas por *ciclos*, que se necesitan identificar y resaltar en el diagrama de secuencia. En la Figura 5.15 se puede observar un cuadro que indica las acciones que son repetitivas.

El SAE cuenta con un módulo que permite dar de baja alguna materia, inscrita previamente por el alumno. El proceso que el sistema sigue se describe en la Figura 5.16.

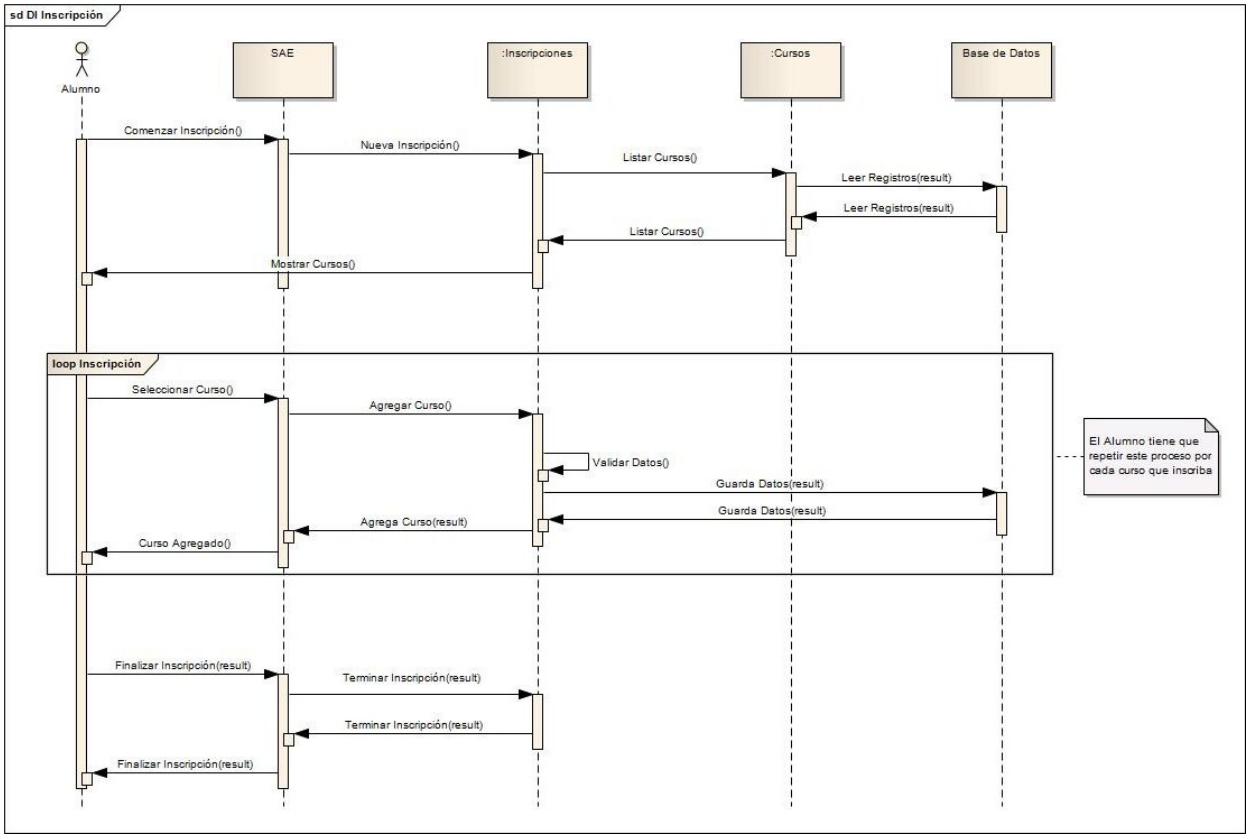

Figura 5.15: Diagrama de secuencia del caso de uso inscripción a cursos

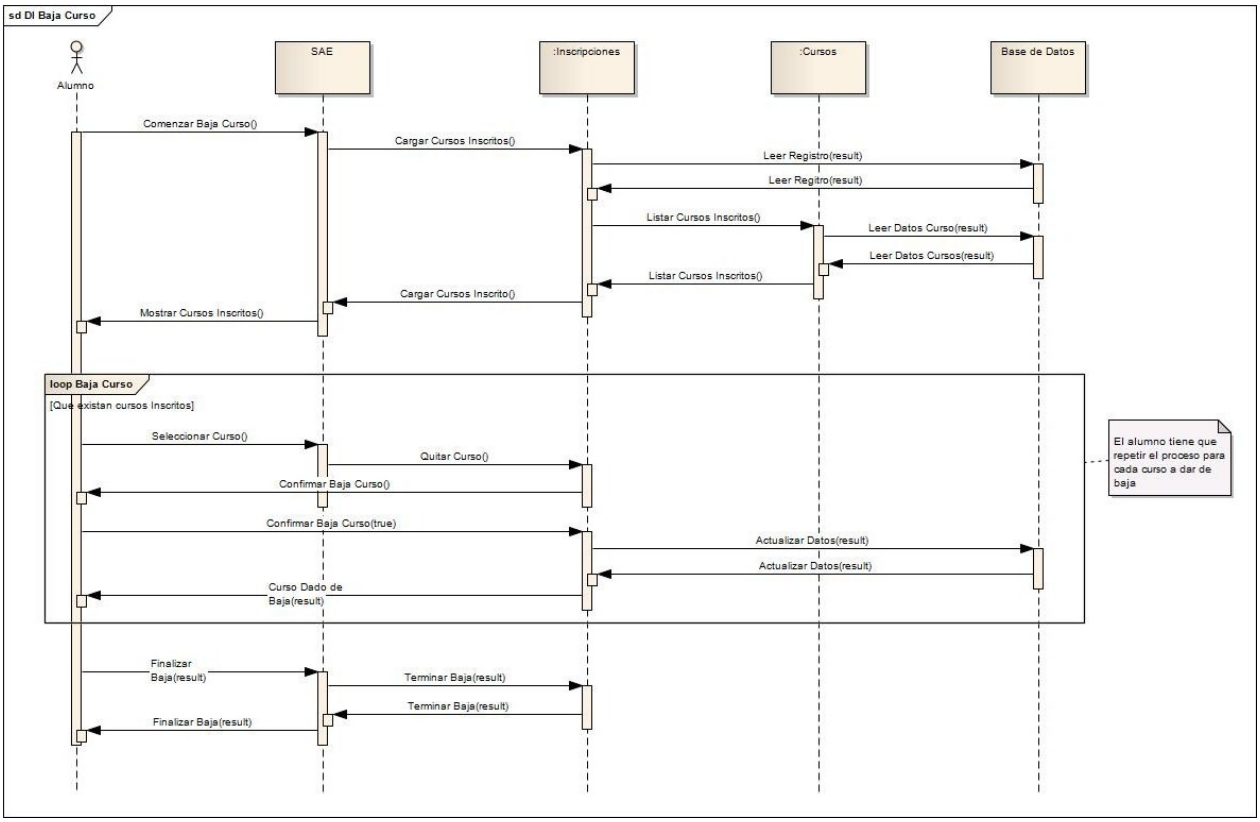

Figura 5.16: Diagrama de secuencia del caso de uso baja de curso

5.7. Arquitectura

Una vez que hemos reunido los requerimientos de la aplicación y hemos analizado a cada uno por medio de los escenarios (diagramas de casos de uso y diagramas de secuencia), podemos proponer una arquitectura para desarrollar una aplicación que cumpla con los requerimientos antes mencionados. La arquitectura muestra la organización general del sistema, los componentes y la interacción que se tiene entre todos los componentes. Para la arquitectura del SAE utilizaremos la arquitectura propuesta en el capítulo 4.

A continuación se describen cada una de las vistas de la arquitectura.

5.7.1. Vista Lógica

La vista lógica nos ayuda a definir la estructura de nuestra aplicación. Podemos comenzar con una definición de alto nivel y posteriormente ir definiendo a mayor detalle cada uno de los componentes. Para nuestro caso de estudio tenemos una vista lógica que contempla una arquitectura de N capas como se muestra en la Figura 5.17.

- **Capa del cliente**: contiene todos los componentes interactuan con el cliente. En esta capa se debe de contar con un navegador Web y recursos (plug-ins) que permita al usuario tener acceso a la aplicación.

- **Capa de presentación**: contiene los componentes necesarios para interactuar con las peticiones del usuario. En esta capa se manejan las interfaces de usuario y los procesos encargados para el manejo de las mismas. También se incluyen los mecanismos para el manejo de sesiones de usuario.

- **Capa de negocio**: contiene los componentes encargados de transformar las peticiones (provenientes de la capa de presentación) en respuestas aplicando las reglas de negocio. Además de tener las reglas de negocio están los componentes necesarios para el acceso a la capa de datos o persistencia.

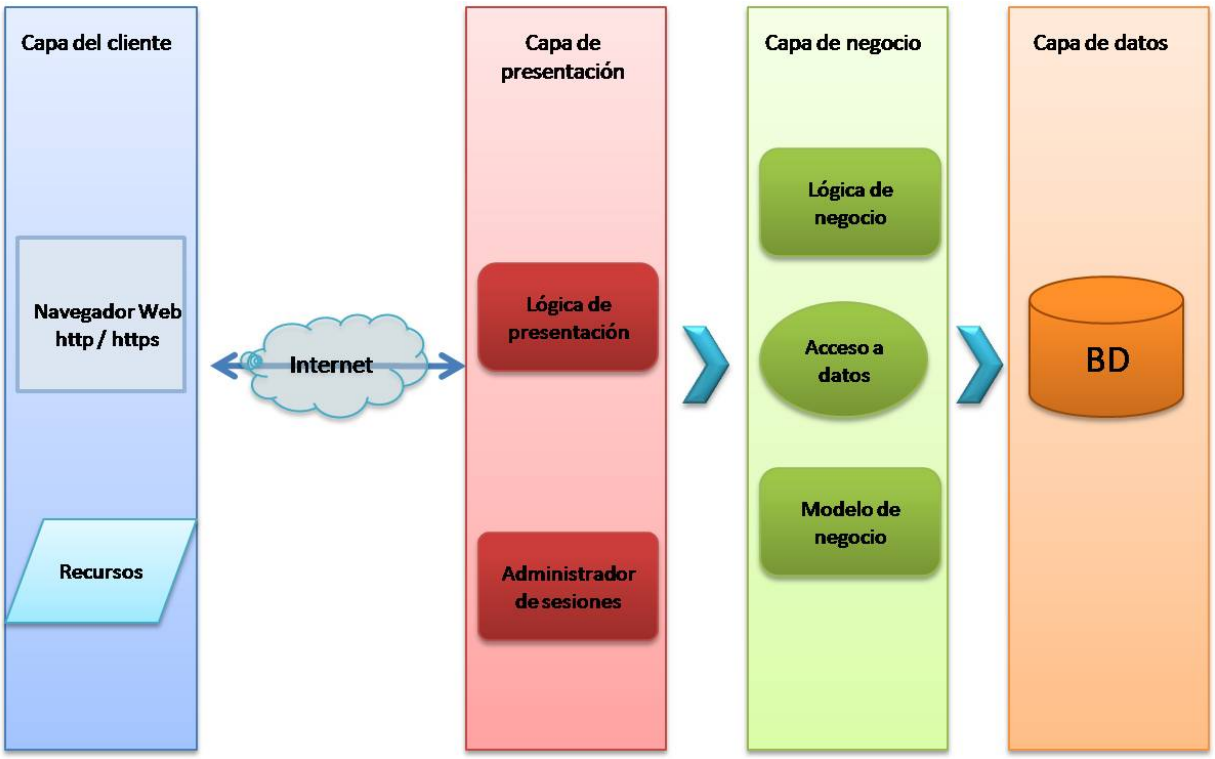

Figura 5.17: Arquitectura de N capas para una aplicación Web

- **Capa de datos**: es la encargada de proporcionar los recursos necesarios para el almacenamiento de los datos. Esta capa contiene componentes lógicos (estructuras, definiciones) y físicos (discos duros, cintas magnéticas) que permiten almacenar toda información requerida por la capa de negocio.

La vista física de la Figura 5.17 muestra los componentes de la aplicación a un nivel muy alto. Esto permite observar su estructura a nivel general pero no a un nivel más detallado. La vista física contempla varios diagramas que permiten observar a cada capa de forma más detallada. La arquitectura de N capas nos permite aislar o encapsular en cada capa determinadas responsabilidades y como consecuencia un bajo acoplamiento entre los distintos componentes de cada capa. Debido a que cada capa aísla determinadas responsabilidades va a existir una dependencia de capas inferiores.

El siguiente paso que necesitamos realizar es identificar las relaciones que existen entre cada una de las capas de la arquitectura. En la Figura 5.18 podemos ver un

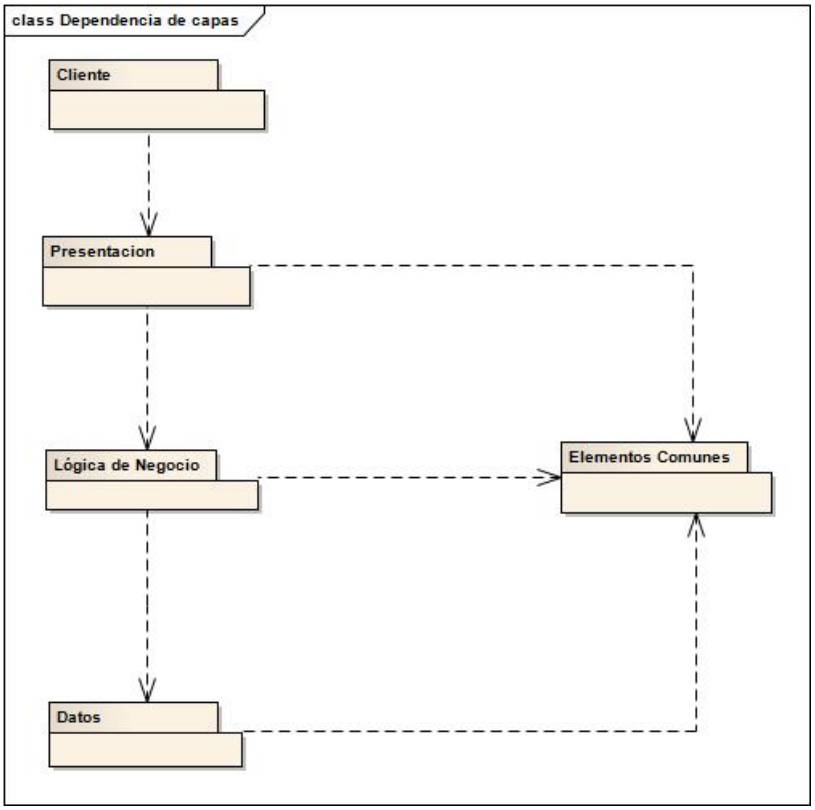

Figura 5.18: Dependencias entre las capas de la arquitectura de N capas

diagrama de paquetes que muestra las dependencias entre las diferentes capas de nuestra arquitectura. Este diagrama de paquetes ayuda a los diseñadores a identificar los componentes necesarios en cada capa.

Una vez identificadas las capas de nuestra aplicación y la relación que se tiene con las demás, podemos comenzar a definir los componentes de cada una de las capas de la aplicación.

Considerando las dependencias que pueden tener cada una de las capas se puede tener una distribución apropiada de los componentes de acuerdo a las necesidades de cada capa. En la Figura 5.19 se muestra la distribución de los componentes en las distintas capas de nuestra arquitectura.

- **Cliente**

  Para el cliente lo único que necesitamos es un navegador Web para poder acce-

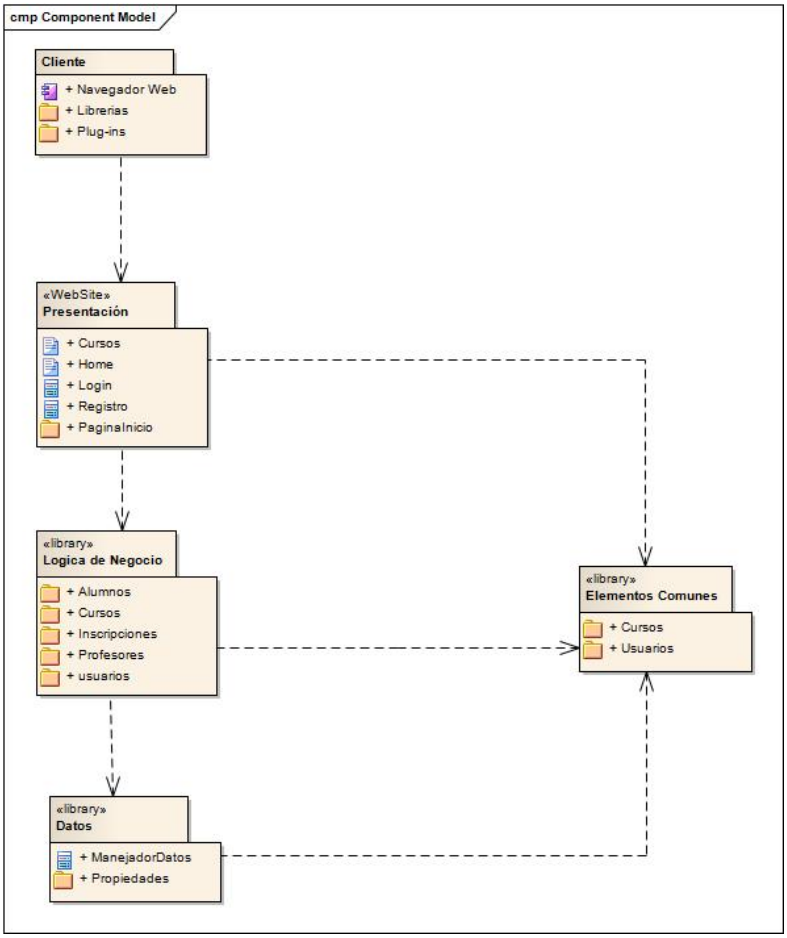

Figura 5.19: Distribución de los componentes entre las capas

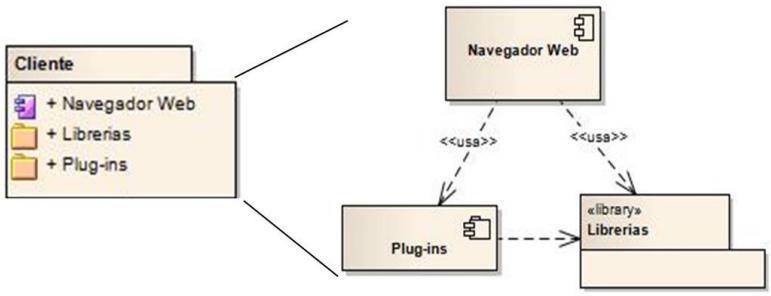

Figura 5.20: Componentes de la capa del cliente

der a nuestra aplicación. Recordemos que se necesita de componentes externos (Plug-ins, librerías) para tener una interfaz más rica en recursos multimedia.

- **Presentación**

  La presentación considera todas las interfaces que van a interactuar directamente con el cliente. En nuestro caso tenemos una página de inicio que va a contener cuatro presentaciones para el cliente.

  - *Home*: es la interfaz que se muestra por default al acceder a la aplicación. Se puede representar por medio de una página Web que contengo el nombre de la institución, logotipo e información general de la institución. Esta página tiene comunicación con las demás interfaces de usuario.
  - *Cursos*: es la interfaz encargada de mostrar información relacionada con los cursos que se imparten en la institución educativa. Se tiene acceso desde la página de Home porque no se necesita ningún privilegio para acceder a esta información.
  - *Registro*: es la interfaz encargada de dar de alta al usuario dentro del sistema. Cuando el usuario desea acceder al sistema y no ha ingresado su login y password, el sistema mostrará esta interfaz.
  - *Login*: es la interfaz encargada de autenticar al usuario por medio de un nombre de usuario (user) y una contraseña (password). Una vez que se han

presentado las credenciales necesarias se mostrará una nueva interfaz que contiene todas las opciones a las que el cliente tiene acceso.

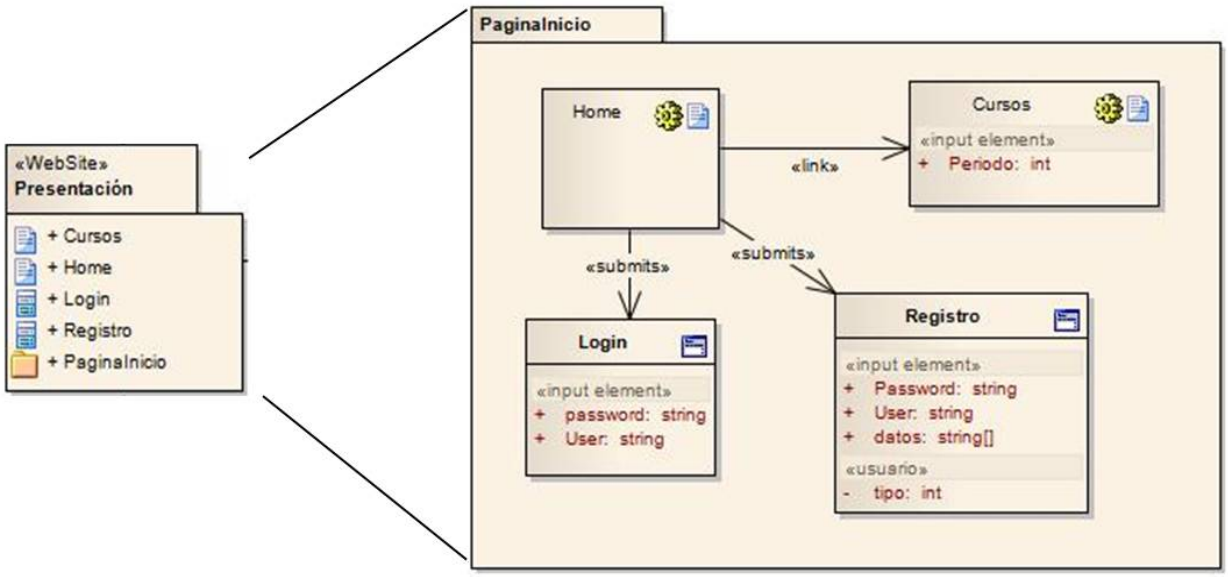

Figura 5.21: Capa de Presentación

- **Lógica de Negocio**

  La capa de lógica de negocio tiene todas las reglas necesarias para atender a las peticiones de los usuarios. En la Figura 5.22 se observan los componentes que se incluyen en esta capa y las dependencias que existen. En la Figura 5.23 se puede observar con más detalle la capa de lógica de negocio. Se puede observar las clases que lo conforman y la relación que existe entre ellas.

- **Datos**

  Los detalles de la capa de datos se muestran en la Figura 5.24. Dentro de los componentes tenemos un manejador de datos para la administración de la información obtenida de la capa de persistencia. Dentro de las propiedades tenemos un *conector* para establecer la comunicación con la capa de persistencia. Un *datasource* para especificar las fuentes de datos a las que vamos a tener acceso.

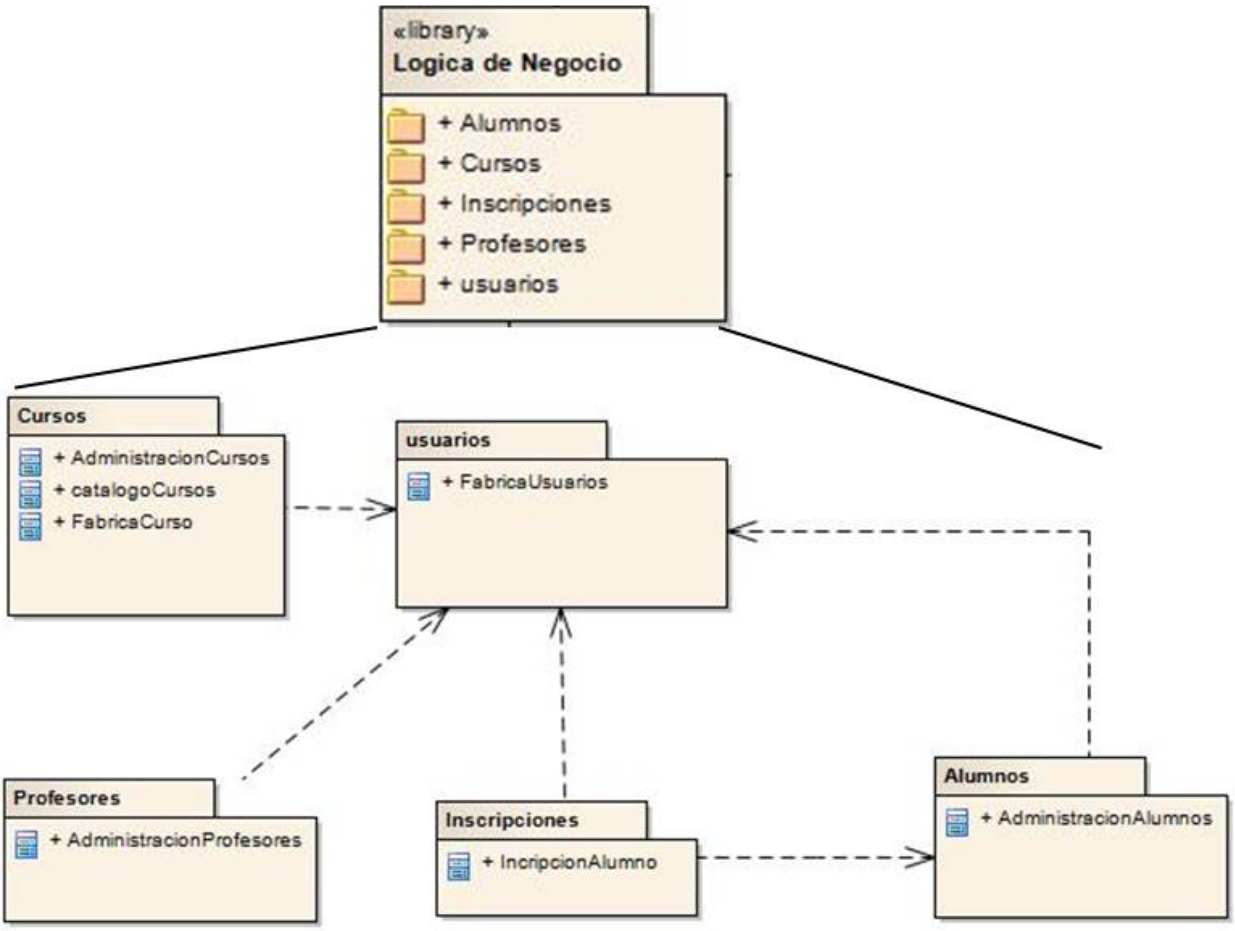

Figura 5.22: A) Capa de lógica de negocio

- **Elementos Comunes**

  En la Figura 5.25 se pueden observar los componentes de la capa de elementos comunes. Esta capa está destinada a hospedar todos los objetos que van a ser utilizados por cualquier capa dentro de nuestra aplicación. Dentro de los elementos comunes dividimos en Cursos y Usuarios de acuerdo al tipo de clase que se quiera manejar.

  Dentro de usuarios manejamos objetos como Usuario. De este objeto se pueden derivar Profesores, Alumnos y Administradores de acuerdo a las características que se requieren para cada Objeto.

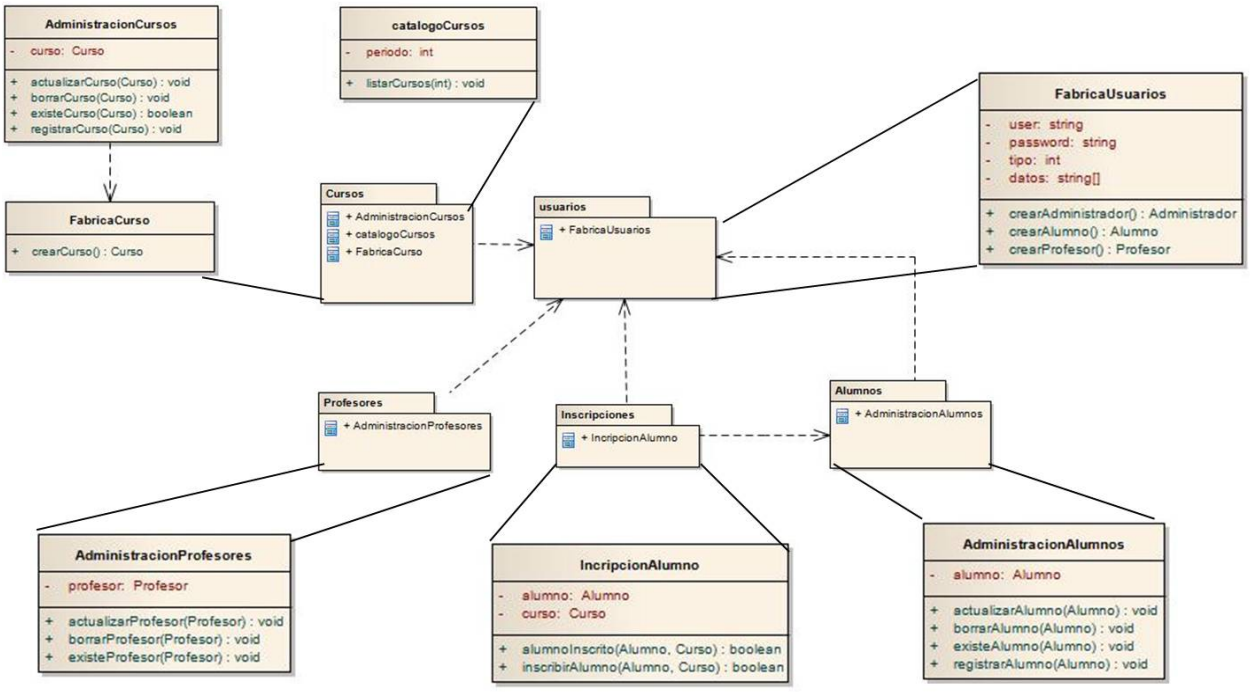

Figura 5.23: B) Capa de lógica de negocio

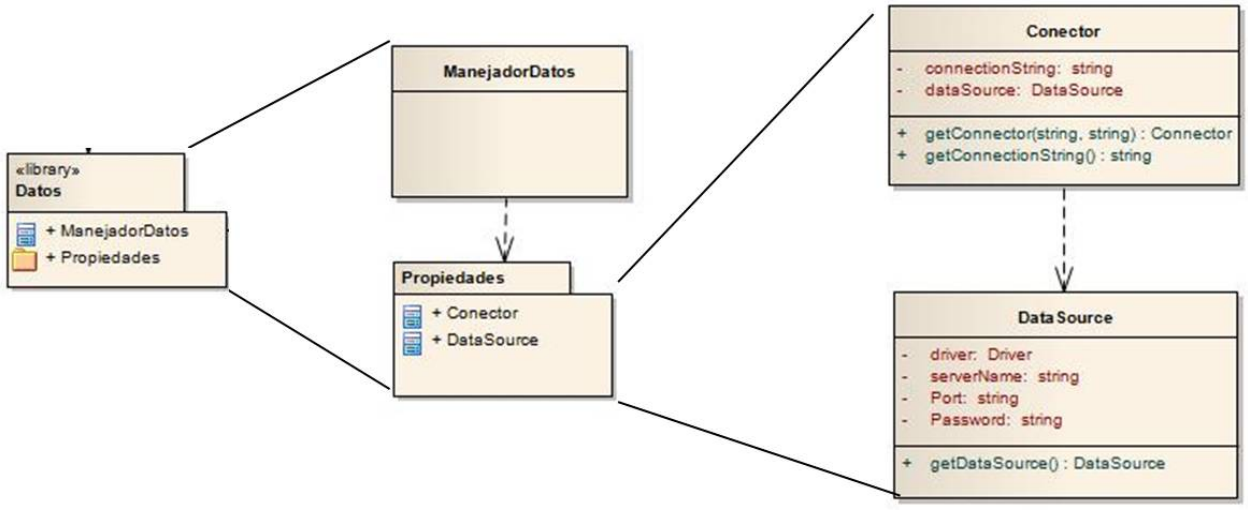

Figura 5.24: Capa de datos

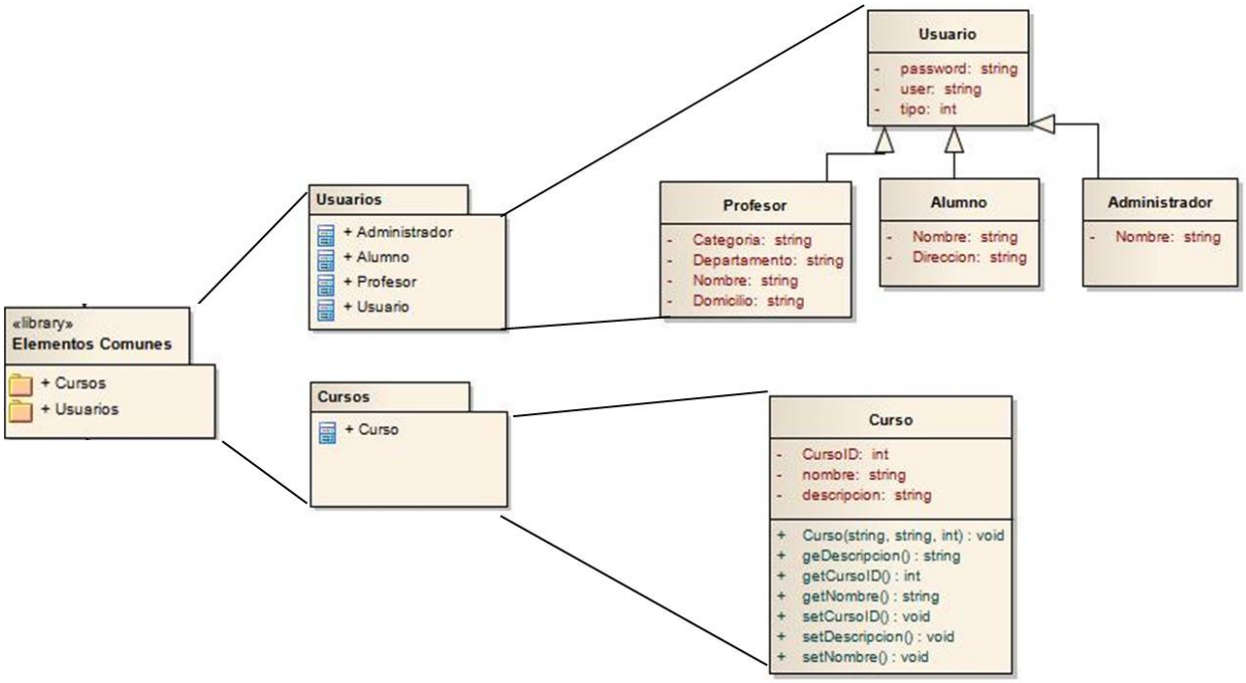

Figura 5.25: Capa de elementos comunes

5.7.2. Vista de Procesos

En la Figura 5.26 se muestra la vista de procesos de nuestra aplicación. La vista de procesos tiene como objetivo principal modelar el proceso de negocio de la aplicación.

El proceso de la Figura 5.26 muestra la interacción entre el usuario y el sistema para validar el acceso al sistema. El usuario envia la información al sistema, el sistema permitirá el acceso dependiendo de la validez de la información proporcionada por el usuario. Para único estado válido es cuando el usuario se ha identificado ante el sistema.

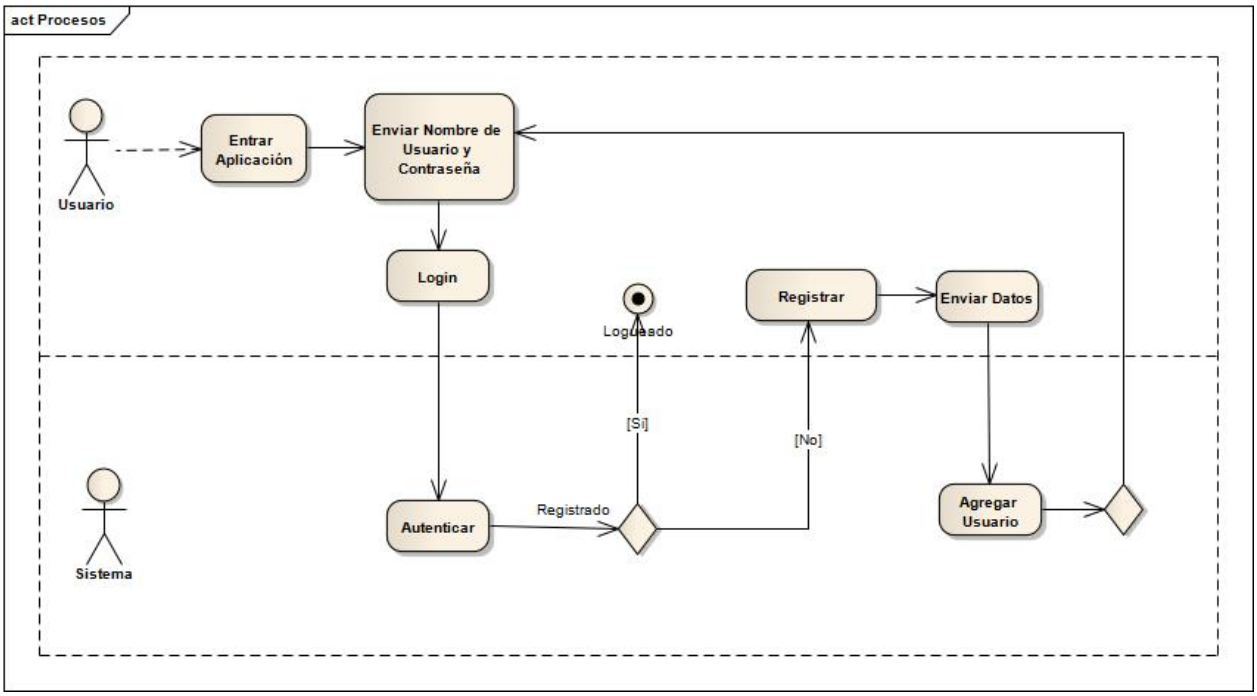

Figura 5.26: Vista de procesos

5.7.3. Vista de seguridad

La vista de seguridad muestra los métodos, técnicas y componentes necesarias para brindar seguridad a nuestra aplicación Web. En la Figura 5.27 se muestra el nivel de seguridad para cada una de los usuarios que necesitan tener acceso al sistema.

Los niveles de seguridad que se manejan en nuestra aplicación son (1) Login mediante el uso de un nombre de usuario y una contraseña. (2) La autenticación mediante certificados digitales y (3) el cifrado de los datos.

El nivel de seguridad en una aplicación Web depende de la sensibilidad de la información de cada usuario. En nuestro esquema se ha optado por cifrar la información de todos los usuarios.

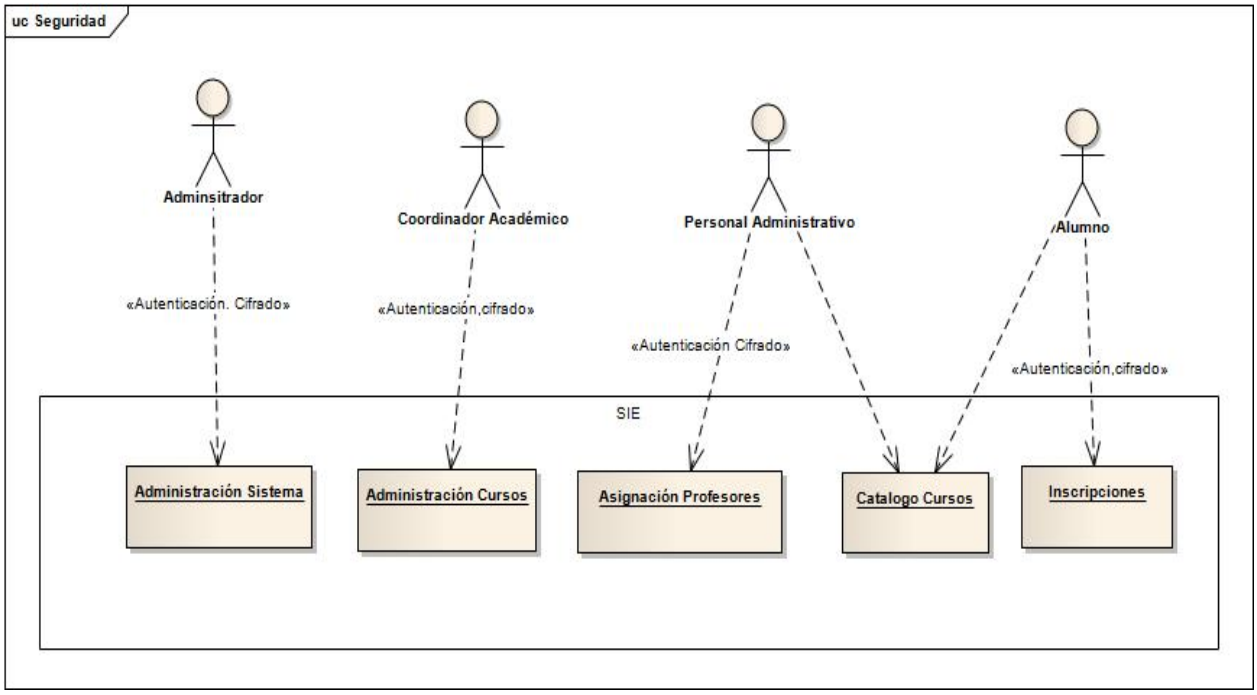

Figura 5.27: Vista de Seguridad

5.7.4. Vista de Desarrollo

La vista de desarrollo de nuestra aplicación Web está descrita en la Figura 5.28. Esta vista muestra todos los componentes de nuestra aplicación y la relación que existen entre ellos. Esta vista permite tener un panorama global sobre los componentes que se van a desarrollar.

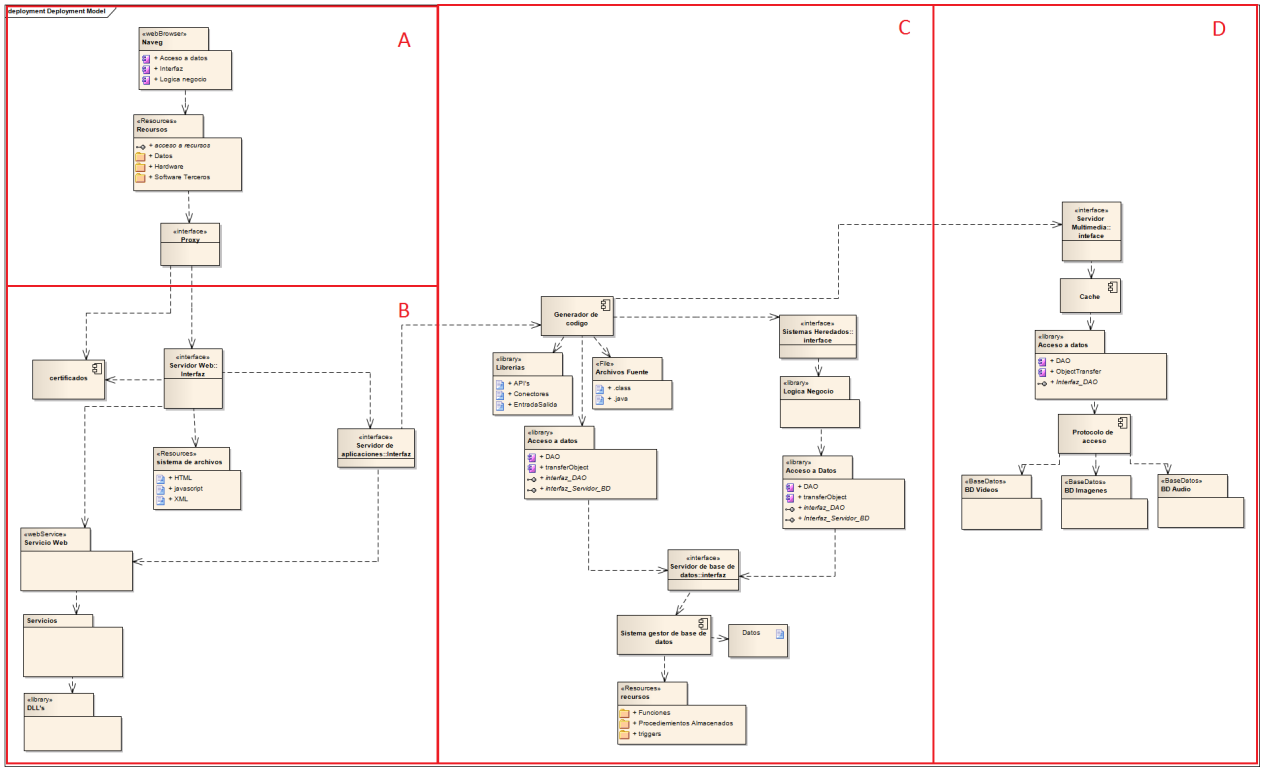

Figura 5.28: Vista de desarrollo de la aplicación Web

En la Figura 5.29 se muestran los componentes que pertenecen al nivel del cliente.

Por otro lado, en la Figura 5.30 se muestra el servidor Web, el servidor de certificados digitales y de componentes.

En la Figura 5.31 se observa al servidor de aplicaciones, sistemas heredados y de persistencia.

Por último, en la Figura 5.32 se muestra la estructura del servidor multimedia.

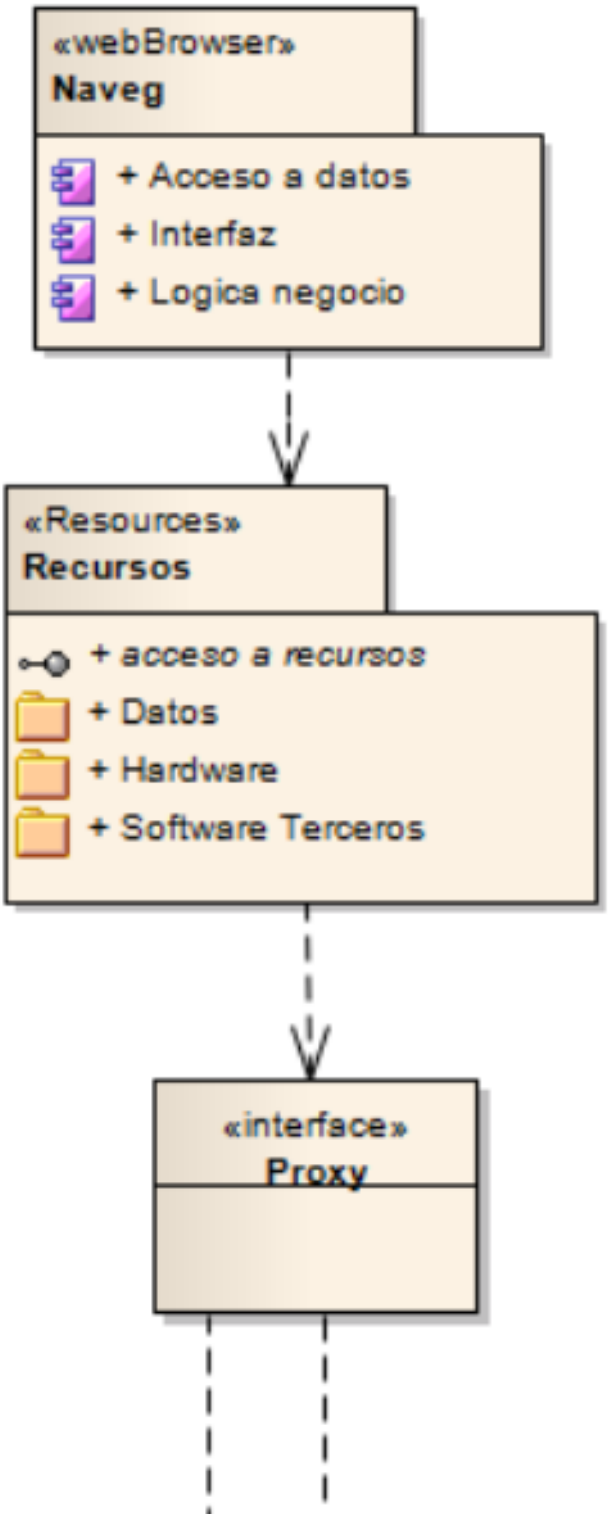

Figura 5.29: A) Vista de desarrollo de la aplicación Web

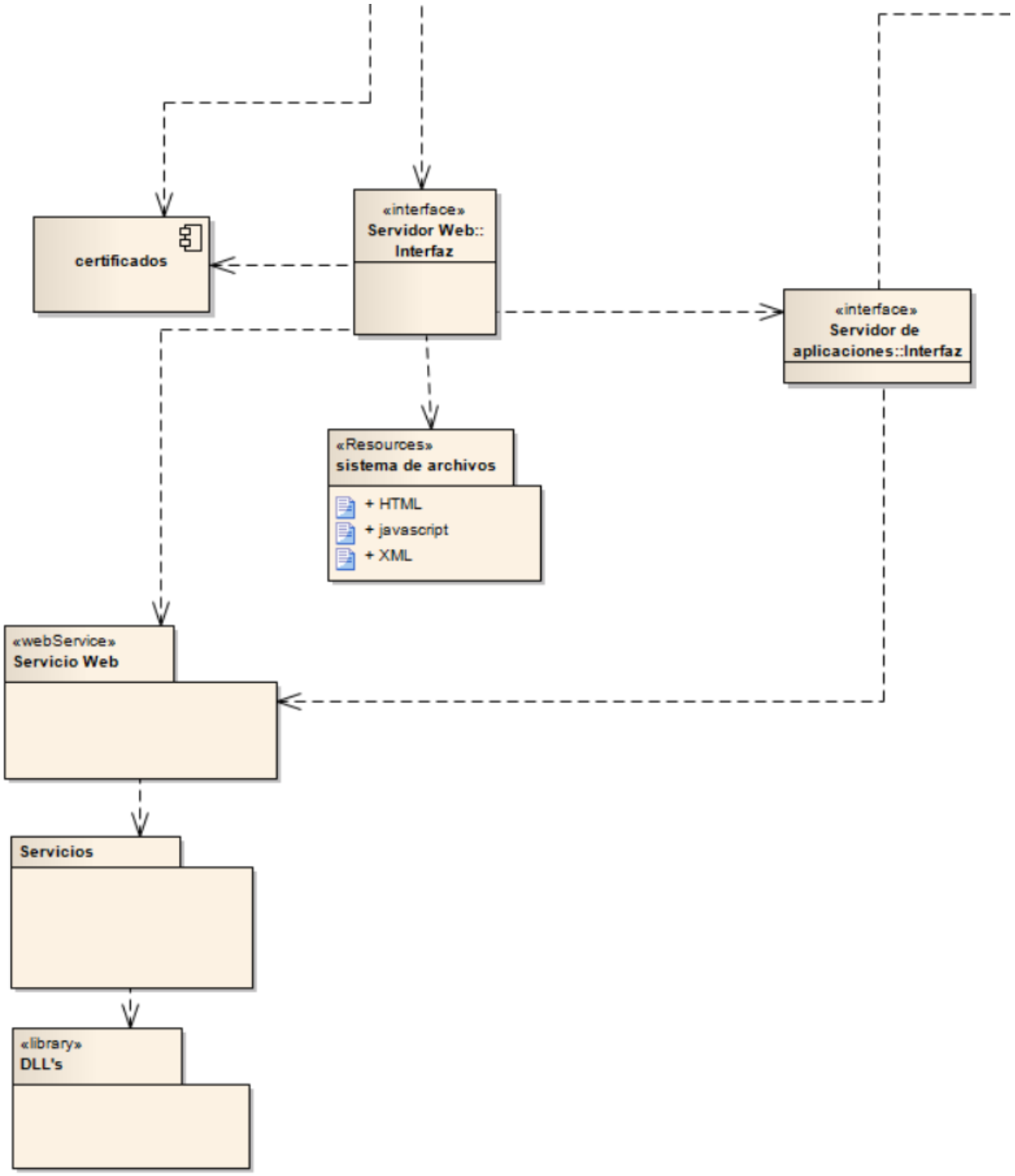

Figura 5.30: B) Vista de desarrollo de la aplicación Web

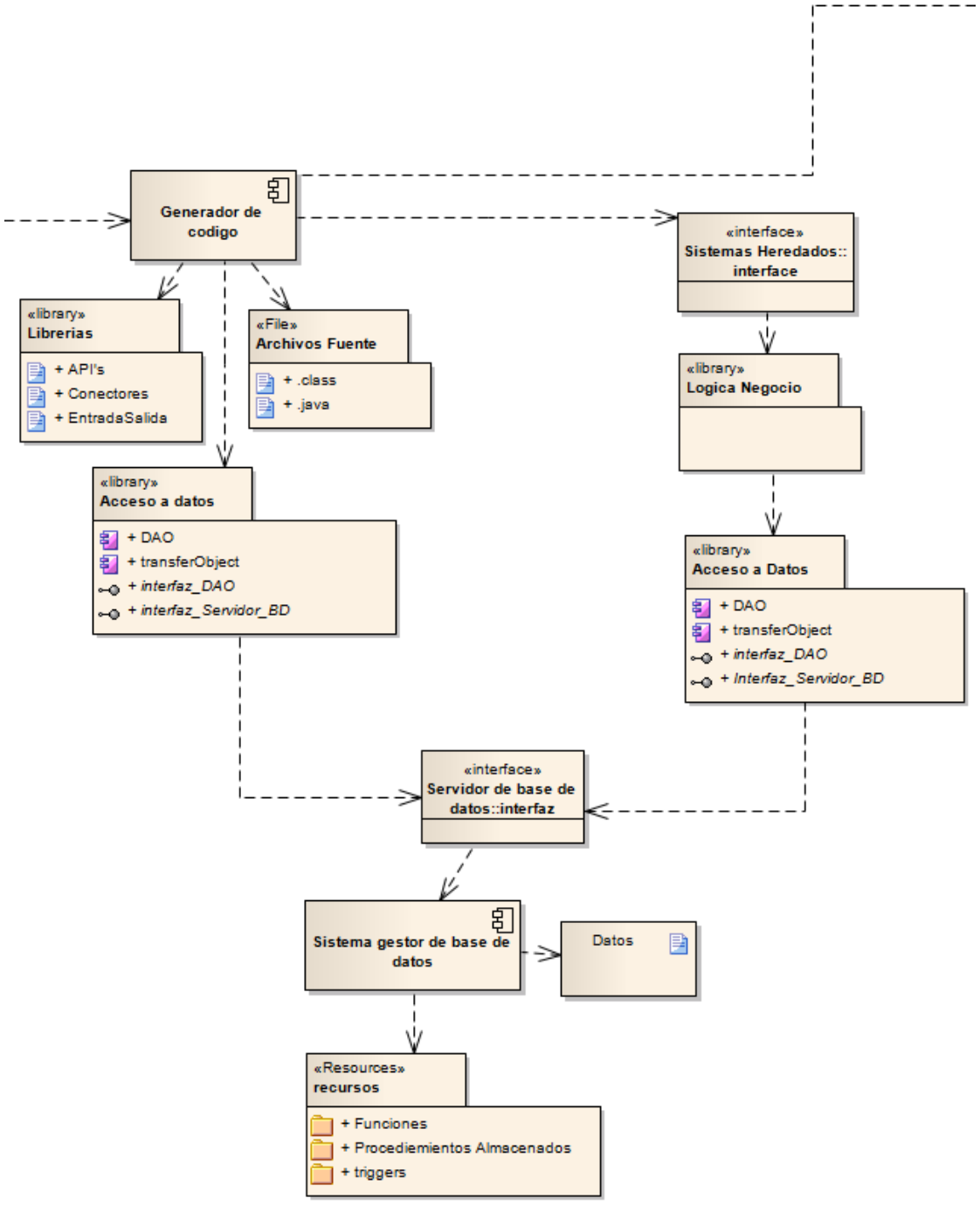

Figura 5.31: C) Vista de desarrollo de la aplicación Web

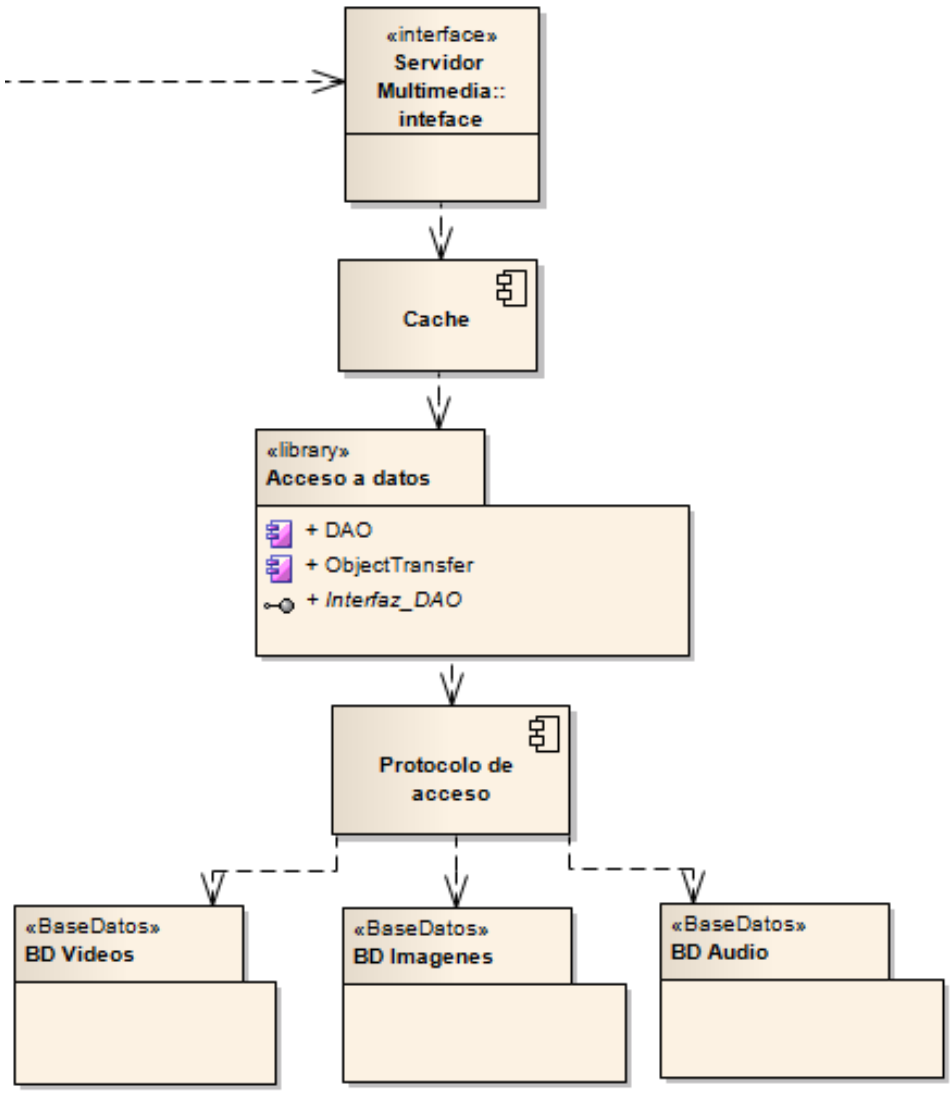

Figura 5.32: D) Vista de desarrollo de la aplicación Web

5.7.5. Vista Física

La vista Física de nuestra aplicación se muestra en la Figura 5.33. En esta vista tenemos la distribución de los componentes mostrados en la vista de desarrollo a componentes físicos tales como servidores y computadoras personales.

Dentro de esta vista se puede observar en donde van a estar cada componente. El objetivo principal de esta vista es brindar una guía al momento de desplegar nuestra aplicación dentro de nuestra infraestructura.

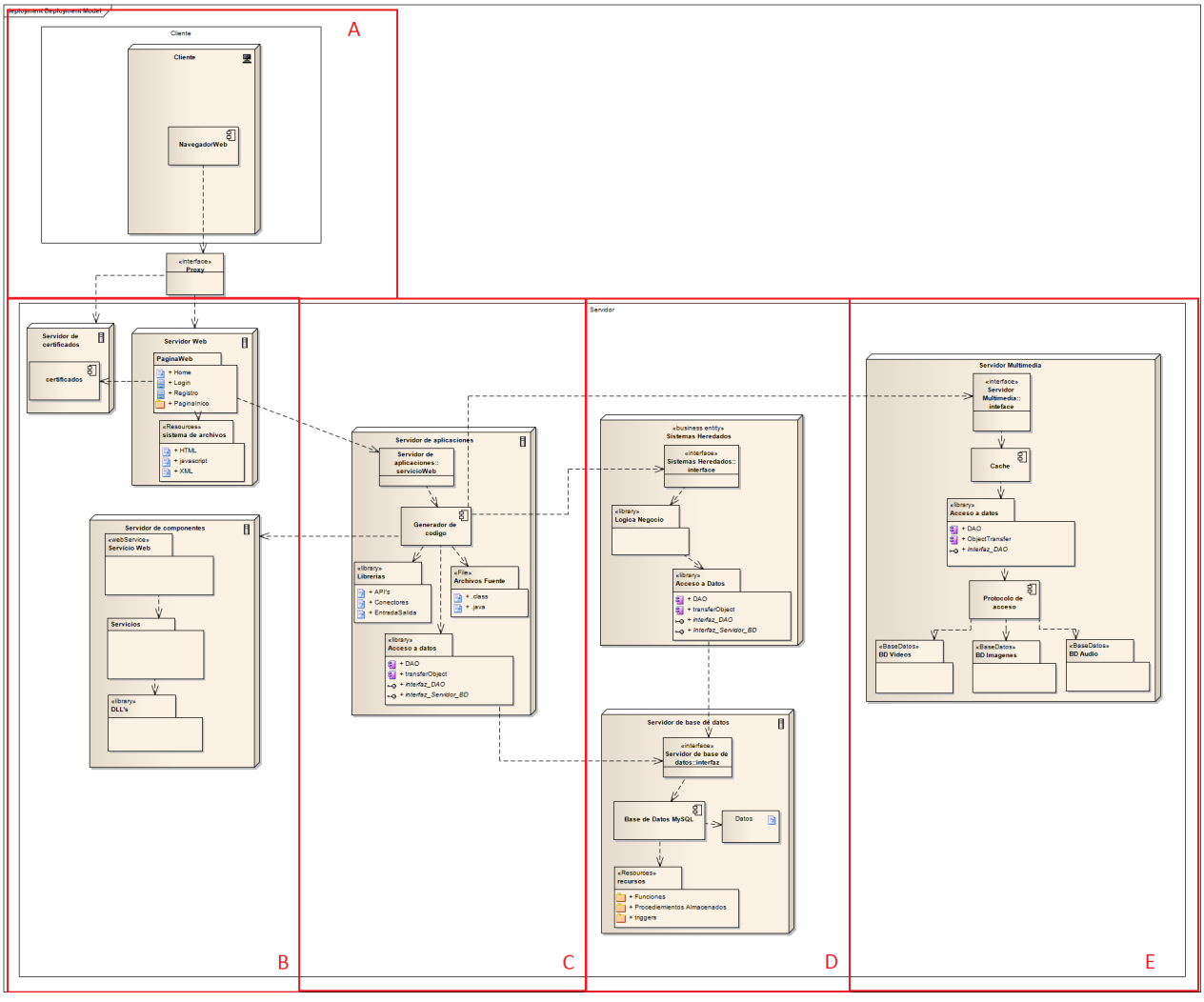

Figura 5.33: Vista Física de la aplicación Web

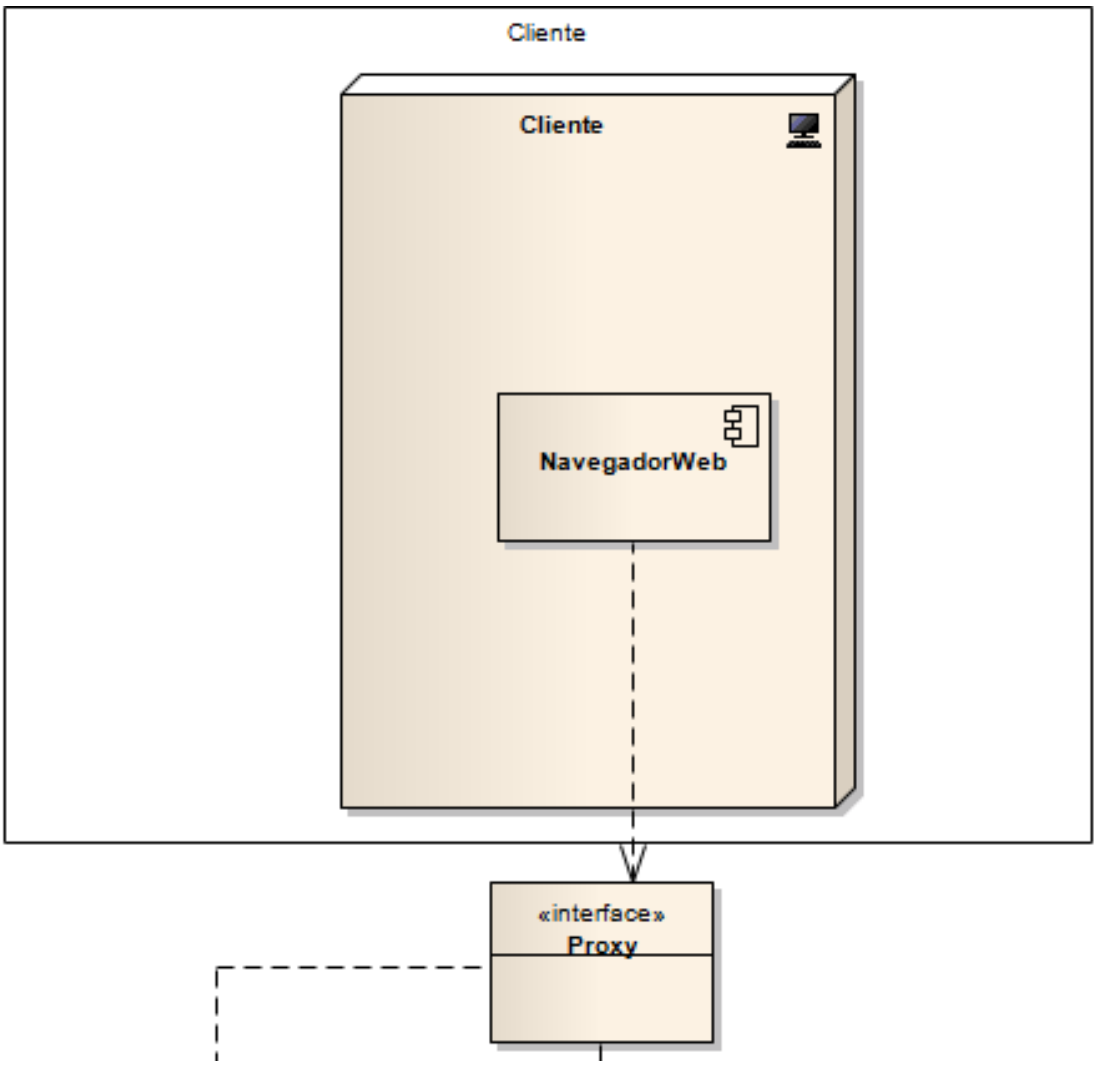

Figura 5.34: A) Vista Física de la aplicación Web

5.8. Resumen

En este caso de estudio se desarrollo la arquitectura de software para una aplicación Web para la administración de centros educativos. Se utilizo el modelo de 4+1 vistas agregando la vista de seguridad. La primera etapa del proceso de ingeniera de software fue la obtención y análisis de los requerimientos, obteniendo la vista de escenarios. El modelo para esta vista fueron los diagramas de casos de uso y de secuencia respectivamente.

Después de analizar cada uno de los requerimientos, realizamos la vista lógica

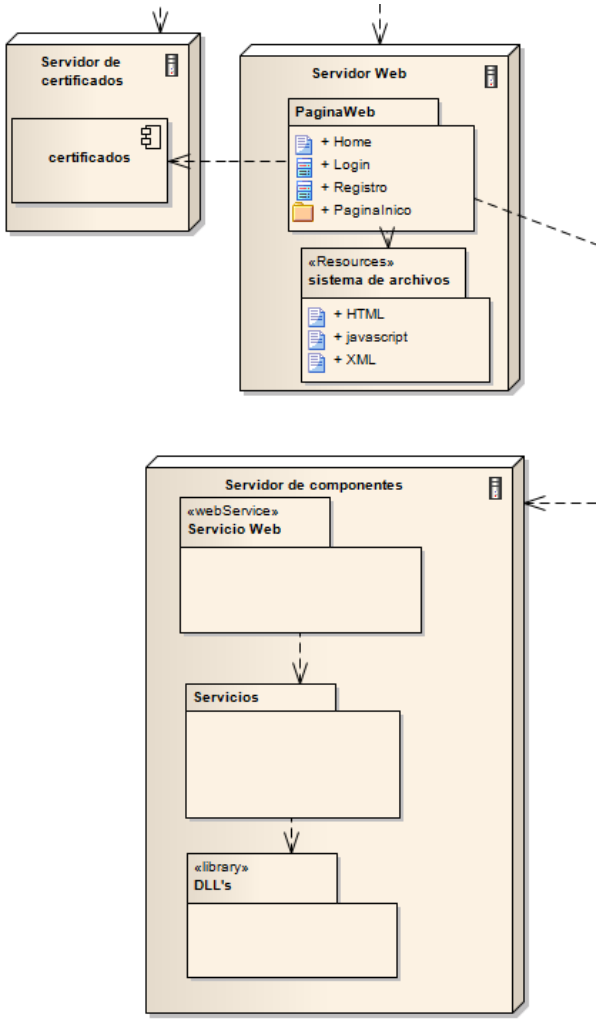

Figura 5.35: B) Vista Física de la aplicación Web

que muestra las capas de la aplicación y los componentes que contiene. La vista de desarrollo detalla cada una de las capas y de los componentes llegando hasta las clases y objetos del sistema. La vista de seguridad muestra la restricción de cada usuario al utilizar la aplicación. La vista de procesos muestra interoperabilidad del sistema con el usuario para la realización de una tarea específica. Para finalizar, la vista física muestra a la aplicación desplegada en los componentes físicos.

Se obtiene la documentación de la arquitectura que refleja los requerimientos de la aplicación.

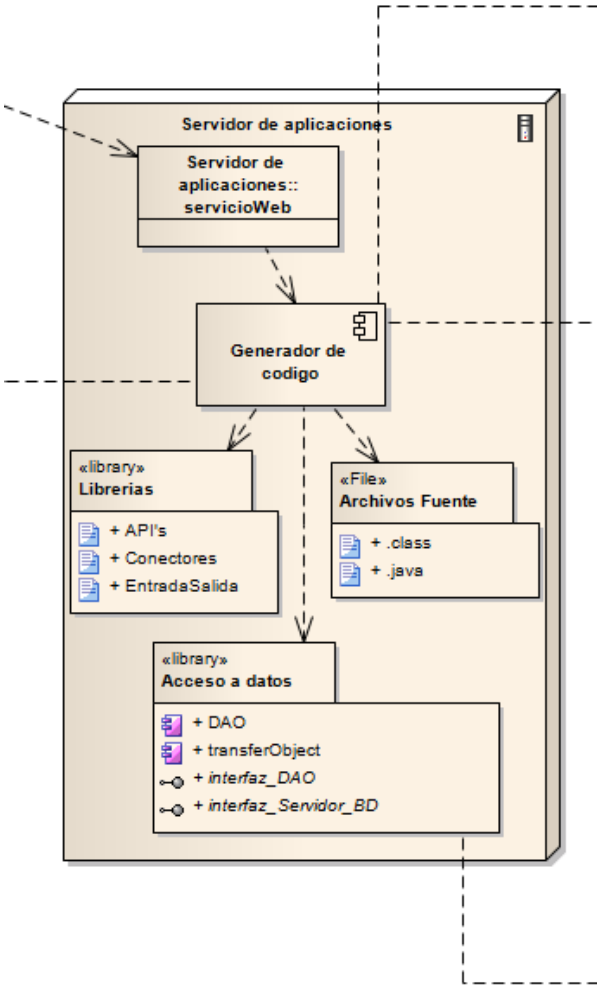

Figura 5.36: C) Vista Física de la aplicación Web

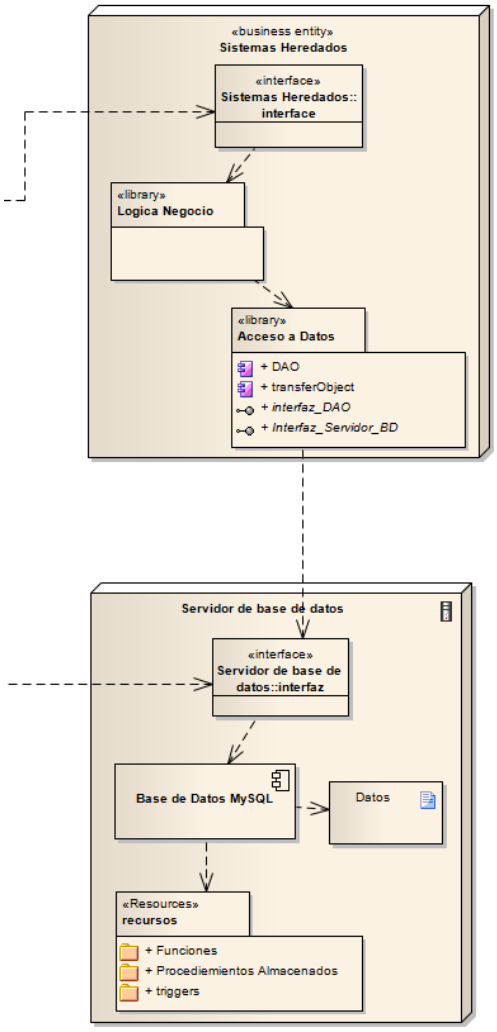

Figura 5.37: D) Vista Física de la aplicación Web

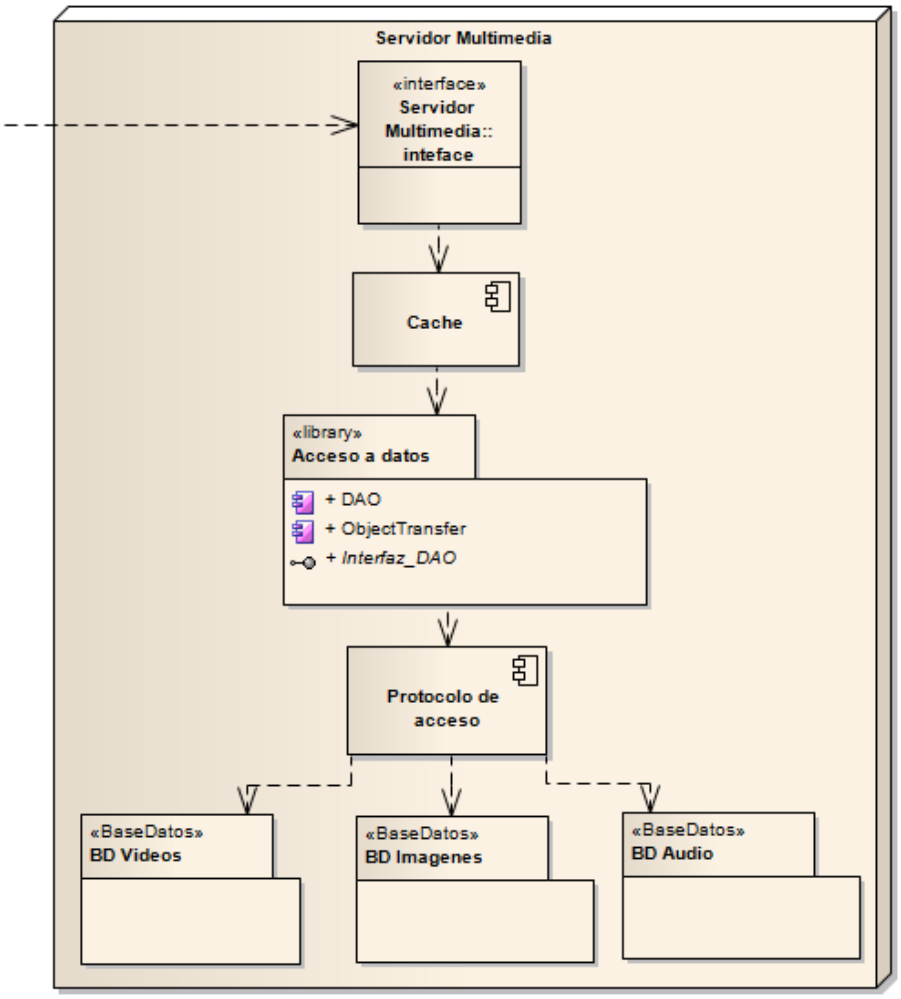

Figura 5.38: E) Vista Física de la aplicación Web

Capítulo 6

Conclusiones y Trabajo Futuro

En este Capítulo se resumen todas las ideas principales mencionadas a lo largo de la tesis para dar una conclusión acerca del trabajo realizado, también se hace una evaluación de los logros obtenidos y se menciona el trabajo que queda por hacer en el campo de las arquitecturas de software para las aplicaciones Web.

6.1. Conclusiones

La principal contribución de nuestro trabajo de investigación consiste en el desarrollo de una arquitectura de software para aplicaciones basadas en Web siguiendo un modelo de ingeniería de software. Otra contribución es la especificación de dicha arquitectura, cada una de las vistas fueron documentadas utilizando el lenguaje UML.

Para el desarrollo de este trabajo de tesis se propuso el uso del modelo en cascada, para definir la arquitectura de software, debido a que describe cada una de las etapas para el desarrollo de productos de software. Nos enfocamos en la etapa de diseño del modelo en cascada. De acuerdo al modelo en cascada el prerrequisito de la etapa de diseño es la especificación de los requerimientos de la aplicación (resultado de la etapa de requerimientos).

Utilizamos el modelo de 4+1 vistas definir a la arquitectura por medio de 5 distintas vistas, tales como: vista de escenarios, lógica, de procesos, de desarrollo y vista

física. Estas vistas tienen como propósito modelar los requerimientos de acuerdo a la función que desempeñen. Se adapto el modelo para ser utilizado con aplicaciones Web incluiyendo una vista de seguridad.

Para utilizar este modelo en el trabajo mencionado se tuvo que identificar los requerimientos de las aplicaciones Web, dentro de los requerimientos se definió cada una de las vistas arquitectónicas: vista de escenarios, lógica, de procesos, de desarrollo, de seguridad, de despliegue y la vista física. La vista de seguridad es un complemento del modelo de 4+1 vistas.

Después de haber identificado los requerimientos de las aplicaciones Web, continuamos con el proceso de diseño arquitectónico. Para proponer una arquitectura inicial analizamos la capa lógica de una aplicación Web, esto nos permitió identificar las capas y niveles iníciales de la arquitectura. Con base en este análisis presentamos una arquitectura para aplicaciones Web simples. Esta arquitectura incluye los componentes esenciales que permiten desarrollar sólo aplicaciones estáticas. La limitante de la arquitectura desarrollada eran sus componentes, por tal motivo se incluyeron componentes e interfaces necesarias para el desarrollo de aplicaciones dinámicas. La arquitectura resultante soportaba a aplicaciones dinámicas.

El proceso de diseño fue acompañado por el patrón de diseño multicapa. Este patrón nos permitió agregar componentes e interfaces a la arquitectura siguiendo una buena distribución y bajo acoplamiento. El resultado del proceso de diseño es una arquitectura que incluye todos los componentes e interfaces necesarios para el desarrollo a aplicaciones Web robustas. Además, la arquitectura pretende ser lo más genérica posible, adaptándose a las necesidades de la mayoría de las aplicaciones y acercándose cada vez más a la arquitectura final de estas aplicaciones.

En el caso de estudio se desarrolló la arquitectura siguiendo el modelo en cascada, este modelo nos permitió apreciar claramente cada una de las etapas y en especial la etapa del diseño. La arquitectura de software propuesta en este trabajo brinda la infraestructura necesaria para que aplicaciones de diferentes dominios puedan ser desarrolladas sin ningún problema. Durante el desarrollo se complemento el modelo

de 4+1 vistas con la vista de seguridad obteniendo un modelo que cumple con la mayoría de los requerimientos de los sistemas basados en Web.

Se desarrollo cada una de las vistas utilizando los diagramas del lenguaje UML obteniendo la documentación de cada una de las vistas propuestas al principio del trabajo. Estas vistas mejoran el concepto de diseño arquitectónico y son una guía para futuros desarrollos en el área de diseño.

Podemos concluir que los objetivos planteados fueron alcanzados, teniendo como principal aportación una arquitectura de software de aplicaciones Web que ayudará a los individuos, involucrados con el desarrollo de aplicaciones Web, a comprender mejor el proceso de diseño arquitectónico de estas aplicaciones.

6.2. Trabajo Futuro

Como trabajo futuro se puede:

- Desarrollar una aplicación basada en Web con la arquitectura propuesta.

  La aplicación Web necesita ser lo suficientemente demandante de recursos que permitan contemplar toda la infraestructura mencionada en el trabajo de tesis. Aplicaciones con gran contenido multimedia y de comercio electrónico pueden ser consideradas debido a la gran cantidad de usuarios que atienden. La viabilidad del desarrollo de este tipo de aplicaciones necesita ser analizada con detenimiento ya que implica destinar una gran cantidad de recursos de software (lenguajes de programación, gestores de bases de datos, protocolos) y hardware (servidores, acceso a internet).

- Desarrollar la arquitectura siguiendo otras metodologías de desarrollo de software tales como RUP, programación extrema y desarrollo ágil: La elección de una metodología de desarrollo adecuada permitirá definir la arquitectura de forma adecuada, mejorando el desarrollo de software. Para desarrollar la arquitectura con otras metodologías es necesario identificar las etapas del proceso de

diseño y adaptarlas a las etapas de las nuevas metodologías. La aportación de este estudio será analizar qué metodología se adapta mejor para el desarrollo de aplicaciones Web.

Bibliografía

[1] J. Conallen. Modeling web application architectures with uml. *Communications of the ACM*, 42(10):63–70, 1999.

[2] Ahmed E. Hassan and Richard C. Holt. Architecture recovery of web applications. *Software Engineering, International Conference on*, 0:349–366, 2002.

[3] Grady Booch. The architecture of web applications. *DeveloperWorks: IBM developer solutions*, 2001.

[4] I. Sommerville. *Ingeniería del software*. Pearson Educación, 2005.

[5] R.N. Taylor and A. Van Der Hoek. Software design and architecture the once and future focus of software engineering. *Future of Software Engineering, 2007. FOSE'07*, pages 226–243, 2007.

[6] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[7] RN Taylor, N. Medvidovic, and EM Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[8] D. Garlan, S.W. Cheng, A.C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[9] J.C. Georgas, E.M. Dashofy, and R.N. Taylor. Architecture-centric development: a different approach to software engineering. *Crossroads*, 12(4):1–6, 2006.

[10] D. Garlan and M. Shaw. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1:1–40, 1993.

[11] M.D. Jacyntho, D. Schwabe, and G. Rossi. A software architecture for structuring complex web applications. *Journal Web Engineering*, 1(1):37–60, 2002.

[12] S. Ziemer. An architecture for web applications. essay in dif 8914 distributed information systems. Technical report, Norwegian University of Science & Technology, 2002.

[13] R.T. Fielding and R.N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[14] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[15] Dilip Soni, Robert L. Nord, and Christine Hofmeister. Software architecture in industrial applications. *Software Engineering, International Conference on*, 0:196–207, 1995.

[16] R.C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2003.

[17] J. Highsmith. *Agile software development ecosystems*. Addison-Wesley, 2002.

[18] C. Alexander. *The timeless way of building*. Oxford University Press, USA, 1979.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley, 1995.

[20] R.T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, 2002.

[21] F. Buschmann, K. Henney, and D.C. Schmidt. *Pattern-oriented software architecture: On patterns and pattern languages*. John Wiley & Sons Inc, 2007.

[22] J. Noack, H. Mehmaneche, H. Mehmaneche, and A. Zendler. Architectural patterns for web applications. Technical report, 1Informatikzentrum der Sparkassenorganisation (SIZ) Bonn/Germany.

[23] Avraham Leff and James T. Rayfield. Web-application development using the model/view/controller design pattern. *Enterprise Distributed Object Computing Conference, IEEE International*, 0:118–127, 2001.

[24] G.E. Krasner and S.T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-oriented programming*, 1(3):26–49, 1988.

[25] P.C. Clements. A survey of architecture description languages. In *Proceedings of the 8th international workshop on software specification and design*, pages 1–16. IEEE Computer Society, 1996.

[26] D. Garlan, R. Monroe, and D. Wile. Acme: An architecture description interchange language. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, pages 7–22. IBM Press, 1997.

[27] P. Mishra and N. Dutt. Architecture description languages. *IEEE proc. Computers and Digital Techniques*.

[28] J. Conallen. *Building Web applications with UML*. Addison-Wesley Professional, 2003.

[29] C. Larman. *UML y Patrones*. Prentice-Hall, 2002.

[30] M. Jørgensen and K. Moløkken-Østvold. How large are software cost overruns? a review of the 1994 chaos report. *Information and Software Technology*, 48(4):297–301, 2006.

[31] D.E. Perry and A.L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[32] J. Li, J. Chen, and P. Chen. Modeling web application architecture with uml. In *Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia'00)*, pages 265–275. IEEE Computer Society, 2000.

[33] N. David and C. Carstea. Web applications architecture. *Review of the"Henry Coanda."Air Force Academy, The Scientific Informative Review*, (1):1842–9238, 2008.

[34] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. Uml-based web engineering. *Web Engineering: Modelling and Implementing Web Applications*, pages 157–191, 2008.

[35] L. Fuentes and A. Vallecillo. Una introducción a los perfiles uml. *Novática*, 168:6–11, 2004.

[36] E. Sánchez and P. Jorge. Diseño de aplicaciones web con uml y arquitecturas de software: Aplicación en dos proyectos basados en tecnología j2ee. In *Conferencia Ibero-Americana WWW/Internet 2004*, pages 513–518. IADIS, 2004.

[37] C. Hofmeister, RL Nord, and D. Soni. Describing software architecture with uml. In *Software architecture: TC2 first Working IFIP Conference on Software Architecture (WICSA1): 22-24 February 1999, San Antonio, Texas, USA*, pages 145–159. Kluwer Academic Pub, 1999.

[38] R.S. Pressman, T.G. Lewis, B. Adida, E. Ullman, T. DeMarco, T. Gilb, B.C. Gorda, W.S. Humphrey, and R. Johnson. Can internet-based applications be engineered. *IEEE Software*, 15(5):104–110, 1998.

[39] I. Gorton. *Essential software architecture*. Springer Berlin, 2006.

[40] S.V.H. Gil and S. Victoria. Representación de la arquitectura de software usando uml.

[41] N. Bieberstein, M. Fiammante, K. Jones, and R. Shah. *Service-oriented architecture compass: business value, planning, and enterprise roadmap.* Prentice Hall, 2006.

[42] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting software architectures: views and beyond.* Pearson Education, 2002.