SOFTWARE DESIGN DESCRIPTION



Historial De Cambios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión | Fecha | Sección del documento modificada | Descripción de cambios (corta) | Responsable (S) |
| 0.1.0 | 24/04/2010 | Sección 1, 2.1, 2.2 | Documentación de dichas secciones | Laura Arias, Gerente de proyectos; David Suarez, Director de calidad y manejo de riesgos. |
| 0.1.1 | 25/04/2010 | Sección 3, 4,5 y 6 | Corrección de la numeración del documento | Andrea Fajardo, Arquitecta, Germán Morales, Director de desarrollo |
| 0.2.0 | 25/04/2010 | Sección 2.4.1, 2.4.2 | Documentación de dichas secciones | Laura Arias, Gerente de proyectos. |
| 0.3.0 | 25/04/2010 | Sección 3 | Documentación de dicha sección | Andrea Fajardo, Arquitecta |
| 0.4.0 | 26/04/2010 | Sección 2.3 y 2.5 | Documentación de dichas secciones | David Suarez, Director de Calidad y Manejo de Riesgos |
| 0.5.0 | 26/04/2010 | Sección 2.4.1, Referencias bibliográficas 1 - 5 | Documentación | Laura Arias, Gerente de proyectos. |
| 0.6.0 | 27/04/2010 | Sección 2.4.2 | Documentación de dicha sección | Laura Arias, Gerente de proyectos. |
| 0.7.0 | 27/04/2010 | Sección 3 y referencias | Documentación y corrección de las referencias | Andrea Fajardo, Arquitecta |
| 0.7.1 | 28/04/2010 | Sección 3 y referencias | Corrección referencias y redacción | Andrea Fajardo, Arquitecta |
| 0.8.0 | 01/05/2010 | Sección 3 | Cambio diagrama de componentes y documentación de las interfaces | Andrea Fajardo, Arquitecta |
| 0.8.1 | 2/05/2010 | Todo el documento | Revisión de calidad | David Suárez, Director de calidad |
| 0.9.0 | 2/05/2010 | Sección 3 | Documentación acerca del patrón Fachada | Andrea Fajardo, Arquitecta |

Tabla 1: Historial de cambios

Tabla de Contenido

[1. Introducción 6](#_Toc260133508)

[1.1 Descripción Del Sistema 6](#_Toc260133509)

[1.2 Mapa del Diseño 6](#_Toc260133510)

[1.3 Referencias Bibliográficas 7](#_Toc260133511)

[1.4 Definiciones, Acrónimos y Abreviaciones 7](#_Toc260133512)

[2. Consideraciones de Diseño 8](#_Toc260133513)

[2.1 Suposiciones 8](#_Toc260133514)

[2.2 Restricciones 8](#_Toc260133515)

[2.3 Entorno del Sistema 8](#_Toc260133516)

[2.4 Metodología de Diseño 9](#_Toc260133517)

[2.4 Metodología de Diseño 9](#_Toc260133518)

[2.5 Riesgos 14](#_Toc260133519)

[3. Arquitectura 15](#_Toc260133520)

[3.2 Diagrama de Componentes 17](#_Toc260133521)

[3.2.1Subsistema Cliente 19](#_Toc260133522)

[3.2.1.1 COMPONENTE COMUNICACIÓN CLIENTE 20](#_Toc260133525)

[3.2.1.2 COMPONENTE COORDINADOR 20](#_Toc260133526)

[3.2.1.3 COMPONENTE GUI 21](#_Toc260133527)

[3.2.2 Subsistema Servidor 21](#_Toc260133528)

[3.2.2.1 COMPONENTE COMUNICACIÓN SERVIDOR 22](#_Toc260133530)

[3.2.2.2 COMPONENTE LÓGICA 22](#_Toc260133531)

[3.2.2.3 COMPONENTE PERSISTENCIA 22](#_Toc260133532)

[3.3 Estrategias de Diseño 23](#_Toc260133533)

[4. Diseño de Alto Nivel 24](#_Toc260133534)

[4.1 Diagrama de despliegue 24](#_Toc260133535)

[4.2 Diagrama de Comportamiento e Interacción 27](#_Toc260133538)

[5. Diseño de Bajo Nivel 30](#_Toc260133541)

[5.1 Subsistema 1 33](#_Toc260133542)

[Componente 1 33](#_Toc260133543)

[6. Diseño de Interfaces de Usuario 37](#_Toc260133544)

[6.1 Diseño general de la aplicación 37](#_Toc260133545)

[6.2 Árbol de navegabilidad 38](#_Toc260133546)

[Anexos 39](#_Toc260133547)

Lista de Tablas

[Tabla 1: Historial de cambios 1](#_Toc180071443)

[Tabla 2: Ventajas y Desventajas de Estilos Arquitectónicos 16](#_Toc180071444)

[Tabla 3: Subsistema 1 19](#_Toc180071445)

[Tabla 4: Componente 1 19](#_Toc180071446)

[Tabla 5: Estrategias de Diseño 20](#_Toc180071447)

[Tabla 6: Nodo 1 23](#_Toc180071448)

[Tabla 7: Conector 1 23](#_Toc180071449)

[Tabla 8: Documentación de clases 29](#_Toc180071450)

[Tabla 9: Descripción de entradas y salidas 32](#_Toc180071451)

Lista de Ilustraciones

[Ilustración 1: Descripción del sistema 6](#_Toc180071452)

[Ilustración 2: Mapa de Diseño 7](#_Toc180071453)

[Ilustración 3: Suposiciones generales 11](#_Toc180071454)

[Ilustración 4: Restricciones 12](#_Toc180071455)

[Ilustración 5: Entorno del Sistema 13](#_Toc180071456)

[Ilustración 6: Administración de Riesgos 14](#_Toc180071457)

[Ilustración 7: Apreciación Global 16](#_Toc180071458)

[Ilustración 8: Capas Arquitectónicas 17](#_Toc180071459)

[Ilustración 9: Diagrama de componentes 17](#_Toc180071460)

[Ilustración 10: Subsistema Usuario 7 Texas Poker 18](#_Toc180071461)

[Ilustración 11: Estrategias de Diseño 20](#_Toc180071462)

[Ilustración 12: Descripción de Nodos 21](#_Toc180071463)

[Ilustración 13: Descripción Conexiones 22](#_Toc180071464)

[Ilustración 14: Nodo Usuario 7 Texas Poker 22](file:///C:\Documents%20and%20Settings\Juan%20David\Desktop\Proyecto%20Especial\SDD\SDD%201.0.docx#_Toc180071465)

[Ilustración 15: Diagrama de Actividad 24](#_Toc180071466)

[Ilustración 16: Diagrama de secuencia 25](#_Toc180071467)

[Ilustración 17: Diseño de bajo nivel 26](#_Toc180071468)

[Ilustración 18: Niveles de diseño 27](#_Toc180071469)

[Ilustración 19: Ejemplo diseño de bajo nivel 28](#_Toc180071470)

[Ilustración 20: Ejemplo Descripción de Clase 29](#_Toc180071471)

[Ilustración 21: Ejemplo Resumen de Métodos 30](#_Toc180071472)

[Ilustración 22: Ejemplo descripción de método 31](#_Toc180071473)

[Ilustración 23: Diseño general de la aplicación 32](#_Toc180071474)

[Ilustración 24: Ejemplo árbol de navegabilidad 33](file:///C:\Documents%20and%20Settings\Juan%20David\Desktop\Proyecto%20Especial\SDD\SDD%201.0.docx#_Toc180071475)

# 1. Introducción

### 1.1 Descripción Del Sistema

El objetivo del presente documento es brindar una descripción detallada del diseño y arquitectura de la aplicación T-Monopoly® el cual se encuentra en proceso de implementación por parte de los integrantes de Alimnova®. El propósito principal del SDD consiste en representar un sistema de software; este documento permite comunicar el diseño de software[1].

La sección 3 presenta a nivel detallado el proceso que se llevo a cabo para determinar la arquitectura que corresponde al diseño de alto nivel de T-Monopoly® [[**sección 3**](#_Arquitectura)]. Alimnova® selecciono 2 tipos de arquitectura. La primera y principal es la arquitectura Cliente-Servidor que permite la conexión de varios usuarios a un servidor que contiene la información de la aplicación. La segunda arquitectura corresponde a modelo-vista-controlador que está asociada a los subsistemas cliente y servidor, donde el controlador y la vista se encuentran en el subsistema cliente y el modelo es responsable de manejar la lógica del juego y la persistencia de datos, este componente se encuentra en el subsistema servidor; de esta manera el servidor es pesado y el cliente es liviano [[**sección 3.2**](#_Diagrama_de_Componentes)].

La forma en que Alimnova® manejara la comunicación es por medio de RMI (Remote Method Invocation), el cual permite invocar métodos de manera remota desde cualquier computador.

### 1.2 Mapa del Diseño

El diseño de software es importante para el mantenimiento de software, de manera que si se realiza algún cambio o si se agrega una nueva característica no se desestabilice el software[2]. Existen 2 tipos de diseño:

* Diseño de alto nivel: Es el primer paso que se realiza en el diseño de software, donde se selecciona una arquitectura que satisfaga los requerimientos funcionales y no funcionales. Un buen diseño de alto nivel lleva a un mejor diseño de bajo nivel[2]. El diseño de alto nivel se compone de:
  + Arquitectura que se adapte a las necesidades de la aplicación T-Monopoly® [[**sección 3**](#_Arquitectura)].
  + Diagrama de despliegue el cual muestra las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos [3][[**sección 4.1**](#_Diagrama_de_despliegue)].
  + Diagramas dinámicos permite validar las interacciones entre los distintos objetos.
* Diseño de bajo nivel: también denominado modulo de diseño considerar donde se debe considerar el lenguaje de programación a usar en la implementación de manera que se determinen los tipos de interfaces[2], para el caso de Alimnova® el lenguaje de programación será Java [[**sección 5**](#_Diseño_de_Bajo)].

### 1.3 Referencias Bibliográficas

1. IEEE Std. 1016-1998, IEEE Recommended Practice for Software Design Descriptions, IEEE Computer Society.
2. Consulta relacionada con diseño de software, <http://www.the-software-experts.de/e_dta-sw-design.htm>, última consulta 24/04/2010.
3. Consulta relacionada con diagrama de despliegue, virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc, última consulta 24/04/2010.
4. S. Albin, The Art of Software Architecture: Design Methods and Techniques, John Wiley & Sons 2003.
5. L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison Wesley, 2003.
6. K. Barclay, J. Savage, Object-Oriented Design with UML and Java, El Sevier Butterworth Heinemann, 2004.
7. Consulta de diagrama de clases. Disponible: <http://www.agilemodeling.com/artifacts/classDiagram.htm#Classes>, última consulta: 26/04/2010.
8. A. Shalloway, J. Trott, Design Patterns Explained: A New Perspective on Object Oriented Design.
9. Consulta relacionada con patrones, <http://www.oodesign.com/proxy-pattern.html>, última consulta 27/04/2010.
10. Consulta relacionada con patrones, <http://home.earthlink.net/~huston2/dp/state.html>, última consulta 27/04/2010.
11. Patrón Observer. Disponible: [http://www.proactiva-calidad.com/java/patrones/observador.html. Consultado el 25/04/2010](http://www.proactiva-calidad.com/java/patrones/observador.html.%20Consultado%20el%2025/04/2010)
12. Patrones de diseño. Disponible: <http://patronesdediseno.blogspot.com/2009/05/patron-observer.html>. Visitado el 27/04/2010
13. Patrones de diseño. Disponible: [http://patronesdediseno.blogspot.com/2009/05/patron-state.html . Visitado el 25/04/2010](http://patronesdediseno.blogspot.com/2009/05/patron-state.html%20.%20Visitado%20el%2025/04/2010).
14. Patron state, patrón Bridge. Biel Massot Puigserver y Edu Herraiz Aparicio. Disponible: [http://dmi.uib.es/~yuhua/APOO07-08/Presentation/Bridge-State.pdf. Visitado el 25/04/2010](http://dmi.uib.es/~yuhua/APOO07-08/Presentation/Bridge-State.pdf.%20Visitado%20el%2025/04/2010).
15. Definición arquitectura cliente servidor. Disponible: [http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml Visitado el 28/04/2010](http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml%20Visitado%20el%2028/04/2010)
16. Garlan D. et al, An Introduction to Software Architecture, Enero 1994, Carnegie Mellon University
17. Wapedia.Proxy patron de diseño. Disponible: <http://wapedia.mobi/es/Proxy_(patr%C3%B3n_de_dise%C3%B1o)>
18. Patrón Proxy. Raúl Heras, Alberto Blasco , José Manuel Arévalo. Disponible: <http://www.google.com.co/url?sa=t&source=web&ct=res&cd=2&ved=0CAoQFjAB&url=http%3A%2F%2Fkybele.escet.urjc.es%2Fdocumentos%2FSI%2FPatrones%2F18_Proxy.ppt&rct=j&q=proxy+patron+&ei=X13cS4aCIJCg8ATJvfmvBw&usg=AFQjCNFoc9pclXEPdCrVjWTk00Rc2cZwrA>.
19. Diseño de software con patrones. Java Hispano Disponible: <http://www.javahispano.org/contenidos/es/diseno_de_software_con_patrones_parte_4/> Visitada el 02/05/2010.

### 1.4 Definiciones, Acrónimos y Abreviaciones

|  |  |
| --- | --- |
| LETRA | DEFINICIONES |
| A | * Arquitectura de software: es una extensión del proceso de diseño de ingeniería. Este se centra en la descomposición de una sistema en componentes y las interacciones entre ellos para satisfacer requerimientos funcionales y nos funcionales[4]. |
| B |  |
| C | * Componente: Definición de lo que existe en un modulo[4]. |
| D |  |
| E |  |
| F |  |
| G |  |
| H |  |
| I |  |
| J |  |
| K |  |
| L |  |
| M | * Módulo: unidad de software que puede ser diseñada, implementada y compilada en un sistema ejecutable[4]. |
| N |  |
| O |  |
| P |  |
| Q |  |
| R | * RMI: Remote Method Invocation |
| S | * SDD: Software Design Description * SAD: Software Architecture Document |
| T |  |
| U |  |
| V |  |
| W |  |
| X |  |
| Y |  |
| Z |  |

Tabla 2: Definiciones y Acrónimos

# 2. Consideraciones de Diseño

### 2.1 Suposiciones

* El diagrama de clases tendrá aplicados patrones con el fin de garantizar un buen diseño de bajo nivel.
* El equipo donde se instalara la aplicación deberá tener la maquina virtual JRE versión 6 Update 16.
* El equipo de desarrollo cuenta con las herramientas necesarias para el diseño y la implementación de la aplicación T-Monopoly®. Las herramientas estas descritas en el SPMP y el SRS Línea Alimnova®.
* El equipo de desarrollo utilizo las librerías de Look and Feel para el manejo de interfaz grafica.

### 2.2 Restricciones

* La resolución de la pantalla donde se instale la aplicación será de 1000x800 pixeles.
* Para restricciones de Hardware ver SRS Línea Base Alimnova®.
* El equipo donde se instale la aplicación deberá contar con parlantes que sean compatibles con dicho equipo.
* La aplicación deberá tener una arquitectura cliente-servidor.
* La aplicación contará con interfaz gráfica fuerte.
* La aplicación deberá manejar persistencia de datos.

### 2.3 Entorno del Sistema

La aplicación T-Monopoly® no está ligada a ningún otro sistema o aplicación por lo que en esta sección se describirán únicamente el entorno a nivel de software y hardware que este posee.

A nivel de software T-Monopoly® requiere de varias especificaciones que son mencionadas en la sección 3.3.2.2 del documento SRS[Alimnova]LineaBaseV3.0.0.

A nivel de hardware también debe cumplir con ciertas restricciones y especificaciones, mencionadas en la sección 3.3.2.1 del documento SRS[Alimnova]LineaBaseV3.0.0.

### 2.4 Metodología de Diseño

El diseño de software se maneja a 2 niveles: diseño de alto nivel y diseño de bajo nivel [[**sección 1.2**](#_1.2_Mapa_del)]. Esta sección presenta el proceso llevado a cabo por Alimnova® para definir de manera adecuada y coherente ambos diseños.



Ilustración 2: Proceso para definición de diseño de alto nivel y bajo nivel. Adaptado de [4].

### 2.4 Metodología de Diseño

El diseño de software se maneja a 2 niveles: diseño de alto nivel y diseño de bajo nivel [[**sección 1.2**](#_1.2_Mapa_del)]. Esta sección presenta el proceso llevado a cabo por Alimnova® para definir de manera adecuada y coherente ambos diseños.



Ilustración 2: Proceso para definición de diseño de alto nivel y bajo nivel. Adaptado de [4].

##### 2.4.1 Arquitectura

2.4.1.1 Entendimiento del Problema

Muchos proyectos de software fallan porque crean soluciones para un problema de negocio inválido[4]. La tarea consiste en identificar el problema real que se desea solucionar.

Comprender los requerimientos es una técnica bastante útil para esta etapa [5]. Dado que estos ya fueron verificados con los casos de uso. Esta etapa fue realizada en el Hito 2 y especificada en el documento SRS ya que se realizó el proceso de ingeniería de requerimientos. Esta técnica permite a Alimnova® determinar la forma de la arquitectura[4]. Este proceso permite mantener la consistencia de la aplicación T-Monopoly® y la trazabilidad.

2.4.1.2 Identificar los Elementos de Diseño y las Relaciones

El primer paso es establecer la descomposición de los componentes basado en los requerimientos funcionales. La arquitectura de la aplicación de software es un conjunto de subsistemas interconectados denominados módulos[4]. Estos módulos son la construcción organizacional del código fuente en el momento de la implementación[4]. La arquitectura definida por Alimnova® después de aplicar el proceso la podrá ver más adelante en este documento [[**sección 3**](#_Arquitectura)].

2.4.1.2.1 Definir el Contexto del Sistema

Permite describir la aplicación desde una perspectiva externa (Stakeholders). En el documento SPMP hubo una descripción de todas las interfaces externas que afectarían la aplicación T-Monopoly®. La arquitectura es el resultado de un conjunto de decisiones técnicas y de negocio[5]. Las decisiones de negocio son influenciadas por los Stakeholders, para este proceso Andrea Fajardo, Arquitecto deberá tomar en cuenta las opiniones de ellos para construir la arquitectura. Sin embargo, dicha arquitectura no solo es influenciada por los Stakeholders, sino también por otros aspectos.

Ilustración 3: Influencias sobre el arquitecto para definir la arquitectura. Tomado de [5].

El ambiente técnico para la aplicación T-Monopoly® está definido en el documento SRS.

2.4.1.2.2 Identificar los Módulos

Los módulos son unidades discretas de software, generalmente se denominan componentes y conexiones [[**sección 1.3**](#_1.3_Referencias_Bibliográficas)]. Una arquitectura clásica modular es el sistema cliente-servidor el cual constituye una de las arquitecturas de la aplicación T-Monopoly® [[**sección 1.1**](#_1.1_Descripción_Del)].

Una técnica útil para establecer las conexiones entre los componentes consiste en hacer la matriz de estructura de diseño[4] y luego representar esta estructura en un diagrama de componentes [[**sección 3.2**](#_Diagrama_de_Componentes)].



Tabla 3: Matriz de estructura de diseño.

Luego de establecer el diagrama de componentes basado en la matriz anterior, se debe hacer la respectiva documentación de los componentes y los conectores.

2.4.1.3 Evaluar la Arquitectura

La evaluación del diseño se hace en medidas cualitativas o datos cuantitativos, con el fin de determinar si se satisface la arquitectura con los requerimientos. El encargado de evaluar la calidad del diagrama de componentes que representa la arquitectura será David Suárez, Director de calidad y manejo de riesgos; acompañado de Andrea Fajardo, Arquitecta. La siguiente ilustración definirá los 2 criterios para evaluar la calidad de la arquitectura propuesta.

Ilustración 4: Criterios de evaluación de la arquitectura.

Es importante que el arquitecto analice las dependencias entre los requerimientos, ya que esto permite realizar una mejor descomposición a nivel de subsistemas y componentes. Otro tema relevante consiste en que el arquitecto debe tener juicio crítico para juzgar que requerimientos implican una mayor complejidad y cuáles no[4].

Una técnica que permite evaluar estos 3 criterios se denomina ATAM (Architecture Trade-off Analysis Method)[4].

2.4.1.4 Trasformar la Arquitectura

Luego de la evaluación de la arquitectura si esta no satisface los atributos de calidad se deben realizar ciertos cambios. El diseño se transforma aplicando operadores de diseño, estilos o patrones.

Hay 2 tipos de operadores de diseño. Modulares (mejora la vista modular del sistema) y de Diseño (mejora el tiempo de ejecución del componente del sistema)[4]:

Modulares:

* División del diseño en 2 o más módulos
* Substituir un modulo de diseño por otro
* Agregar un nuevo modulo
* Sacar un modulo del sistema
* Convertir un modulo en una interfaz
* Trasladar un modulo a otro sistema

Diseño:

* Descomposición del sistema en componentes
* Replicar módulos para mejorar la confiabilidad
* Comprimir 2 o más módulos en uno para mejorar el desempeño
* Abstracción de un componente para mejorar la adaptación a los cambios o la adaptabilidad.

##### 2.4.2 Diseño de Bajo Nivel

Este diseño determinara la clase de interfaces que Alimnova® podría usar[2]. Germán Morales, director de desarrollo, será el responsable de llevar a cabo el desarrollo de esta etapa donde se debe establecer el diagrama de clases para la implementación, con la colaboración del todos los integrantes de Alimnova®. En las siguientes sub secciones se realizara una explicación de la forma en que se realizo el proceso para establecer el diagrama de clases, con su correspondiente documentación [[**sección 5**](#_5._Diseño_de)].

Un diagrama de clases describe los tipos de objetos en el sistema y las relaciones que existen entre ellos[6]. El diagrama de clases debe ser coherente con los diagramas de colaboración.

2.4.2.1 Identificación de Clases

Un objeto es una persona, lugar, cosa, concepto, evento, pantalla o reporte aplicable a la aplicación. Los objetos conocen cosas (tienen atributos) y hacen cosas (tienen métodos)[7]. Un objeto debe mantener un estado interno, donde se representan los datos privados encapsulados por el objeto[6].

2.4.2.2 Identificación de Relaciones

Los objetos se relacionan de diferentes maneras:

* Asociación
* Agregación
* Composición
* Herencia

2.4.2.3 Aplicación de Patrones

El proceso descrito en los numerales permitió establecer un diagrama de clases inicial, luego se agregaron patrones. Los patrones permiten enseñar principios de orientación a objetos, además permiten explicar por qué y que se hace con los objetos[8]. El uso de los patrones se describe en secciones posteriores [[**sección 5**](#_Diseño_de_Bajo)].

2.4.2.3.1 Observer

Define una o más dependencias entre objetos, es decir, que cuando un objeto cambie su estado, todos los objetos dependientes deberán ser notificados y actualizados automáticamente[8].

2.4.2.3.2 Proxy

El patrón de diseño Proxy es aplicable cuando existe la necesidad de controlar el acceso a un objeto, así como cuando existe la necesidad de hacer referencia a un objeto sofisticado[9].

2.4.2.3.1 State

Este patrón permite que un objeto modifique su comportamiento cuando cambia su estado interno. El problema ocurre cuando se necesita cambiar el comportamiento en tiempo de ejecución dependiendo del estado[10].

### 2.5 Riesgos

Los riesgos para este hito, están fuertemente ligados a la implementación del diseño, por lo tanto se identificaron los siguientes riesgos:

Ilustración 5: Riesgos del SDD

|  |  |  |
| --- | --- | --- |
| Riesgo | Probabilidad | Efecto |
| Retraso en la entrega del documento de diseño | Bajo | Catastrófico |
| Requerimientos no satisfechos | Media | Crítico |
| Modificación del diseño | Media | Serio |
| El diseño no cumple alguna de las restricciones impuestas por el cliente | Baja | Critico |

Tabla 4: Probabilidad y efecto de los riesgos en el SDD

|  |  |  |
| --- | --- | --- |
| **Riesgo** | **Plan de mitigación** | **Plan de contingencia** |
| **Retraso en la entrega del documento de diseño** | Buena planeación de actividades, siendo consientes de la cantidad de trabajo delegado y el tiempo para realizarlo. | Reasignación de actividades [[**ver sección 5.3.2**](#_Plan_de_Control_6)] |
| **Requerimientos no satisfechos** | Se llevara a cabo una buena planeación para la trazabilidad de los requerimientos y así evitar dejar de lado alguno de ellos. [ver sección 4.2 SRS[Alimnova]LineaBaseV3.0.0] | Se reasignaran actividades a fin de que se disponga de algún tiempo para poder cumplir con el requerimiento que no está implementado. |
| **Modificación de diseño** | Llevar a cabo un buen proceso en el análisis de la trazabilidad a fin de que cuando se deba cambiar algún requerimiento, esto se haga antes de implementarlo. | Reasignación de actividades y adaptación del nuevo diseño. |
| **El diseño no cumple aluna de las restricciones impuestas por el cliente.** | Seguir una buena trazabilidad de las restricciones que se convertirán en requerimientos y de ahí a ser implementados. [ver SRS sección 4.2] | Reasignación de actividades a fin de cumplir con todas las restricciones impuestas. |

Tabla 4: planes de mitigación y contingencia de los riesgos en el SDD

# Arquitectura

#### 3.1 Apreciación Global

Para el desarrollo del juego T-Monopoly, Alimnova® decidió usar la arquitectura Cliente/Servidor, inicialmente porque fue una de las restricciones dadas por el cliente.

Una vez se realizaron investigaciones acerca de esta arquitectura Alimnova® identificó que permite que el servidor sea quién tenga la lógica del programa y del juego como tal, permitiendo así la protección de los datos y de información importante para el desarrollo del juego, siendo esta seguridad una parte muy importante para la ejecución del juego, así como también esta arquitectura permite que el servidor sea independiente de los clientes, que no los conozca, pero que los clientes si puedan acceder al servidor para pedir determinados servicios que le ofrece, cumpliendo con las restricciones dispuestas por el [**ver anexo 8.1 SPMP[Alimnova]LineaBaseV3.0.0**].

Por otro lado esta arquitectura permite que el usuario vea el sistema de forma homogénea, ya que los cambios o actualizaciones son independientes a él, permitiendo así mantener la seguridad que todos los clientes están viendo lo mismo, reduciendo así el riesgo de presentar información errónea a todos o a algún cliente en determinado momento de la ejecución del juego, así como también mantener al cliente alejado de las actividades de la ejecución del juego a un nivel inferior de diseño. **[16]**

Otra de las ventajas que brinda usar esta arquitectura es eliminar la necesidad de mover grandes bloques de información por la red, entre diferentes equipos, lo cual permite que tan solo se envíe la información necesaria y con ello reducir el tiempo de respuesta. Los servidores son quienes manejan los datos, procesan peticiones y luego transfieren solamente los datos pedidos por determinado cliente.**[16]**

Dentro del juego T-Monopoly Alimnova® se basó en algunas restricciones dadas por el juego o por el cliente siendo estos datos determinantes para el diseño de la arquitectura, dentro de las cales estuvieron aspectos como:

* Para jugar T-Monopoly, se necesitan como mínimo 2 personas y máximo 5 jugadores , cada jugador tendrá su turno para poder realizar diferentes actividades dentro del desarrollo normal del juego. Para llevar a cabo esta característica es importante contar con un mecanismo que controle la asignación y ejecución de los turnos para cada cliente y que valide si puede realizar determinada transacción, acorde con las reglas establecidas por el [**ver anexo 8.1 SPMP[Alimnova]LineaBaseV3.0.0**].
* Debido a que durante la ejecución del juego habrán varios jugadores es importante tener un mecanismo que controle la interfaz gráfica de cada uno y que esta corresponda con la realidad de la ejecución del juego.

A partir de la información obtenida y antes expresada Alimnova® decidió tomar esta arquitectura, puesto que permite cumplir con lo propuesto , en donde existe un servidor que es un mecanismo encargado de controlar todos los clientes que ingresan a una partida, validará la información de cada jugador tanto respecto al perfil de cada uno, como las actividades llevadas a cabo durante la ejecución de T-Monopoly , siendo este el encargado de modificar o actualizar la información según sea requerido y su objetivo es gestionar toda la información necesaria para poder llevar a cabo el objetivo del juego T-Monopoly. Por otro lado el cliente corresponde a la representación del jugador humano dentro del juego; el cual estará comunicándose continuamente con el servidor mientras se esté ejecutando T-Monopoly.

Ilustración 3: Apreciación Global

### 3.2 Diagrama de Componentes

El propósito del diagrama de componentes es poder reflejar a partir de los requerimientos identificados por Alimnova®, la razón de ser de cada uno de los componentes y la relación que existe entre cada componente con la trazabilidad de cada requerimiento identificado, con el fin de implementarlos, evaluados, modificados de forma independiente.

Con el fin de mantener la claridad de este documento y el entendimiento del mismo Alimnova® dividió la aplicación en dos subsistemas, cada uno con sus respectivos componentes. Los cuales han sido documentados teniendo en cuenta la siguiente tabla:

|  |  |
| --- | --- |
| Nombre Subsistema | * *Componente N* |
| Propósito del Subsistema | *Descripción del Propósito* |
| Composición del Subsistema | * *Componente 1* * *Componente 2* * *Componente N* |

Tabla 4: Descripción Subsistemas

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente |  |  |  |
| Propósito del  Componente | *Descripción del propósito* | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *Interfaz 1* | *Componente 2* | *Descripción de la utilidad o función de la interfaz mencionada. ¿Qué servicios presta a los componentes que la utilizan?* |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *Interfaz 2* | *Componente 3* | *Descripción de la utilidad o función de la interfaz mencionada. ¿Qué servicios presta a este componente?* |

Tabla 5: Descripción Componentes

El siguiente diagrama, representa los subsistemas, las relaciones entre ellos, sus componentes y las relaciones entre estos dentro del subsistema, las cuales fueron identificadas por Alimnova® y se definió para T-Monopoly.



Ilustración 6: Diagrama de Componentes T-Monopoly

|  |  |  |
| --- | --- | --- |
| Nombre de la Interface | Componentes que la utilizan | Servicios prestados |
| ComunicaciónRMI | Cliente y Servidor | Encargada de solicitar determinado servicio al servidor por parte del Cliente.  Esta interface conoce los servicios que presta el servidor y por ende están dispuestos para el Cliente |

Tabla 7: Interfaces en el diagrama de componentes T-Monopoly

# 3.2.1Subsistema Cliente

|  |  |
| --- | --- |
| Nombre Subsistema | * *Cliente* |
| Propósito del Subsistema | * Es la representación del jugador humano, encargado de solo interactuar con la interfaz gráfica , donde se le muestra el tablero y demás objetos importantes para llevar a cabo el desarrollo del juego * Es liviano ya que tiene tan solo los componentes de interfaz gráfica, es decir, sólo la parte visual del juego como tal y solicita los servicios al servidor por medio de la interface (ComunicacionRMI). |
| Composición del Subsistema | * Comunicación Cliente * Coordinador * GUI |

Tabla 8: Descripción Subsistema Cliente

### 3.2.1.1 COMPONENTE COMUNICACIÓN CLIENTE

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *Comunicación Cliente* |  |  |
| Propósito del  Componente | Encargado de solicitar los servicios ofrecidos por la interface en el subsistema del servidor (comunicación servidor) por medio de la comunicaciónRMI | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *N/A* |  |  |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *ComunicaciónRMI* | *T-Monopoly* | *Esta interfaz permite realizar la solicitud al servidor como tal, de forma que si esta comunicación no está bien hecha la información no llega a este subsistema y por ello ya existen fallas en la ejecución de este.* |

Tabla 9: Descripción Componente Comunicación Cliente

### 3.2.1.2 COMPONENTE COORDINADOR

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *Coordinador* |  |  |
| Propósito del  Componente | Pide ejecución del servicio ofrecido por el servidor, por medio de comunicación cliente y a partir de esta información actualiza la interfaz del usuario(GUI). | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *Solicita* | *Comunicación cliente*  *Coordinador* | *Permite al Coordinador solicitar determinados servicios al servidor* |
| *Actualiza* | *Coordinador*  *GUI* | *Permite que el controlador actualice la interfaz del jugador, acorde a la información que disponga en determinado momento de ejecución del juego* |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *ComunicaciónRMI* | *T-Monopoly* | *Esta interfaz permite realizar la solicitud al servidor como tal, de forma que si esta comunicación no está bien hecha la información no llega a este subsistema y por ello ya existen fallas en la ejecución de este.* |

Tabla 10: Descripción Componente Coordinador

### 3.2.1.3 COMPONENTE GUI

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *GUI* |  |  |
| Propósito del  Componente | Encargada de mostrar gráficamente el juego (tablero, fichas, propiedades….) al cliente ,muestra lo que está sucediendo en el servidor, a medida que se ejecuta o termina la ejecución de un turno | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *N/A* |  |  |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *Actualiza* | *Coordinador* | *Permite a la interfaz actualizarse acorde a la información que tiene el coordinador* |

Tabla 11: Descripción Componente GUI

# Subsistema Servidor

|  |  |
| --- | --- |
| Nombre Subsistema | * *Servidor* |
| Propósito del Subsistema | Representa la lógica del juego, el cual es el manejador del sistema, encargado de ejecutar cada una de las actividades necesarias para llevar a cabo el juego. Así mismo es el encargado de realizar la conexión a determinada partida creada y habilita la opción de almacenar la información del cliente (perfil) o de cada partida. |
| Composición del Subsistema | * Comunicación Servidor * Lógica * Persistencia |

Tabla 12: Descripción Subsistema Servidor

### 3.2.2.1 COMPONENTE COMUNICACIÓN SERVIDOR

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *Comunicación Servidor* |  |  |
| Propósito del  Componente | Encargado de mostrarle al cliente, por medio del RMI, los servicios que él puede brindarle al cliente | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *N/A* |  |  |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *ComunicaciónRMI* | *T-Monopoly* | *Esta interfaz permite realizar la solicitud al servidor como tal, de forma que si esta comunicación no está bien hecha la información no llega a este subsistema y por ello ya existen fallas en la ejecución de este.* |

Tabla 13: Descripción Componente Comunicación Servidor

### 3.2.2.2 COMPONENTE LÓGICA

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *Lógica* |  |  |
| Propósito del  Componente | Representa el juego, donde se llevan a cabo todas las acciones del juego y demás actividades correspondientes a este. | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *Ejecuta* | *Comunicación Servidor* | *Permite a la lógica iniciar operaciones acorde a la información que le envía por medio de esa interfaz* |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *N/A* |  |  |

Tabla 14: Descripción Componente Lógica

### 3.2.2.3 COMPONENTE PERSISTENCIA

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Componente | *Persistencia* |  |  |
| Propósito del  Componente | Encargado de almacenar en archivos planos la información correspondiente a:  Ranking  Perfiles  Historial del juego | | |
| Interfaces Disponibles | ***Nombre de la Interfaz*** | ***Componentes que la utilizan*** | ***Servicios prestados*** |
| *Guarda* | *Lógica* | *Permite almacenar la información de la partida y demás operaciones que requieran ser almacenadas por lo lógica* |
| Dependencias | ***Nombre de la Interfaz*** | ***Componente que la ofrece*** | ***Descripción de la dependencia*** |
| *Ejecuta* | *Lógica* | *El almacenamiento se realiza a partir de la ejecución de la lógica, por lo cual es importante que se ejecute, para realizar el almacenamiento* |

Tabla 15: Descripción Componente Comunicación Persistencia

### 3.3 Estrategias de Diseño

Con el fin de facilitar el desarrollo del a aplicación es importante definir e implementar determinadas estrategias de diseño, ya que al usarlas facilita la identificación del esquema general de las arquitectura a la vez que la forma en la que se va a implementar.

Para esto Alimnova® ha definido estrategias a partir de patrones de diseño que facilitarán la implementación de la arquitectura y el entendimiento de la misma, acorde a la arquitectura ya escogida (Cliente- Servidor) y teniendo en cuenta los requerimientos funcionales conforme a la prioridad asignada a cada uno de ellos.

|  |  |  |
| --- | --- | --- |
| **Estrategia de Diseño** | | **Descripción** |
| **Patrones de Diseño** | OBSERVER [11] | * Define una dependencia uno a muchos entre objetos de tal forma que cuando el estado de un objeto cambia, se les notifica el cambio a todos los que dependen de él y se actualizan de forma automática. * Será implementado en la parte del tablero, en donde este está pendiente cuando se presente un movimiento de una ficha para actualizar el tablero. * Es útil también cuando un cambio en un objeto exige cambios en otros, pero se desconoce a cuántos objetos afectan dichos cambios. En este caso, cuando es necesario informar un cambio de estado de una ficha dentro del tablero de juego, no se conoce con certeza la cantidad de jugadores o clientes que se encontrarán en ese momento usando la aplicación y quienes necesitan dicha información de estado. |
| PROXY [18] | * Para crear objetos que consumen muchos recursos, pero instanciarlos cuando el cliente lo solicite o se cumplan las condiciones para que esto suceda. * Este patrón ayuda a facilitar la comunicación entre el cliente y el servidor |
| FACHADA[19] | * Simplifica la interface entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace las veces de pantalla o fachada. * Este patrón es empleado en el subsistema cliente, con el fin de aislar cada una de las clases presentes en la Interfaz Gráfica de usuario (GUI), de la comunicación del cliente con el servidor, permitiendo así aislar este sistema complejo de las demás clases. |
| STATE[14] | * Cambia comportamiento dependiendo del estado del objeto * Usado cuando un objeto tiene diferentes estados y cambia su comportamiento para cada estado * Aplica este patrón si el comportamiento del un objeto depende de un estado y debe cambiar en tiempo de ejecución dependiendo del estado * Este patrón será muy útil para modificar el estado de cada propiedad acorde a si está comprada, hipotecada o libre, dependiendo de las transacciones que se lleven a cabo durante el juego |
| **Estilo arquitectónico** | CLIENTE – SERVIDOR | [**3.Apreciación Global**](#_3.Apreciación_Global) |
| **Priorización de Requerimientos** |  | Implementación de la aplicación de acuerdo a la priorización de los requerimientos funcionales. Es decir, primero se implementarán los requerimientos con mayor importancia y así hasta los de menos prioridad. |

Tabla 16: Estrategia de diseño

# 4. Diseño de Alto Nivel

### 4.1 Diagrama de despliegue

Dentro del diseño y el paradigma orientado objetos competen el diagrama de despliegue, hay dos tipos de diagramas que se desarrollan para tal objetivo: **[UML and the Unified Process, JIM ARLOW AND IL A NEUSTADT, Practical object-oriented analysis and design, Pearson Education, página 355-364**]

1. Descripción: Los nodos describen un tipo de hardware (como un PC), y los componentes representan un tipo de software.

2. Instanciación: Los nodos son instancias que describen un tipo de hardware, se relacionan con otras instancias de nodos y con instancias de componentes que describen un tipo de software.

Alimnova® Decidió trabajar con el diagrama de descripción, ya que para los propósitos de un diagrama de instanciación, se busca una aproximación hacía sistemas distribuidos, que no se contempla en nuestros objetivos, ni corresponde al alcance del proyecto T-Monopoly. Adicionalmente no se requiere como regla de de negocio [ver anexos documentos SPMP Hito 1].

Un diagrama de componentes debe proveer la información de localización física, donde se encuentran los diferentes paquetes y/o componentes del sistema. De esta manera identificando no sólo software sino hardware. [**Software Engineering for Game Developers, John P. Flynt with Omar Salem,Editorial Thomson, páginas 117-120** ]

Es importante recordar el tipo de arquitectura seleccionada por Alimnova® para el desarrollo de T-Monopoly [ver sección 3]. De la misma manera para poder comprender este diagrama, es necesario tener en cuenta el diagrama de componentes [ver sección 3.2].

Trabajando sobre terreno conocido, es importante describir los ítems u objetos que componen un diagrama de componentes:

Los diagramas de despliegue contemplan varios puntos de vista, buscan relacionar el hardware y software de diversas maneras. [**Large-Scale Software Architecture, Jeff Garland and Richard Anthony,A Practical Guide using UML, John Wiley & Sons, LTD, páginas 177-201**].

Para ello se trabajan 3 vistas:

Para lo que compete a nuestro proyecto resulta útil utilizar únicamente la vista de despliegue, ya que no contamos con modelos de instanciación, donde sea necesario localizar y estructurar varias máquinas. En referencia a la vista de procesos, no necesitamos hilos de ejecución debido a que su administración corre por cuenta del servicio RMI, y la ejecución de procesos es simple, se basa en una lógica para cada uno de los paquetes definidos (cliente, servidor).

Para la construcción de él diagrama de Despliegue de Descripción se definen dos tiempos: **[UML and the Unified Process, JIM ARLOW AND IL A NEUSTADT, Practical object-oriented analysis and design, Pearson Education, página 355-364**]

1. El flujo de los nodos y sus conexiones.
2. Asignación pertinente y coherente de los componentes a los nodos, que concuerdan con la arquitectura del sistema.

### 4.1.1 El flujo de los nodos y sus conexiones.

Ilustración xx: Descripción Conexiones

Para un mejor conocimiento de las restricciones del Sistema T-Monopoly [VER Anexo. DE [Documento\_SRS]]

### 4.1.2 Asignación pertinente y coherente de los componentes

Alimnova® escogió dos nodos que ejecutarán un paquete de funcionalidad, según el diagrama de componentes cliente y servidor [ver sección 3.2].

**Para el cliente:**

Ilustración xx: Descripción Máquina Cliente

**Para el servidor:**

Ilustración xx: Descripción Máquina Servidor

### 4.1.3 DIAGRAMA DESPLIEGUE

### 

### 4.1.4 documentación DIAGRAMA DESPLIEGUE

Para la especificación del diagrama de despliegue, el cual está basado en el de componentes se tendrá la siguiente tabla de especificación de las asociaciones entre componentes.

|  |  |
| --- | --- |
| id |  |
|  | |
| componente origen |  |
|  | |
| componente asociado |  |
|  | |
| descripción |  |

Tabla 11-Especificación diagrama despliegue

Cada uno de los campos se describe como:

* Id: identificador únicos de la asociación.
* Componente origen: De los dos componentes involucrados en una asociación, el componente que asocia, es decir de donde se origina la flecha de asociación en el diagrama.
* Componente destino: de los dos componentes involucrados en una asociación, el componente que es asociado, es decir de donde se apunta la flecha de asociación en el diagrama.

|  |  |
| --- | --- |
| id | SC1 |
|  | |
| componente origen | Server |
|  | |
| componente asociado | Comunication |
|  | |
| descripción | Define la interacción de cómo el servidor envía los datos en tiempo de ejecución, con el objetivo de comunicárselos al cliente, por medio del modulo de comunicaciones |

### 4.2 Diagrama de Comportamiento e Interacción

*Los diagramas de comportamiento de UML permiten tener un mejor acercamiento a lo que sucederá cuando la aplicación realice alguna funcionalidad específica, mientras que los diagramas de interacción añaden a esto el flujo de control y de datos [8].*

*Para estos diagramas lo ideal es escoger uno de los Casos de Uso principales del análisis realizado y hacer los diagramas en base a la elección. Una vez se ha seleccionado el caso de uso, también se hace la traza con los requerimientos asociados y se procede a la construcción de los diagramas.*

*Esta sección se divide en Diagramas de Actividad y de Secuencia*

### Diagrama de Actividad

*Una actividad es un paso en la ejecución de un programa, en la cual un trabajo se ejecuta. Esto puede incluir un cálculo, una consulta, manipulación de datos o simplemente verificación de información [6].*

*Los diagramas de actividad generalmente se utilizan para especificar [9]:*

* *Un método*
* *Un caso de uso*
* *Un proceso de negocio (Workflow)*

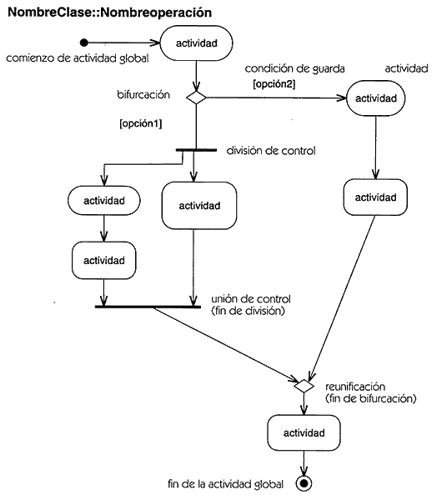


Ilustración 15: Diagrama de Actividad

### Diagrama de Secuencia

*Los diagramas de secuencia (al igual que los de colaboración) son usados para ilustrar la interacción entre objetos, por lo tanto estos diagramas modelan objetos y paso de mensajes entre objetos. Estos diagramas se hacen visualización basada en el tiempo [6].*

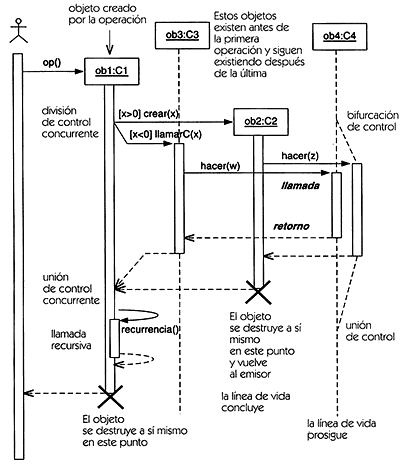


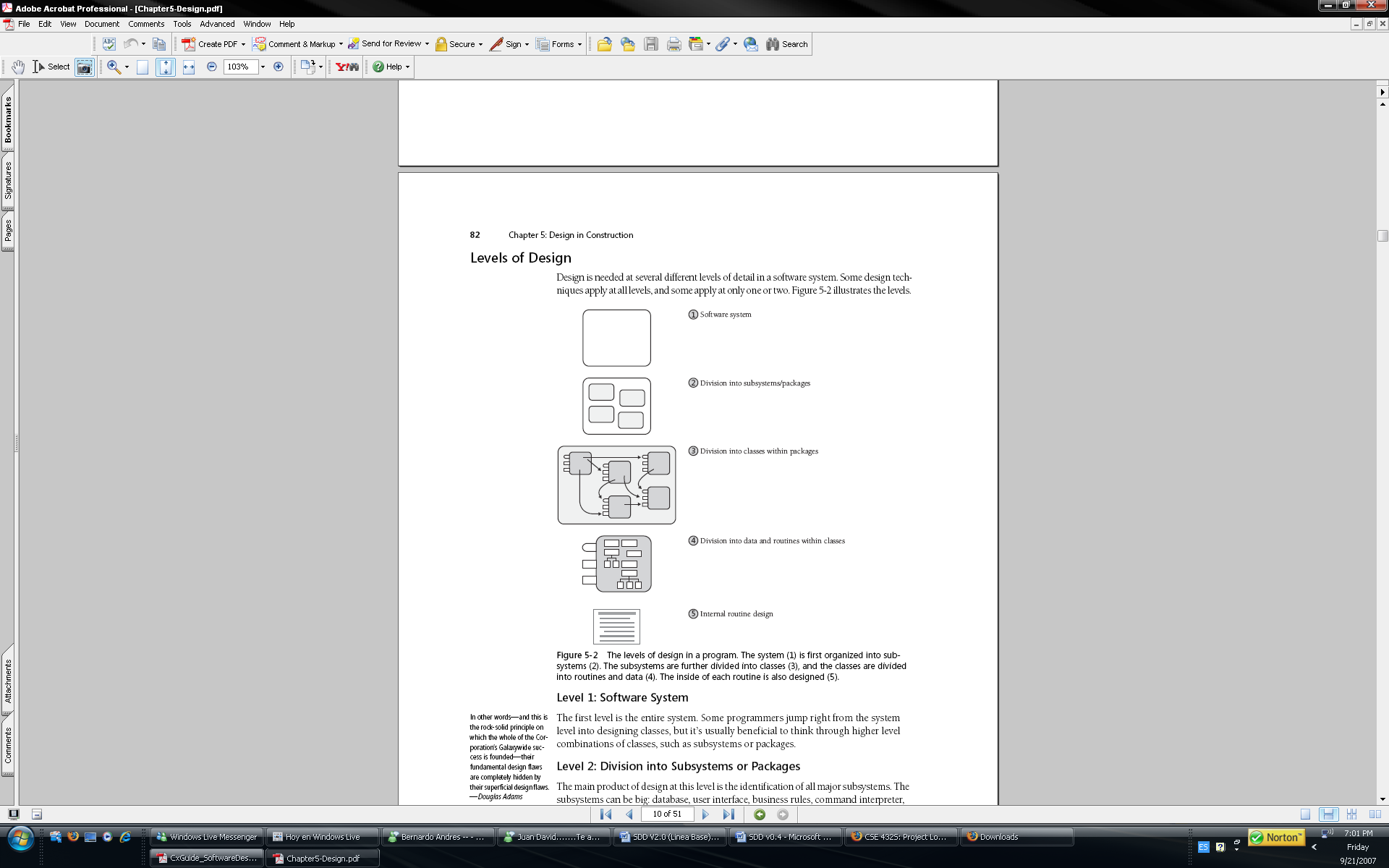
Ilustración 16: Diagrama de secuencia

# 5. Diseño de Bajo Nivel

*El diseño de bajo nivel describe de forma detallada cómo van a ser implementados los subsistemas, paquetes, módulos y/o componentes que se hayan definido en el diseño de alto nivel [10], para cada uno de estos se deben especificar las clases, interacciones y demás estructuras necesarias para su construcción [11] (ver ilustración 17).*

Ilustración 17: Diseño de bajo nivel

*Para realizar esta sección se deben llevar a cabo diferentes iteraciones, con las cuales se obtenga un mayor nivel de detalle hasta llegar a uno lo suficientemente claro para una fácil implementación desde el punto de vista del arquitecto. En la ilustración 18 se presentan los diferentes niveles de detalle del sistema de software, indicando cómo después del diseño de alto nivel se empieza con niveles más específicos como lo son la definición de clases con sus métodos y atributos. [12]*



Diseño de alto nivel

Diseño de bajo nivel

1

2

3

4

5

Sistema de Software

División en subsistemas/paquetes/ componentes

División en clases dentro de paquetes

División de datos y rutinas dentro de clases

Diseño interno de rutinas

Ilustración 18: Niveles de diseño

*Para la descripción de bajo nivel se sugiere manejar una estructura jerárquica de los niveles que se hayan definido, en la figura 19 se presenta un ejemplo en el cual se han definido inicialmente subsistemas y estos a su vez se organizan en componentes.*

Ilustración 19: Ejemplo diseño de bajo nivel

### 5.1 Subsistema 1

*En esta sección se deben listar todos los componentes, paquetes o módulos que contenga el subsistema 1.*

### Componente 1

*Se debe presentar el diagrama de clases del Componente 1 con su respectiva documentación, sin embargo, para comodidad se sugiere dejar en el presente documento la imagen del diagrama y realizar la documentación en el IDE seleccionado para la implementación con el fin de obtener la caracterización de las clases como un API, por ejemplo, utilizando JavaDoc.*

|  |  |
| --- | --- |
| Descripción de la clase | *Indica el objetivo que cumple la clase y las características propias de esta* |
| Descripción de atributos | *Por cada uno se debe especificar: el tipo de atributo, rango y una breve descripción.* |
| Descripción de métodos | *Por cada uno se debe incluir una breve descripción, la precondición para que el método pueda ser invocado, la poscondición indicando el resultado obtenido después de ser invocado, los parámetros que recibe (de estos debe estar especificado el rango y una breve descripción), el retorno indicando el tipo de dato que se retorna y el rango que posee; por último, se deben incluir las excepciones que va a manejar el método, describiendo en qué casos ocurriría.* |

Tabla 8: Documentación de clases

*En las ilustraciones 20, 21 y 22 se presenta un ejemplo de documentación de la clase “ConsultasDao” del proyecto 7 Texas Poker.*



Ilustración 20: Ejemplo Descripción de Clase



Ilustración 21: Ejemplo Resumen de Métodos



Ilustración 22: Ejemplo descripción de método

# 6. Diseño de Interfaces de Usuario

### 6.1 Diseño general de la aplicación

Se debe especificar cada una de las interfaces, indicando el diseño tanto de las entradas como de las salidas que generan información valiosa a los usuarios del sistema.

Para cada una de las interfaces de debe indicar:

Ilustración 23: Diseño general de la aplicación

Además se debe manejar el siguiente formato para cada una de las entradas y salidas de la interfaz

|  |  |
| --- | --- |
| Convención | *Identificador de la entrada/salida dentro de la interfaz* |
| Nombre | *Nombre de la entrada/salida dado en las documentaciones* |
| Alias (Interfaz) | *Nombre o Label con el que identifican la entrada/salida* |
| Alias (Código) | *Nombre a nivel del código fuente del programa de la entrada/salida* |
| Propósito | *Propósito de funcionalidad de la entrada/salida* |
| Tipo de Entrada | *Por ejemplo: Campo de Texto, Tabla, Botón, etc.* |
| Tipo de Dato | *Por ejemplo: Numérico, Alfanumérico o Alfabético* |
| Longitud mínima | *Longitud mínima asignada a la entrada/salida* |
| Longitud máxima | *Longitud máxima asignada a la entrada/salida* |
| Valor por defecto | *Valor por defecto que se encuentra en la entrada/salida* |
| Rango | *Rango de los valores que puede tener la entrada/salida* |
| Validación | *Como se verifica que la entrada/salida es valida* |
| Secuencia de Entrada | *Orden en la secuencia de entradas/salidas de la interfaz* |
| Comentarios | *Comentarios como por ejemplo: obligatoriedad, medio por el cual entra o sale* |

Tabla 9: Descripción de entradas y salidas

### 6.2 Árbol de navegabilidad

En el árbol de navegabilidad de las interfaces gráficas, cada nodo representa el título de dicha interfaz y las flechas los posibles caminos





Ilustración : Ejemplo árbol de navegabilidad

# Anexos

**REFERENCIAS**

[1] Garlan D. et al, An Introduction to Software Architecture, Enero 1994, Carnegie Mellon University

[2] Avila J, Estilos Arquitectónicos, Disponible en http://sophia.javeriana.edu.co/~javila/arquitectura/estilosArquitectonicos.pdf

[3] IronWorks, Descripción Del Diseño De Software 7 Texas Poker, Primer Semestre 2007, Pontificia Universidad Javeriana

[4] Bruegge B, Dutoit AH. Ingeniería de Software orientada a objetos. 1st ed. Trujano G. México: Pearson Educación; 2002.

[5] IEEE (Institute of Electrical and Electronics Engineers), IEEE Recommended Practice for Software Design Descriptions, IEEE-SA Standards Board, Septiembre 1998.

[6] Pender T, UML Bible, Wiley Publishing Inc. Publicado en 2003

[7] Construx Software, DESIGN - CXOne Standard, Construx Software Builder, Inc, Noviembre 2002.

[8] Larman C. UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al pro-ceso unificado. 2nd ed. Aragon DF. Madrid: Pearson Educación. S.A.; 2003.

[9] Booch G. et al, El lenguaje Unificado de Modelado, Manual de Referencia. Rational Software Corporation. Publicado en 2000

[10] Construx Software, DESIGN - CXOne Guide, Construx Software Builder, Inc, Noviembre 2002.

[11] Gill C. CSE 432S: Project Low Level Design. Washington University, Departamento de Ciencia y Computación, Febrero 2006.

[12] Mcconnell S. Code Complete, capítulo 5 Desing in Construction. Segunda edición. Disponible en: ww.cc2e.com/File.ashx?cid=336