Universidad de las Ciencias Informáticas

Facultad 10

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

“Arquitectura del sistema de clonación y distribución de imágenes de sistemas operativos”

Autor: Irina González Oduardo

Tutor: Ing. Sergio García de la Puente

Ciudad de la Habana 29 de mayo del 2009

“Año del 50 Aniversario del Triunfo de la Revolución”

Pensamiento

"Programar sin una arquitectura en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas"

Danny Thorpe

Declaración de autoría

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas con los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____.

_____	_____
Irina González Oduardo	Ing. Sergio García de la Puente

Dedicatoria

A mi mamá Gisela, mi hermanito Luis, mi hermana Indira...

los adoro y los adoraré por siempre.

Agradecimientos

*A mi madre **Gisela Oduardo Morales**, por todo su amor, por el apoyo, el ejemplo y por saberme guiar siempre por el camino correcto.*

*A mi hermanito **Luisito** y a mi hermana **Indira** por ser los mayores tesoros que tengo en la vida y por ser mi fuerza en los momentos difíciles.*

*A **Chichi** por haberme apoyado siempre que lo necesite y por su ayuda para poder terminar mi carrera.*

*A mis abuelos **Juan** y **Candelaria**, a mis tías **Gina y Gloria,** a mis primas **Lenicel y Yaisel**, a mis primos por sus consejos, por apoyarme y alentarme siempre para continuar.*

*A **Amaury Viera Hernández** un amigo y hermano más al cual le debo muchas de las cosas que he aprendido en mi carrera y en la vida, a él le estoy muy agradecida por su paciencia y amor conmigo, a él por ser una persona especial en mi vida.*

*A mi tutor **Sergio García de la Puente**, por ayudarme siempre que lo necesité.*

*A mis amigas y hermanas **Lisandra y Mairelis**, que le debo mucho en mi carrera y en la vida.*

*A mis amigas **Kiro, Ailema** porque se que siempre podre contar con ellas, las quiero.*

*A mis amigos "**Los Feos"** que a pesar de los sustos que me han ocasionado los quiero mucho y siempre van a ser mis amigos.*

*A mi amigo **Jeicer Rodríguez** por ser una personas a la que quiero mucho y porque sé que siempre voy a poder contar con el.*

*A mis **compañeros de estudio** por toda la ayuda que me han prestado.*

A todo el que de una forma u otra ha hecho posible la realización de este sueño.

Datos de contacto

Nombre del tutor: Sergio J. García de la Puente
Correo electrónico: sgarcia@uci.cu.

Graduado en el curso 2007-2008 de Ingeniero en Ciencias de la Informática, en la Universidad de las Ciencias Informáticas. Se ha desempeñado como profesor de pregrado en la asignatura de Ingeniería de Software con dos años de experiencia laboral. Ha cursado cursos de superación posgraduada en temáticas afines a su especialidad, así como en la contribución de su formación como docente. Está vinculado al grupo de Migración y Soporte al Software Libre, adscrito al polo de Software Libre de la Facultad 10 de la Universidad de las Ciencias Informáticas. Se desempeña en los roles de Analista de sistema y como asesor en temas de Arquitectura de Software. Está vinculado a un grupo de desarrollo de software. Pertenece al grupo de Migración y se desempeña como especialista de Migración Parcial. Ha presentado temas de investigación en los eventos UCIENCIA 2009 e Informática 2009. Participó en el Comité Organizador del IV Taller Internacional de Software Libre y estándares abiertos de Software, celebrado en el marco de la XIII Convención y Feria Internacional Informática 2009, celebrada en La Habana, Cuba, durante los días del 9 al 13 de febrero de 2009.

Resumen

En el trabajo de diploma se define la arquitectura del Sistema de Clonación y Distribución de imágenes de Sistemas Operativos “SISTCLON”, para lo cual se tienen en cuenta diferentes aspectos:

- Se realiza un estudio del estado del arte de los estilos arquitectónicos más utilizados hoy en día y de los principales modelos y tendencias de la arquitectura. Se definen las herramientas y la plataforma de desarrollo así como la metodología que guiará el proceso de desarrollo del sistema SISTCLON.
- Se realiza la descripción de la arquitectura, se definen los patrones de diseño utilizados y se fundamenta la utilización de los patrones arquitectónicos cliente/servidor, modelo-vista-controlador y en capas.
- Se realiza la evaluación y análisis de la propuesta arquitectónica realizada mediante el empleo de las métricas de diseño de alto nivel que se concentran en las características de la arquitectura del programa, con énfasis en la estructura arquitectónica y en la eficiencia de los módulos.

Hasta la actualidad en Cuba no se ha desarrollado ningún software que realice la clonación, distribución de imágenes de sistemas operativos, administración de las computadoras clientes, entre otras funciones, por lo que este sistema tiene un impacto significativo.

Índice de contenido

Introducción

En los últimos años, las nuevas Tecnologías de la Información y las Comunicaciones (TIC) han provocado un fuerte movimiento en las comunidades científicas de todo el mundo. Cuba no está exenta a este auge surgido en la nueva “era de la información”, por lo que ha percibido la necesidad de informatizar la sociedad cubana. Con el objetivo de alcanzar un mayor desarrollo, se ha introducido aun más en la esfera de la producción de software, con la creación de la más joven de las universidades cubanas, la Universidad de las Ciencias Informáticas (UCI).

Esta universidad está estructurada en diferentes facultades, donde cada facultad se orienta a un perfil determinado, pero que tienen como principal objetivo la formación de especialistas en informática, teniendo como principio del proceso docente educativo: "La formación desde la producción". De acuerdo con este principio, la formación docente está vinculada a la producción, donde el estudiante adquiere las habilidades prácticas para el desarrollo y producción de software, el cual se desarrolla en los laboratorios docentes.

Estos laboratorios docentes se encuentran equipados con diferentes tipos de computadoras, que son atendidas por el grupo de técnicos de la universidad, quienes les dan el mantenimiento e instalan el software necesario de acuerdo con los requerimientos del proceso docente educativo. Este proceso es muy complejo y engorroso teniendo en cuenta que para cada facultad las necesidades en cuanto a software que utilizan son diferentes y que las imágenes a instalar en las computadoras tienen que ser personalizadas para cada tipo de computadora de acuerdo al perfil de cada facultad.

A raíz de este problema se desarrolló un sistema de clonación y distribución de imágenes de sistemas operativos que también realizas otras funciones como es la administración de las computadoras clientes con el objetivo de automatizar el proceso. Dicho sistema en la actualidad no tiene bien definida una arquitectura de software que brinde una flexibilidad, reusabilidad y mantenimiento al mismo. Para darle solución a esta problemática se plantea el siguiente **problema científico** ¿Qué arquitectura de software definir que brinde una flexibilidad, reusabilidad y mantenimiento al sistema de clonación y distribución de imágenes de

sistemas operativos en la UCI?

El **objeto de estudio** de esta investigación está centrado en el proceso de desarrollo de software del sistema de clonación y distribución de imágenes de sistemas operativos (SISTCLON).

El **campo de acción** está enmarcado la arquitectura de software de SISTCLON empleando metodología ágil.

El **objetivo general** de este trabajo es diseñar la arquitectura por la que se regirá el desarrollo, soporte y mantenimiento de SISTCLON.

Para cumplir con el objetivo propuesto se han definido los siguientes **objetivos específicos:**

- Realizar un estudio del arte referente a los diferentes tipos de diseños arquitectónicos usando metodologías ágiles.
- Realizar la descripción de la arquitectura de SISTCLON.
- Definir el patrón arquitectónico de SISTCLON.
- Evaluar el diseño arquitectónico de SISTCLON.

Las principales **tareas de la investigación** que se proponen para concretar los objetivos específicos son las siguientes:

- Investigar los diferentes diseños arquitectónicos existentes, sus características, ventajas y desventajas usando metodologías ágiles.
- Desarrollar la descripción de la arquitectura de SISTCLON.
- Realizar un estudio de los diferentes patrones arquitectónicos.
- Determinar el o los patrones que se vinculan a la propuesta.
- Realizar la evaluación del diseño arquitectónico de SISTCLON mediante el empleo de métricas para la validación de la solución propuesta.

Para dar cumplimiento a estas tareas se plantea el seguimiento y estudio de dos tipos de métodos, los teóricos y los empíricos, los cuales aparecen a continuación.

Métodos teóricos

- **Análisis y Síntesis:** Permite procesar información, proponer la arquitectura teniendo en cuenta tendencias y modelos de arquitectura, arribar a las conclusiones de la investigación y precisar las características del los modelos a utilizar en el desarrollo del sistema.
- **Inducción y deducción:** Proponer estilos de arquitectura a partir del estudio de los modelos utilizados para el desarrollo arquitectónico de software.
- **Histórico Lógico:** Permite determinar las tendencias actuales tanto de enfoques como de modelos de arquitecturas a utilizar.

Métodos empíricos

- **Análisis de documento:** para la fundamentación teórica del problema.

El contenido de este documento está estructurado en tres capítulos:
El **Capítulo 1.** Fundamentación teórica: donde se realiza un estudio detallado de las tendencias, tecnologías y metodologías actuales con el fin de proponer las más adecuadas para la solución del problema. Además se da a conocer el estado del arte referente a los diseños arquitectónicos usando metodologías ágiles.

El **Capítulo 2.** Descripción y análisis de la solución propuesta: en este capítulo se realiza la propuesta de la arquitectura para el Sistema de clonación y distribución de imágenes de Sistemas Operativos, teniendo en cuenta plataforma, lenguaje de programación, patrones de arquitectura, requisitos del sistema, etc.

El **Capítulo 3.** Análisis de resultados y validación de la propuesta: en este capítulo se realiza la evaluación el diseño arquitectónico de SISTCLON para garantizar la fiabilidad y calidad de dicha propuesta de solución

mediante el empleo de métricas de diseño de alto nivel que se concentran en la estructura arquitectónica y en la eficiencia de los módulos.

Finalmente se exponen las conclusiones del trabajo y se hace referencia de la bibliografía utilizada en la investigación.

1. Capítulo 1. Fundamentación Teórica

1.1. Introducción

En este capítulo se analizan los aspectos fundamentales a tener en cuenta para el desarrollo de la propuesta de arquitectura de software para SISTCLON en la UCI, basados en el objeto de estudio y el campo de acción. Además de realizar un análisis de algunas definiciones de arquitectura de software, estilos arquitectónicos y las arquitecturas más comunes para seleccionar la más adecuada. Conjuntamente se hace un análisis de algunos de los protocolos de comunicación y tecnologías a usar.

1.2. Conceptos generales

1.2.1. Sistema de clonación y distribución de imágenes de sistemas operativos

El término clonación en el vocablo informático viene asociado exactamente a replicar una información que se encuentra en una zona de memoria a otra zona de memoria, a esto también se le llama “copia exacta”. En la clonación de sistemas operativos existe una variante y es la copia exacta o clonación de una zona de memoria de una computadora a otra zona de memoria en el mismo disco duro o en otro disco duro remoto ya sea por medio de una salva de el sistema operativo en algún dispositivo de almacenamiento o la manera más usada actualmente, vía red usando el Pre-Boot Execution Enviornment (PXE) de la tarjeta de red y a estos software se les denomina sistemas de instalación remota.

Los sistemas de instalación remota se definen como los software que permiten que un usuario instale y configure nuevos equipos clientes de forma remota, sin necesidad de trabajar directamente en cada equipo cliente. Estos sistemas en la actualidad se han desarrollado en dos ramas fundamentales: los sistemas de instalación remota básica, y los sistemas de clonación y distribución de software.

Los sistemas de instalación remota básica se limitan a la instalación de un sistema operativo base en los equipos clientes, haciendo uso de la técnica de los llamados archivos de respuestas. Esta técnica consiste

en crear un archivo donde se especifican las respuestas a las preguntas que hace el instalador por defecto de los sistemas operativos, estos archivos son creados por el administrador del sistema y una vez concluida su configuración son distribuidos en los diferentes equipos clientes con el objetivo de automatizar el proceso de instalación de los diferentes equipos.

Los sistemas de clonación y distribución de software se basan en un clásico modelo cliente/servidor, donde los clientes son los equipos que se van a instalar, y el servidor es la computadora que le proveerá a los clientes las imágenes que se les instalará. Por lo tanto enestos sistemas se tienen dos tipos de aplicaciones diferentes: una para el cliente y otra para el servidor. La aplicación servidor utiliza diferentes servicios DHCP, TFTP, NFS para poder comunicarse con los diferentes clientes y realizar el proceso de clonación y distribución. **[16]**

1.2.2. Arquitectura de Software

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, probadores, programadores, y el resto de los integrantes del equipo o grupo de desarrollo trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Una arquitectura de software se selecciona y diseña con base en objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad que es la propiedad de un sistema que representa la cantidad de esfuerzo requerida para conservar su funcionamiento normal o para restituirlo una vez se ha presentado un evento de falla, la flexibilidad y escalabilidad, que es la capacidad de un sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes. **[1]**

La arquitectura de software dirige el desarrollo de los sistemas de software y contribuye a que estos se lleven a cabo en los límites establecidos de costos y tiempo, y fundamentalmente debe garantizar que se cumpla con los requisitos funcionales y no funcionales de los usuarios. La arquitectura de software es el resultado del trabajo durante todo el ciclo de vida del proyecto, y principalmente en las primeras iteraciones de un grupo de trabajo encabezado por el arquitecto (o grupo de arquitectura, en dependencia de las dimensiones del proyecto). **[1]**

Definiciones de Arquitectura de Software

La Arquitectura de Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que expresa que: “La Arquitectura de Software es la organización fundamental de un sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, además de los principios que orientan su diseño y evolución” **[5].**

La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad **[3].**

1.2.3. Estilos

Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en re-utilizarlos en situaciones semejantes que se presenten en el futuro. Igual que los patrones de arquitectura y diseño, todos los estilos tienen un nombre: estilos de centrados en los datos, estilos de llamada y retorno, tubería-filtros. **[7]**

1.2.4. Patrones

Un patrón es un modelo que podemos seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Los patrones capturan la experiencia existente y probada para promover buenas prácticas.

Los patrones expresan esquemas de organización estructural fundamentales para los sistemas de software, estos se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento, el código. Los patrones arquitectónicos, se han materializado con referencia a lenguajes y paradigmas también específicos de desarrollo. **[24]**

1.3. Tendencias, tecnologías y metodologías de desarrollo actuales

1.3.1. Metodologías de desarrollo de software

El desarrollo de software no es una tarea fácil. En un proyecto de desarrollo de software la metodología es la que define “quién” debe hacer “qué”, “cuándo” y “cómo” debe hacerlo. No existe una metodología de software universal.

Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos.

Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el

proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto de software.

Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

Cada equipo de desarrollo escoge la metodología según las características de su proyecto, por lo que es importante determinar el alcance del proyecto antes de escoger la metodología que se va a emplear en el desarrollo del mismo. A continuación se mencionan algunas de las metodologías de desarrollo de software que más se utilizan en la actualidad.

1.3.1.1. Proceso Unificado de Rational (RUP)

No es objetivo de esta investigación tener en cuenta los métodos pesados o tradicionales, se hace una excepción con el Proceso Unificado de Rational ya que es uno de los procesos más generales de los existentes actualmente, unifica los mejores elementos de las metodologías anteriores.

RUP está pensado para adaptarse a cualquier proyecto. Un proyecto realizado siguiendo RUP se divide en cuatro fases:

- Inicio
- Elaboración
- Construcción
- Transición

En cada fase se ejecutarán una o varias iteraciones (de tamaño variable según el proyecto), y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo que requieren las nuevas actividades anteriormente citadas.

RUP define nueve flujos de trabajo a realizar durante el ciclo de desarrollo del proyecto, los seis primeros flujos son de ingeniería, y los tres últimos de son de apoyo.

- Modelamiento del negocio
- Requisitos
- Análisis y diseño
- Implementación
- Prueba
- Instalación

El proceso define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y el resultado (artefactos en el vocabulario de RUP) que se espera de ellos.

RUP presenta tres características fundamentales, es dirigido por casos de uso para describir lo que se espera del software, es centrado en la arquitectura, documentándose lo mejor posible, basándose en UML como lenguaje principal para el modelado, además de iterativo e incremental.

RUP es un proceso muy general y muy grande, por lo que antes de usarlo hay que adaptarlo a las características del sistema. En la actualidad ya existen muchos procesos descritos que son versiones reducidas del RUP.

1.3.1.2. Extreme Programming (XP)

XP **[15]** es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua

entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en "*Una explicación de la programación extrema. Aceptar el cambio*" **[15]** sin cubrir los detalles técnicos y de implantación de las prácticas.

XP se basa en 4 valores fundamentales **[8]**:

- Comunicación
- Sencillez
- Retroalimentación
- Valentía

El objetivo principal que persigue XP es la satisfacción del cliente. Esta metodología fue diseñada para proporcionar el software que el cliente necesita cuando lo necesite. Debe responder de forma rápida a los cambios en las necesidades del cliente, incluso cuando estos cambios se produzcan al final del ciclo de vida de dicho software (simplicidad y realimentación).

El segundo objetivo es potenciar al máximo el trabajo en equipo. Tanto los jefes de proyecto, como los clientes y desarrolladores, son parte del equipo encargado de la implementación de software de calidad (comunicación). Esto implicará que los diseños deberán ser claros y sencillos. Y los clientes deberán disponer de versiones operativas cuanto antes para poder participar en el proceso creativo mediante sus sugerencias y aportaciones.

Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, de manera que nos podamos beneficiar de la retroalimentación tan a menudo como sea posible.

«Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.» Kent Beck.

1.3.1.3. Scrum

Scrum es un proceso de desarrollo de software iterativo e incremental utilizado comúnmente en entornos basado en la metodología ágil de desarrollo de software. El desarrollo se realiza en forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre 2 y 4 semanas. **[13]**

Scrum se utiliza como marco para otras prácticas de ingeniería de software como RUP o Extreme Programming. Se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión.

Está diseñado especialmente para adaptarse a los cambios en los requerimientos, por ejemplo en un mercado de alta competitividad. Los requerimientos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. De esta forma se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente. Se busca entregar software que realmente resuelva las necesidades, aumentando la satisfacción del cliente.

En Scrum, el equipo se focaliza en una única cosa: construir software de calidad. Por el otro lado, la gestión de un proyecto Scrum se focaliza en definir cuales son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo. Se busca que los equipos sean lo más efectivos y productivos que sea posible.

Scrum tiene un conjunto de reglas muy pequeño y muy simple y está basado en los principios de inspección continua, adaptación, auto-gestión e innovación. El cliente se entusiasma y se compromete con el proyecto dado que ve crecer el producto iteración a iteración y encuentra las herramientas para alinear el desarrollo con los objetivos de negocio de su empresa. Por otro lado, los desarrolladores encuentran un ámbito propicio para desarrollar sus capacidades profesionales y esto resulta en un incremento en la motivación de los integrantes del equipo. **[14]**

Aunque Scrum estaba enfocado a la gestión de procesos de desarrollo de software, puede ser utilizado en equipos de mantenimiento de software, o en una aproximación de gestión de programas.

1.3.1.4. SXP

SXP es una metodología compuesta por la metodología XP y SCRUM que es un método adaptativo de gestión de proyectos que se basa en los principios ágiles. SXP ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, ayudando al líder del proyecto a tener un mejor control del mismo.

SCRUM es una forma de gestionar un equipo de manera que trabaje de forma eficiente y de tener siempre medidos los progresos, de forma que sepamos por dónde andamos. XP más bien es una metodología encaminada para el desarrollo; consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto.

Consta de 4 fases principales:

- **Planificación-Definición:** donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto;
- **Desarrollo:** es donde se realiza la implementación del sistema hasta que este listo para ser

entregado;

- **Entrega:** puesta en marcha;
- **Mantenimiento:** donde se realiza el soporte para el cliente.

De cada una de estas fases se realizan numerosas actividades tales como el levantamiento de requisitos, la priorización de la lista de reserva del producto (LRP), definición de las historias de usuario (HU), diseño, implementación, pruebas, entre otras; de donde se generan artefactos para documentar todo el proceso. Las entregas son frecuentes, y existe una refactorización continua, lo que nos permite mejorar el diseño cada vez que se le añada una nueva funcionalidad.

SXP esta especialmente indicada para proyectos de pequeños equipos de trabajo, rápido cambio de requisitos o requisitos imprecisos, muy cambiantes, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro, permitiendo además seguir de forma clara el avance de las tareas a realizar, de forma que los jefes pueden ver día a día cómo progresa el trabajo.**[4]**

1.3.2. Protocolos de comunicación

Un protocolo de red es un conjunto de estándares que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red.

Los protocolos son reglas de comunicación que permiten el flujo de información entre computadoras distintas que manejan lenguajes distintos, por tal sentido, el protocolo TCP/IP fue creado para las comunicaciones en Internet, para que cualquier computador se conecte a Internet, es necesario que tenga instalado este protocolo de comunicación.

Los protocolos pueden estar implementados en hardware (tarjetas de red), software (drivers), o una combinación de ambos.

1.3.2.1. Unicast

Unicast se basa en un proceso de envío de una información en una o más unidades de datos (datagramas IP) desde una máquina origen a una única máquina destinataria o receptor final. Por tanto, es una transmisión punto a punto con cada destinatario. Si se desea enviar la misma información y hay "n" destinatarios, habrá "n" comunicaciones punto a punto independiente o "n" copias de la misma información enviadas desde la máquina origen **[51]**.

1.3.2.2. Multicast

Se basa en un único proceso de envío, independientemente del número de máquinas receptoras, de una misma información en una o más unidades de datos (datagramas IP) desde una misma información en una o mas unidades de datos desde una máquina origen a todas las máquinas destinatarias que posean al menos un miembro de un determinado grupo de multidifusión y que, además, compartan una misma dirección de multidifusión; y, posiblemente, dispersas geográficamente en múltiples redes por Internet.

Y por tanto, sin necesidad de transmitir desde el origen una copia de la misma información, por separado, a cada una de dichas máquinas. Se resalta el hecho de que desde la máquina origen sólo se envía una vez la pertinente información y no se transmiten "n" copias de la misma aunque haya "n" destinatarios. En este escenario, los routers intermedios de multidifusión por Internet tienen que poseer previamente la capacidad5 necesaria para hacer las copias necesarias, de la información transmitida, desde el origen a las correspondientes máquinas destinatarias **[50]**.

1.3.2.3. Peer to Peer (P2P)

Una red informática P2P se refiere a una red que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red, todos los nodos se comportan igual y pueden realizar el mismo tipo de operaciones; pudiendo no obstante diferir en configuración local, velocidad de proceso, ancho de banda y capacidad de almacenamiento **[49]**.

1.3.2.4. BitTorrent

BitTorrent esta basado en los principios del protocolo P2P, es un protocolo diseñado para el intercambio de archivos entre iguales, funciona bajo el principio del intercambio permanente. Varios descargan el mismo archivo de un servidor. Pera para hacerlo más rápido, todos se intercambian fragmentos de archivos que uno posee pero no el otro. Por consiguiente, uno se convierte también en servidor. De ahí que BitTorrent necesita que se conozca exactamente la dirección IP de los que descargan **[47]**.

Una gran diferencia con el P2P clásico, es que en BitTorrent no hay necesidad de estar en lista de espera como con el P2P. En P2P, cuanto más se da más se recibe. En BitTorrent, es de inmediato: No se necesita que ya hayas dado para recibir. En la medida en la que los que dan estén disponibles **[48]**.

Otra consecuencia, para hacer más rápida la velocidad de descarga, se debe permanecer conectado en BitTorrent incluso después del final de la descarga.

1.3.3. Estilos arquitectónicos

Un estilo arquitectónico expresa la organización estructural de un sistema de software. Provee un conjunto predefinido de subsistemas, sus responsabilidades, y guías para organizar las relaciones entre los mismos.

En la ingeniería de software al igual que en la construcción; los estilos arquitectónicos describen categorías, conjunto de componentes, cooperación entre ellos, conectores y modelos. Podemos definir los estilos arquitectónicos como: conceptos descriptivos que definen una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales.

Los estilos arquitectónicos son la clave en la reducción del costo durante el desarrollo del software. Reutilizar soluciones efectivas es una de las prácticas fundamentales en el éxito del proceso de ingeniería, y

lo mismo pasa en el contexto de la arquitectura de software. La reutilización a nivel de arquitectura se consigue con la adopción de estilos arquitectónicos.

Los estilos arquitectónicos fueron identificados por Mary Shaw y Paul Clements como un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Así los estilos más universales se encuentran encapsulados en las clases de estilos **[6]**.

A continuación se mencionan algunos de los estilos arquitectónicos más utilizados en la actualidad.

1.3.3.1. Estilos de flujo de datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote. **[7]**

- **Tuberías y filtros.**
- **Secuencial por lotes**

1.3.3.2. Estilo centrado en datos

El estilo centrado en datos resulta apropiado para sistemas que se centran en el acceso y actualización de datos en estructuras de almacenamiento que son compartidos por un número indefinido de componentes consumidores. Una de las propiedades más destacable de este estilo arquitectónico es la necesidad de crear persistencia de los datos almacenados. Existen dos tipos de componentes, un componerte que realiza el almacenaje y persistencia de los datos y otros componentes que interactúan con dichos datos.

En estos sistemas hay que tener encuentra que se pueden producir problemas en la interacción de los componentes que manejan los datos con el componente que almacena los datos (creación, lectura y actualización simultáneas a un mismo dato) y la coherencia (hay que evitar que un componente pueda tener

un dato con un valor desactualizado). Pueden existir módulos de almacenamiento activos o pasivos según quien se encargue de la notificación de la actualización de los datos (si el componente encargado del almacenamiento es el notificador, entonces este es un modelo activo). Patrones característicos de este estilo son la pizarra y el repositorio. **[7]**

- **Arquitectura de pizarra**
- **Arquitectura de repositorio**

1.3.3.3. Estilos de llamada y retorno

Este estilo es interesante en los sistemas que se centran en la interacción de un usuario con el propio sistema. Podemos dividir la arquitectura en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica de negocio que se realiza tras la correspondiente llamada del usuario.

Esta familia de estilos enfatiza la modificabilidad, la escalabilidad, la reusabilidad y la usabilidad del sistema. Al diseñar una arquitectura de este tipo es importante saber que datos son los que servirán para interactuar con el usuario y cuales servirán solo como lógica de aplicación. En esta familia de arquitecturas se destacan los patrones modelo-vista-controlador y arquitectura en tres capas. **[7]**

- **Model-View-Controller (MVC)**
- **Arquitecturas en Capas**
- **Arquitecturas Orientadas a Objetos**
- **Arquitecturas Basadas en Componentes**

1.3.3.4. Estilo basado en eventos

Al igual que la familia de arquitecturas de llamada y retorno, las arquitecturas basadas en evento centran su funcionamiento en la interacción tras la llegada de un evento. La diferencia radica en que en el estilo basado

en eventos la interacción se realizan entre dos componentes contenidos en el sistema, mientras que la interacción de las arquitecturas llamada-retorno se producen entre un usuario, no incluido en el sistema, y un componente del sistema.

Una característica que tiene que cumplir los elementos incluidos en la arquitectura es que deben ser independientes entre sí, para poder lograr la interacción deseada entre los mismos. Estas comunicaciones son dos a dos, por lo que una de las preocupaciones que hay que tener en cuenta a la hora de diseñar una arquitectura de este tipo es lograr la independencia en la interacción entre dos componentes con otra interacción.

Dentro de este estilo los patrones se diferencian según sea el tipo de comunicación de la interacción, que puede ser síncrona o asíncrona y por paso de mensajes o llamada directa. Entre los patrones de este estilo se destacan el publicador/subscriptor. **[7]**

- **Publicador/subscriptor**

1.3.3.5. Estilo basado en código móvil

Este estilo se caracteriza por su enorme portabilidad. Las arquitecturas que siguen este estilo tienen una parte del sistema que pertenece al propio entorno nativo de la máquina que lo incluye, la otra parte no. Para realizar una comunicación entre ambas partes se necesita de un intérprete que permita la traducción.

Este intérprete es un conector con datos que contiene las instrucciones en el lenguaje no nativo de la máquina y una información de estado de esta parte no nativa. Por tanto, la principal preocupación que hay que tener en cuenta a la hora de diseñar una arquitectura perteneciente a esta familia es como llevar a cabo esta traducción y realizar la integración de la parte del sistema no incluida en la máquina física. El principal ejemplo de esta familia de arquitecturas es la arquitectura de máquinas virtuales. **[7]**

- **Arquitectura de máquinas virtuales**

1.3.3.6. Estilo Peer-to-Peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast. Miembros de la familia son los estilos basados en eventos, en mensajes, en servicios y en recursos. **[7]**

- **Arquitecturas Basadas en Eventos.**
- **Arquitecturas Orientadas a Servicios (SOA).**
- **Arquitecturas Basadas en Recursos**

1.4. Patrones de arquitectura

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. Este esquema se especifica describiendo las componentes, con sus responsabilidades, relaciones, y las formas en que colaboran.

1.4.1. Patrón Modelo Vista Controlador

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Su principal finalidad es mejorar la reusabilidad y que las modificaciones en las vistas impacten en menor medida en la lógica de negocio o de datos.

El **Modelo** es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

- Definir las reglas de negocio (la funcionalidad del sistema).
- Llevar un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo.

El **Controlador** es responsable de:

- Recibir los eventos de entrada.
- Contiene reglas de gestión de eventos, estas acciones pueden suponer peticiones al modelo o a las vistas.

Las **Vistas** son responsables de:

- Recibir datos del modelo y lo muestra al usuario.
- Tienen un registro de su controlador asociado.
- Pueden dar el servicio de "Actualización", para que sea invocado por el controlador o por el modelo cuando es un modelo activo.

Es un patrón que convierte una aplicación en un paquete mantenible, modular y de desarrollo rápido. La modularidad y el diseño independiente permiten a los desarrolladores y diseñadores hacer cambios en alguna parte de la aplicación sin afectar a los demás.

1.4.2. Patrón Capas

La funcionalidad principal de este patrón es crear una modularidad del sistema asignando a cada capa funcionalidades específicas y bien definidas.

- **La capa de la Presentación**

Esta capa reúne todos los aspectos del software que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Estos típicamente incluyen el manejo y aspecto de las ventanas, el formato de

los reportes, menús, gráficos y elementos multimedia en general.

➢**La capa del Dominio de la Aplicación**

Esta capa reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

➢**La capa del Acceso a Datos**

Esta capa reúne todos los aspectos del software que tienen que ver con el manejo de los datos persistentes, por lo que también se le denomina la capa de las Bases de Datos.

Existen especificaciones de este patrón donde se manipula como será la comunicación entre cada una de las capas definidas.

1.5. Posibles arquitecturas a utilizar

Generalmente, no es necesario innovar una nueva arquitectura de software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así, las arquitecturas más utilizadas son:

- **Cliente-Servidor.**
- **Arquitectura en capas.**
- **Orientada a Servicios.**

1.5.1. Arquitectura Cliente-Servidor

Una arquitectura es un conjunto de reglas, definiciones, términos y modelos que se emplean para producir un producto. La arquitectura Cliente/Servidor agrupa conjuntos de elementos que efectúan procesos distribuidos y computo cooperativo.

La tecnología Cliente/Servidor es el procesamiento cooperativo de la información por medio de un conjunto de procesadores, en el cual múltiples clientes, distribuidos geográficamente, solicitan requerimientos a uno o más servidores centrales.

Desde el punto de vista funcional, se puede definir la tecnología Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información de forma transparente aún en entornos multiplataforma. Se trata pues, de la arquitectura más extendida en la realización de Sistemas Distribuidos.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

Un sistema Cliente/Servidor es un sistema de información distribuido basado en las siguientes características:

- **Servicio:** unidad básica de diseño. El servidor los proporciona y el cliente los utiliza.
- **Recursos compartidos:** Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.

- **Protocolos asimétricos:** Los clientes inician "conversaciones". Los servidores esperan su establecimiento pasivamente.
- **Transparencia de localización física de los servidores y clientes:** El cliente no tiene por qué saber dónde se encuentra situado el recurso que desea utilizar.
- **Independencia de la plataforma HW y SW que se emplee.**
- **Sistemas débilmente acoplados**. Interacción basada en envío de mensajes.
- **Encapsulamiento de servicios**. Los detalles de la implementación de un servicio son transparentes al cliente.
- **Escalabilidad horizontal** (añadir clientes) **y vertical** (ampliar potencia de los servidores). Se puede aumentar la capacidad de clientes y servidores por separado.
- **Integridad:** Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.
- **Centralización del control:** Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.

En el modelo usual Cliente/Servidor, un servidor, se activa y espera las solicitudes de los clientes. Habitualmente, programas cliente múltiples comparten los servicios de un programa servidor común. Tanto los programas cliente como los servidores son con frecuencia parte de un programa o aplicación mayores.

El esquema de funcionamiento de un Sistema Cliente/Servidor sería:

1. El cliente solicita una información al servidor.
2. El servidor recibe la petición del cliente.
3. El servidor procesa dicha solicitud.
4. El servidor envía el resultado obtenido al cliente.
5. El cliente recibe el resultado y lo procesa.

Componentes de la arquitectura Cliente/Servidor

El modelo Cliente/Servidor es un modelo basado en la idea del servicio, en el que el cliente es un proceso

consumidor de servicios y el servidor es un proceso proveedor de servicios. Además esta relación está establecida en función del intercambio de mensajes que es el único elemento de acoplamiento entre ambos.

De estas líneas se deducen los tres elementos fundamentales sobre los cuales se desarrollan e implantan los sistemas Cliente/Servidor: el proceso cliente que es quien inicia el diálogo, el proceso servidor que pasivamente espera a que lleguen peticiones de servicio y el middleware que corresponde a la interfaz que provee la conectividad entre el cliente y el servidor para poder intercambiar mensajes.

Para entender en forma más ordenada y clara losconceptos y elementos involucrados en esta tecnología se puede aplicar una descomposición o arquitectura de niveles. Esta descomposición principalmente consiste en separar los elementos estructurales de esta tecnología en función de aspectos más funcionales de la misma:

- **Nivel de Presentación:** Agrupa a todos los elementos asociados al componente Cliente.
- **Nivel de Aplicación:** Agrupa a todos los elementos asociados al componente Servidor.
- **Nivel de comunicación:** Agrupa a todos los elementos que hacen posible la comunicación entre los componentes Cliente y servidor.
- **Nivel de base de datos:** Agrupa a todas las actividades asociadas al acceso de los datos.

Este modelo de descomposición en niveles, como se verá más adelante, permite introducir más claramente la discusión del desarrollo de aplicaciones en arquitecturas de hardware y software en planos.

Elementos principales

Cliente

Un cliente es todo proceso que reclama servicios de otro, el cliente es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor. Se lo conoce con el término front-end. Este normalmente maneja todas las funciones relacionadas con la manipulación y despliegue de datos, por lo que están desarrollados sobre plataformas que permiten construir interfaces gráficas de usuario (GUI), además de

acceder a los servicios distribuidos en cualquier parte de la red.

Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

La funcionalidad del proceso cliente marca la operativa de la aplicación. De este modo el cliente se puede clasificar en:

- **Cliente basado en aplicación de usuario**. Si los datos son de baja interacción y están fuertemente relacionados con la actividad de los usuarios de esos clientes.
- **Cliente basado en lógica de negocio**. Toma datos suministrados por el usuario y/o la base de datos y efectúa los cálculos necesarios según los requerimientos del usuario.

Servidor

Un servidor es todo proceso que proporciona un servicio a otros. Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se lo conoce con el término back-end. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos. Las principales funciones que lleva a cabo el proceso servidor se enumeran a continuación:

1. Aceptar los requerimientos de bases de datos que hacen los clientes.
2. Procesar requerimientos de bases de datos.
3. Formatear datos para trasmitirlos a los clientes.
4. Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

Puede darse el caso que un servidor actúe a su vez como cliente de otro servidor. Existen numerosos tipos de servidores, cada uno de los cuales da lugar a un tipo de arquitectura Cliente/Servidor diferente. Desde el punto de vista de la arquitectura cliente/servidor y del procesamiento cooperativo un servidor es un servicio software que atiende las peticiones de procesos software clientes.

Middleware

El middleware es un módulo intermedio que actúa como conductor entre sistemas permitiendo a cualquier usuario de sistemas de información comunicarse con varias fuentes de información que se encuentran conectadas por una red. En el caso que nos concierne, es el intermediario entre el cliente y el servidor y se ejecuta en ambas partes.

La utilización del middleware permite desarrollar aplicaciones en arquitectura Cliente/Servidor independizando los servidores y clientes, facilitando la interrelación entre ellos y evitando dependencias de tecnologías propietarias.

El middleware se estructura en tres niveles:

1. Protocolo de transporte.
2. Network Operating System (NOS).
3. Protocolo específico del servicio.

Las principales características de un middleware son:

1. Simplifica el proceso de desarrollo de aplicaciones al independizar los entornos propietarios.
2. Permite la interconectividad de los Sistemas de Información del Organismo.
3. Proporciona mayor control del negocio al poder contar con información procedente de distintas plataformas sobre el mismo soporte.
4. Facilita el desarrollo de sistemas complejos con diferentes tecnologías y arquitecturas.
5. Dentro de los inconvenientes más importantes destacan la mayor carga de máquina necesaria para que puedan funcionar.

Tipos de arquitectura Cliente/Servidor

Uno de los aspectos claves para entender la tecnología Cliente/Servidor, y por tanto contar con la capacidad de proponer y llevar a cabo soluciones de este tipo, es llegar a conocer la arquitectura de este modelo y los conceptos o ideas asociados al mismo. Más allá de entender los componentes cliente/middleware/servidor, es preciso analizar ciertas relaciones entre éstos, que pueden definir el tipo de solución que se ajusta de mejor forma a las estadísticas y restricciones acerca de los eventos y requerimientos de información que se obtuvieron en la etapa de análisis de un determinado proyecto.

De hecho el analista deberá conocer estos eventos o restricciones del proyecto para que a partir de ahí, se puedan hacer las consideraciones y estimaciones de la futura configuración, teniendo en cuenta aspectos como por ejemplo, la oportunidad de la información, tiempo de respuesta, tamaños de registros, tamaño de bases de datos, estimaciones del tráfico de red, distribución geográfica tanto de los procesos como de los datos, etc.

Modelos Cliente/Servidor

Una de las clasificaciones mejor conocidas de las arquitecturas Cliente/Servidor se basa en la idea de planos, la cual es una variación sobre la división o clasificación por tamaño de componentes. Esto se debe a que se trata de definir el modo en que las prestaciones funcionales de la aplicación serán asignadas, y en qué proporción, tanto al cliente como al servidor. Dichas prestaciones se deben agrupar entre los tres componentes clásicos para Cliente/Servidor: interfaz de usuario, lógica de negocios y los datos compartidos, cada uno de los cuales corresponde a un plano. Ni que decir tiene que la mala elección de uno u otro modelo puede llegar a tener consecuencias fatales.

A nivel de software

Este enfoque o clasificación es el más generalizado y el que más se ajusta a los enfoques modernos, dado que se fundamenta en los componentes lógicos de la estructura Cliente/Servidor y en la madurez y popularidad de la computación distribuida. Por ejemplo, esto permite hablar de servidores de aplicación

distribuidos a lo largo de una red, y no tiene mucho sentido identificar a un equipo de hardware como servidor, si no más bien entenderlo como una plataforma física sobre la cual pueden operar uno o más servidores de aplicaciones.

Modelo Cliente/Servidor 2 capas

Las aplicaciones cliente-servidor clásicas o de 2 capas como su nombre lo indica agrupan la lógica de presentación (interfaz) y la lógica de aplicación en la máquina cliente y acceden a fuentes de datos compartidos a través de una conexión de red que se encuentran en el servidor de datos. Estas aplicaciones de 2 capas trabajan bien en aplicaciones a escala de departamentos con un modesto número de usuarios, una base de datos sencilla y una red segura y rápida.

La ventaja que presenta este tipo de aplicaciones es que los datos están centralizados. Esta centralización beneficia a la empresa pues es más fácil compartir los datos, se simplifica la generación de reportes y se proporciona consistencia en el acceso a los datos.

Modelo Cliente/Servidor 3 capas

Esta estructura se caracteriza por elaborar la aplicación en base a dos capas principales de software, más la capa correspondiente al servidor de base de datos. Al igual que en la arquitectura dos capas, y según las decisiones de diseño que se tomen, se puede balancear la carga de trabajo entre el proceso cliente y el nuevo proceso correspondiente al servidor de aplicación.

En este esquema el cliente envía mensajes directamente al servidor de aplicación el cual debe administrar y responder todas las solicitudes. Es el servidor, dependiendo del tipo de solicitud, quien accede y se conecta con la base de datos.

Ventajas:

- Reduce el tráfico de información en la red por lo que mejora el rendimiento de los sistemas (especialmente respecto a la estructura en dos planos).

- Brinda una mayor flexibilidad de desarrollo y de elección de plataformas sobre la cual montar las aplicaciones. Provee escalabilidad horizontal y vertical.
- Se mantiene la independencia entre el código de la aplicación (reglas y conocimiento del negocio) y los datos, mejorando la portabilidad de las aplicaciones.
- Los lenguajes sobre los cuales se desarrollan las aplicaciones son estándares lo que hace más exportables las aplicaciones entre plataformas.
- Dado que mejora el rendimiento al optimizar el flujo de información entre componentes, permite construir sistemas críticos de alta fiabilidad.
- El mismo hecho de localizar las reglas del negocio en su propio ambiente, en vez de distribuirlos en la capa de interfaz de usuario, permite reducir el impacto de hacer mantenimiento, cambios urgentes de última hora o mejoras al sistema.
- Disminuye el número de usuarios (licencias) conectados a la base de datos.

Inconvenientes:

- Dependiendo de la elección de los lenguajes de desarrollo, puede presentar mayor complejidad en comparación con Cliente/Servidor dos planos.
- Existen pocos proveedores de herramientas integradas de desarrollo con relación al modelo Cliente/Servidor dos planos, y normalmente son de alto costo.

Las distintas arquitecturas cliente/servidor presentan variaciones acerca de cómo son distribuidas las diferentes funciones de las aplicaciones de sistemas entre el cliente y el servidor, sobre la base de los conceptos de los tres componentes generales de cualquier sistema de información:

- **La lógica de acceso a datos**. Funciones que gestionan todas las interacciones entre el SW y los almacenes de datos (archivos, bases de datos, etc.) incluyendo recuperación/consulta, actualización, seguridad y control de concurrencia.
- **La lógica de presentación**. Funciones que gestionan la interfaz entre los usuarios del sistema y el SW, incluyendo la visualización e impresión de formas y reportes, y la posibilidad de validar entradas

del sistema.

- **La lógica de negocio o lógica de la aplicación**. Funciones que transforman entradas en salidas, incluyendo desde simples sumas hasta complejos modelos matemáticos, financieros, científicos, de ingeniería, etc.

1.5.2. Arquitectura en Capas

La arquitectura en capa define el patrón en capas como una organización jerárquica. Lo que posibilita un diseño basado en niveles de abstracción creciente, posibilitando a los implementadores, particionar un problema en una secuencia de pasos incrementales. Este estilo de desarrollo en varios niveles, facilita que en caso que ocurra algún cambio, sólo se tendrían que realizar las correcciones necesarias en el nivel requerido sin tener que revisar código de otros niveles.

La programación por capas es un estilo de programación en la que el objetivo primordial es dividir, fraccionar y llegar a separar la Presentación (donde muestras y obtienes datos), de la Lógica de Negocio (donde realizas operaciones) y con todo esto se obtendría independencia, por lo que si hay cambios de arquitectura se puede acceder a los datos o cambiar el negocio o modificar la presentación y solo sería en esa fracción de la capa.

En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

La arquitectura en capas es algo referente a la arquitectura de software, pero realmente si una aplicación no tiene arquitectura de capas adecuada para el tipo de necesidad se podría decir que no cumple con la programación orientada a objetos. Una arquitectura como ésta puede tener tantas capas como se necesite, dependiendo siempre de la complejidad de la aplicación.

1- Capa de presentación:

En esta capa se diseña todo lo que constituye la interfaz gráfica y la interacción del usuario con el software.

Se comunica únicamente con la capa de negocio. Representa el conjunto de componentes que genera la información que se representará en la interfaz de usuario del cliente. La norma es que la mayor parte de las interacciones con el usuario se realicen en las JSP debido a su flexibilidad, ya que integran de forma natural etiquetas XHTML, HTML, XML y JSF.

La capa de presentación se implementa mediante tecnologías JSP, Servlets, con las que se construyen y conectan las interfaces gráficas con la capa de lógica del negocio. La función de la capa de presentación es la de proveer una interfaces para realizar la transferencia de datos. Es la encargada de interactuar con el usuario, obtener, validar y enviar los datos al servidor.

Los componentes de la interfaz de usuario deben mostrar datos al usuario, obtener y validar los datos procedentes del mismo e interpretar las acciones de éste que indican que desea realizar una operación con los datos. Asimismo, la interfaz debe filtrar las acciones disponibles con el fin de permitir al usuario realizar sólo aquellas operaciones que le sean necesarias en un momento determinado. Los componentes de interfaz de usuario:

- No inicializan, participan ni votan en transacciones.
- Presentan una referencia al componente de proceso de usuario actual si necesitan mostrar sus datos o actuar en su estado.
- Pueden encapsular tanto la funcionalidad de visualización como un controlador.

Las interfaces web deben ser consistentes con la información que se requiere, no se deben utilizar más campos de los necesarios, así como la información requerida tiene que ser especificada de manera clara y concisa, no debe haber más que lo necesario en cada formulario. El objetivo principal de este componente es facilitar al usuario la interacción con la misma.

Dentro de la parte técnica, la capa de presentación contiene los objetos encargados de comunicar al usuario con el sistema mediante el intercambio de información, capturando y desplegando los datos necesarios para realizar alguna tarea.

2- Capa de lógica de negocio:

En esta capa residen los objetos de negocio que interactúan con los objetos de dominio. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse para un correcto funcionamiento lógico de la aplicación.

Los objetos de negocio, separan los datos y la lógica de negocio haciendo uso del modelo de objetos definido, dichos objetos de negocio son los encargados de procesar la información proveniente tanto de la Presentación, como de la Capa de Acceso a Datos.

Los objetos de negocio son los objetos del modelo que implementan la lógica de negocio. Sus funciones fundamentales son:

- Realizar la validación de los datos introducidos por el usuario.
- Ejecutar la petición realizada por el usuario. Para ello podrá valerse de transacciones. contra Sistemas Host, consultas a bases de datos, consultas a proveedores de contenidos, etc.
- Generar los objetos que utilizarán las Vistas para mostrar los resultados obtenidos.

Su función es garantizar que toda la lógica de negocio esté bien localizada y no mezclada con objetos de otras capas. Esto conlleva grandes ventajas, pues define los objetos de negocio, y crea las interfaces de servicio con sus implementaciones, las cuales hacen uso de los objetos de dominio.

Además de encontrarse toda la lógica de la aplicación y el modelo de objetos encargados de la manipulación de los datos existentes, así como el procesamiento de la información ingresada o solicitada por el usuario en la capa de presentación.

En la misma se reciben solicitudes de la capa de presentación, la cual procesa y obtiene los resultados invocando a la capa de Acceso a Datos. La comunicación con otros sistemas que actúan en conjunto, se hace mediante esta capa, es en ella están expuestas el conjunto de funcionalidades o métodos a exportar.

La responsabilidad de esta capa es implementar lógica de negocio que será consumida por la interfaz del sistema y por otros sistemas a través de los servicios compartidos. Recibe los datos que ingresó el usuario del sistema mediante la capa de presentación, luego los procesa y crea objetos según lo que se necesite hacer con los datos.

Al encapsular los datos, la aplicación asegura mantener la consistencia de los mismos, así como obtener información precisa de las bases de datos e ingresar en las mismas solamente la información necesaria, a través de la capa de acceso a datos, asegurando así no tener datos duplicados ni en las bases de datos, ni en los reportes solicitados por el usuario.

Existen tres tipos de componentes de negocio fundamentales:
- **Lógica de Negocio:** Implementan la funcionalidad de negocio del sistema.
- **Entidades de negocio:** Representan las entidades del sistema.
- **Workflow:** Implementan procesos de negocio en los cuales participan entidades y lógica de negocio.

3- Capa de Acceso a Datos:
La capa de Acceso a Datos es el puente entre la capa de Lógica de Negocio y el Sistema de Base de Datos. Encapsula la lógica de acceso a datos. Aquí se encuentran componentes que hacen transparente el acceso a la base de datos. Este es el lugar idóneo para implementar los objetos de acceso a datos, permitiendo ingresar, obtener, actualizar y eliminar información del Sistema de Bases de Datos.

Los Objetos de Acceso a Datos (DAOs) encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos. Las implementaciones de los DAOs estarán disponibles para los objetos de negocio.

En esta capa se utiliza el framework Hibernate pues se está trabajando con programación orientada a objetos y bases de datos relacionales, observando que estos son dos paradigmas diferentes.

El modelo relacional trata con relaciones, tuplas y conjuntos y es muy matemático por naturaleza. Sin embargo, el paradigma orientado a objetos trata con objetos, sus atributos y relaciones entre objetos. Cuando se quiere que los objetos sean persistentes utilizando para ello una base de datos relacional, se muestra una desavenencia entre estos dos paradigmas, la también llamada diferencia objeto-relacional (object - relational gap). Un Mapeador Objeto- Relacional (ORM por sus siglas en inglés) ayudará a evitar esta diferencia.

La misma es la responsable de unir todos los índices de funciones y estructuras que devuelven los servicios y combinarlos en una macro estructura. Su función es mantener sincronizada en todo momento la información existente en el sistema web y la que se encuentra en la base de datos. También permite el intercambio de información con clientes y proveedores.

Esta capa de acceso a datos se relaciona directamente con el patrón DataMapper pues permite manejar toda la carga y almacenamiento entre el modelo de dominio y la base de datos, logrando que ambos varíen independientemente. DataMapper permite disponer de dos objetos. El primero representa la lógica de negocio. El segundo el acceso a datos. Se trata de realizar una correlación entre objetos de lógica de negocio y tablas de la base de datos.

La aplicación utiliza una arquitectura multicapa. Para una arquitectura colocada, las capas de presentación, de lógica de negocio y de acceso a datos están físicamente localizadas aislando las responsabilidades de cada capa. La arquitectura colocada hace que la aplicación sea más simple y escalable. **[12]**

Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Además si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores, los cuales recibirán las peticiones del ordenador en que resida la capa de negocio**. [9]**

Si por el contrario fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de acceso a datos.

1.6. Enfoque de la arquitectura de software en las metodologías ágiles

Las metodologías ágiles hacen menos énfasis en la arquitectura del software, la arquitectura se va definiendo y mejorando a lo largo del proyecto. Se parte con una arquitectura muy simple orientada a los requisitos implementados en la iteración. Se hace evolucionar la arquitectura, cambiando la arquitectura sobre un sistema ya implantado. La arquitectura nunca es más compleja de lo estrictamente necesario. Si un cambio no se adapta a la arquitectura, la arquitectura se cambia.

En SXP no se enfatiza la definición temprana de una arquitectura de software estable para el sistema. Dicha arquitectura se asume de forma evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo.

Una metáfora es una historia compartida que describe cómo debería funcionar el sistema. Martin Fowler en "*Is Design Dead"* **[29]** explica que la práctica de la metáfora consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Este conjunto de nombres ayuda al diseño y a los métodos del sistema **[4]** pues brinda un contexto al equipo para entender los elementos básicos y sus relaciones y le proporciona integridad conceptual.

En SXP la descripción de la arquitectura se concibe mediante la utilización del documento de la arquitectura, donde se recopila toda la información referente a la utilización de los diferentes patrones tanto de diseño como los patrones arquitectónicos empleados, herramientas, lenguajes de programación, IDE, framework, y además se define la estrategia de integración entre los diferentes componentes del sistema (Ver **Anexo #**

4).

1.7. Conclusiones

En este capítulo se han analizado algunos aspectos teóricos que serán de gran ayuda para el desarrollo de los próximos capítulos, ya que es de suma importancia tenerlo en cuenta para proponer una solución. Se profundizó en el conocimiento de algunos conceptos necesarios para la compresión de este trabajo. Además se realizó un análisis completo de las tecnologías que serán utilizadas a lo largo del desarrollo del sistema propuesto.

2. Capítulo 2. Descripción y análisis de la solución propuesta

2.1. Introducción

El desarrollo de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema.

La arquitectura de software constituye un conjunto de decisiones significativas acerca de la organización de un sistema, incluye la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía esta organización. La arquitectura del software no sólo se interesa por la estructura y el comportamiento, sino también por las restricciones y compromisos de uso, funcionalidad, funcionamiento, flexibilidad al cambio, reutilización, comprensión, economía y tecnología, así como por aspectos estéticos.

En este capítulo se presentan algunos conceptos relacionados con la arquitectura del software; se define la propuesta de arquitectura para SISTCLON y se presenta la descripción de la arquitectura mediante la utilización de la metodología ágil SXP, se justifica la utilización de los diferentes patrones tanto de diseño como los patrones arquitectónicos empleados y se define la estrategia de integración entre los diferentes componentes del sistema.

2.2. Descripción de la arquitectura del sistema SISTCLON

Según el documento de la arquitectura concebido en la metodología SXP, la solución arquitectónica propuesta para el sistema SISTCLON se define como sigue a continuación.

2.2.1. Lista de reserva del producto de SISTCLON

La lista de reserva del producto es una lista priorizada que define el trabajo que se va a realizar en el proyecto. Cuando un proyecto comienza es muy difícil tener claro todos los requerimientos sobre el producto. Sin embargo, suelen surgir los más importantes que casi siempre son más que suficientes para una Iteración. Aunque puede crecer y modificarse a medida que se obtiene más conocimiento acerca del producto y del cliente. Con la restricción de que solo puede cambiarse entre las iteraciones.

Siempre el objetivo es asegurar que el producto definido al terminar la lista es el más correcto, útil y competitivo posible y para esto la lista debe acompañar los cambios en el entorno y el producto. Considerar que los elementos que tienen más o menos influencian en el éxito del proyecto en un momento dado; por lo que los elementos con mayor prioridad son los que se realizan primero **[4]**.

Tabla 1. Lista de Reserva del Producto de SISTCLON.

Prioridad	Item	Descripción	Estimación	Estimado por
Muy alta				

1	Obtener direcciones IP de los clientes conectados.	0.5	ANA
2	Organizar listado de IP de acuerdo a su hardware.	1	ANA
3	Mostrar listado de IP de los clientes conectados.	0.5	ANA
4	Crear Particiones.	2	ANA
5	Definir sistema de ficheros de la partición creada.	1	ANA
6	Mostrar espacio disponible para particionar.	1.5	PRO
7	Definir tamaño de la partición a crear.	1	ANA
8	Definir tipo de partición a crear (Extendida, lógica, primaria).	1	ANA
9	Eliminar partición.	2	PRO
10	Mostrar estado del disco duro durante el proceso de particionado.	5	PRO
11	Obtener direcciones IP de los clientes conectados(RESC-Client).	3	PRO
12	Mostrar direcciones IP de los clientes conectados(RESC-Client).	1	PRO
13	Enviar comando a cliente(s).	2	PRO
14	Enviar script a cliente(s).	2	PRO
15	Buscar imagen a instalar.	0.5	PRO
16	Escoger la partición donde se instalara la imagen.	1	ANA
17	Enviar imagen a cliente(s).	5	PRO

Alta				
	1	Mostrar información del disco duro de un cliente.	8	ANA
	2	Mostrar información de las particiones del disco duro de un cliente.	2	ANA
	3	Mostrar información de la motherboard de un cliente.	1	PRO
	4	Mostrar la información de la interface de red de un cliente.	1	PRO
	5	Mostrar información del display de un cliente.	1	PRO
	6	Mostrar información de la multimedia de un cliente.	1	PRO
	7	Soporte para Windows.	8	PRO
	8	Configurar los archivos principales del sistema operativo después de instalado. (solo para Linux).	2	PRO
Media				
	1	Permitir realizar una imagen de una partición específica de un cliente de forma remota.	3	PRO
	2	Gestionar tareas pendientes.(RESC)	2	PRO
	3	Soporte para las P5LD2-VM	2	PRO
Baja				
	1	Retroalimentación con el servidor. (RESC)	2	PRO
	2	Listado de aplicaciones instaladas en el sistema operativo.	4	PRO
Requerimientos no funcionales				
	1	Diseñar interfaz gráfica para seleccionar la imagen a clonar en la pc cliente.	2	DIS

	2	Diseñar interfaz gráfica para realizar imagen de un cliente.	3	DIS
	3	El sistema permitirá su extensibilidad, permitiendo agregar nuevas funcionalidades en un futuro.	2	PRO
	4	El sistema debe ser bien documentado de forma tal que en caso de mantenimiento el tiempo que se requiera sea el mínimo.	1	ANA
	5	El sistema deberá funcionar sobre plataforma GNU/Linux.	2	PRO

2.2.2. Plan de releases

El plan de releases no es más que una plantilla donde se recogen las iteraciones a realizar con sus características, además del orden de las historias de usuario con su planificación estimada para ser implementadas. En todas las iteraciones del ciclo de desarrollo de SISTCLON tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración **[4]**.

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software, lo que equivaldría a los casos de uso en el proceso unificado. Las mismas son escritas por los clientes como las tareas que el sistema debe hacer y su construcción depende principalmente de la habilidad que tenga el cliente para definirlas. Son utilizadas como el único documento de requisitos que se genera en XP. Son escritas en lenguaje natural, sin un formato predeterminado, no excediendo su tamaño de unas pocas líneas de texto (*ver Anexos*) **[11]**.

Las historias de usuario tienen el mismo propósito que los casos de uso pero existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario.

Conocida la lista de reserva del producto se pueden definir las historias de usuarios arquitectónicamente significativos para el desarrollo de la nueva versión de SISTCLON, a continuación se muestra el plan de releases donde aparece el orden en que van a implementar las historias de usuarios.

Tabla 2. Plan de releases de SISTCLON.

Release	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	Se obtiene la versión 0.1 de SISTCLON con la terminación de las HU descritas.	3, 1, 2, 5, 4, 6	8
2	Se obtiene la versión 0.2 de SISTCLON con la terminación de las HU descritas	8, 9, 12, 13, 14, 7, 11, 10,15,16	8
3	Se obtiene la versión 0.3 de SISTCLON con la terminación de las HU descritas	17,18,19,20,21,22,23,24,25, 26,27,28,29	7
4	Se obtiene la versión 0.4 de SISTCLON con la terminación de las HU descritas	30,31,32,33,34	10
5	Se obtiene la versión 0.5 de SISTCLON con la terminación de las HU descritas		8
6	Se obtiene la versión 0.6 de		8

	SISTCLON con la terminación de las HU descritas		

2.2.3. Herramientas asociadas al desarrollo del sistema SISTCLON

En el desarrollo de todo sistema informático es de vital importancia la selección de las herramientas, lenguajes y tecnologías a utilizar, paso que garantizará, de realizarse correctamente, un óptimo desempeño del sistema. Para el desarrollo de SISTCLON la selección se realizó teniendo en cuenta la infraestructura tecnológica de la UCI y valorando que el sistema a desarrollar, estaría orientado a funcionar sobre el sistema operativo GNU/Linux.

Para el desarrollo del sistema de clonación y distribución de software se utilizaron numerosas herramientas, lenguajes y tecnologías **[16]**; a raíz de nuevos análisis, se decide agregar nuevas herramientas y en algunos casos muy particulares sustituir una tecnología por una mejor. En esta sección se trata humildemente de justificar la selección de las herramientas, lenguajes y tecnologías a utilizar para la implementación de este sistema. Para esto ello se realizó un estudio de las posibles herramientas a utilizar en su construcción. Teniéndose en cuenta las tendencias actuales y las novedades de cada una de ellas. Las herramientas utilizadas son:

2.2.3.1.Entorno de desarrollo integrado (IDE)

Un entorno de desarrollo integrado o, en inglés, Integrated Development Environment ('IDE'), es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma

interactiva, sin necesidad de trabajo orientado a archivos de texto. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación.

Los programadores recurren a un Entorno de Desarrollo Integrado con el fin de facilitar su tarea de aprendizaje y desarrollo. La elección de un IDE no es fácil, solo es necesario que posea las características obvias y generales que debe ofrecer como son editor sin corrección sintáctica y coloreo, herramientas gráficas, soporte integrado para la compilación, ejecución de programas, y opciones de debugging.

2.2.3.1.1. Code::blocks

Code::blocks es un entorno de desarrollo integrado para el desarrollo de programas en lenguaje C++. Está basado en la plataforma de interfaces gráficas WxWidgets, lo que le permite correr libremente en diversos sistemas operativos, y es de licencia GPL.

Code::Blocks es un IDE construido como un núcleo altamente expansible mediante plugins. Actualmente la mayor parte de la funcionalidad viene provista por los plugins incluidos por defecto. No es un IDE autonómo que acepta plugins, sino que es un núcleo abstracto donde los plugins se convierten en una parte vital del sistema. Esto lo convierte en una plataforma muy dinámica y potente, no solo por la facilidad con que nueva funcionalidad puede ser incluida, sino por la capacidad de poder ser usada para construir otras herramientas de desarrollo tan solo añadiendo plugins.**[39]**

2.2.3.1.2.Anjuta

Anjuta es un entorno de desarrollo integrado (IDE) desarrollado por Naba Kumbar en 1999. Fue desarrollado para programar en C y C++ en sistemas GNU/Linux aunque actualmente admite lenguajes como C#, Java, Perl y Python. Su principal objetivo es trabajar con GTK y en el escritorio GNOME, además ofrece un gran número de características avanzadas de programación. Anjuta es un software libre, liberado bajo la licencia GPL. Incluye un administrador de proyectos, asistentes, plantillas, depurador interactivo y un poderoso editor que verifica y resalta la sintaxis escrita. **[28]**

2.2.3.2. Framework

Un framework no es mas que un conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software. Un framework ofrece una guía arquitectónica partiendo el diseño en clases abstractas y definiendo sus responsabilidades y sus colaboraciones. Un desarrollador personaliza el framework para una aplicación particular mediante herencia y composición de instancias de las clases del framework.

Es una infraestructura de software que crea un entorno común para integrar aplicaciones e información compartida dentro de un dominio dado. Es una aplicación semicompleta que contiene componentes estáticos y dinámicos que pueden ser personalizados para obtener aplicaciones de usuario específicas.

Los frameworks son la piedra angular de la moderna ingeniería del software y están ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente (source code).

Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. La capacidad de reutilización del código y del diseño de frameworks permite una productividad mayor y un tiempo breve en el desarrollo de aplicaciones.

2.2.3.2.1. CodeIgniter

CodeIgniter es un framework para el desarrollo de aplicaciones web usando PHP. Este permite el desarrollo de proyectos mucho más rápidos que si se escribiera código desde cero. Provee una rica colección de librerías para las tareas necesarias más comunes. CodeIgniter permite concentrase en el desarrollo del proyecto en cuestión, minimizando la cantidad de código necesaria para realizar las tareas.

CodeIgniter usa el patrón de diseño arquitectónico Modelo-Vista-Controlador como paradigma de arquitectura de desarrollo, la cual separa en 3 capas distintas: la representación de datos, el interfaz de usuario y el controlador de eventos respectivamente. **[42]**

2.2.3.2.2. WxWidgets

WxWidgets es un framework especializado en el desarrollo de aplicaciones multiplataforma en lenguaje C++ aunque también existen bindings para Python y Perl, soporta Windows, Linux, Mac OS X , Unix y sus variantes, Solaris, Plataformas Embedded (inicios de investigación), también en plataformas móviles como Microsoft Pocket PC, y Palm OS; se distribuye bajo licencia Open Source y GNU LGPL (Lesser General Public License) permitiendo utilizarla para desarrollos comerciales, siempre y cuando estos desarrollos no usen código distribuido bajo alguna licencia GNU.

La razones por las que se eligió wxWidgets son además de sus ya mencionadas características es que cuenta con soporte, documentación en Internet, ayuda en línea, foros, tutoriales en diversos formatos, desarrolladores en la red por lo que se percibe interés y un futuro, cuenta con un libro de 1000 páginas imprimibles de documentación y en línea, sistema flexible a eventos, llamadas a gráficos como líneas, rectángulos con esquinas redondeadas.**[40]**

2.2.3.2.3. JQuery

JQuery es un nuevo tipo de biblioteca o framework de Javascript que permite simplificar la manera de interactuar con los documentos HTML, permitiendo manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a nuestras páginas web. jQuery, al igual que otras librerías, ofrece una serie de funcionalidades basadas en Javascript que de otra manera requerirían de mucho más código. Es decir, con las funciones propias de esta librería se logran grandes resultados en menos tiempo y espacio. La gran ventaja de jQuery es que permite cambiar el contenido de nuestra página web sin necesidad de recargarla, utilizando DOM y AJAX de manera extremadamente sencilla gracias a su sintaxis.

2.2.3.3. Sistema gestor de base de datos

Un Sistema Gestor de base de datos (SGBD) es un conjunto de programas que permiten crear y mantener una Base de datos, asegurando su integridad, confidencialidad y seguridad mediante restricciones o reglas que no se pueden violar. El propósito general de los sistemas de gestión de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos queposteriormente se convertirán en información relevante, para un buen manejo de los datos.

El SGBD es quien se debe encargar de mantenerlas y establecer medidas de seguridad mediante el establecimiento de claves para identificar al personal autorizado a utilizar la base de datos. Requerir de gran capacidad para volúmenes de datos. Los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo. Además debe ser OpenSource y multiplataforma.

2.2.3.3.1. Postgresql

Como propuesta de SGBD se halla el PostgreeSQL basado en software libre, además de ofrecer muchas ventajas para su compañía o negocio respecto a otros sistemas de bases de datos.

Instalación Ilimitada:

- Es frecuente que las bases de datos comerciales sean instaladas en más servidores de lo que permite la licencia. Algunos proveedores comerciales consideran a esto la principal fuente de incumplimiento de licencia. Se puede usar PostgreeSQL, pues no estaría violando acuerdos de licencia, puesto que no hay costo asociado a la licencia del software.**[44]**

Esto tiene varias ventajas adicionales:

- Modelos de negocios más rentables con instalaciones a gran escala.
- No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento.
- Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.**[44]**

PostgreSQL está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo. Posee muchas características que tradicionalmente sólo se podían ver en productos comerciales de alto calibre. **[16]**

2.2.3.4. Otras herramientas que dan soporte al desarrollo del sistema

2.2.3.4.1. Sfdisk

Es una aplicación que se encarga de manipular las tablas de particiones, suministra información sobre las particiones, comprueba las particiones en un dispositivo y con él es posible crear o eliminar particiones y unidades lógicas y definir la partición activa, si es que no lo está. Presenta la ventaja de que se puede automatizar el proceso, creando un script y pasándole todos los parámetros necesarios para realizar el proceso de una forma simple y rápida. **[41]**

2.2.3.4.2. Partimage

Partimage es una aplicación desarrollada para trabajar sobre el sistema operativo GNU/Linux. Fue desarrollada para guardar particiones de cualquier sistema de ficheros en un archivo de imagen. La imagen puede ser comprimida en formato GZIP/BZIP2, con tal de ocupar el mínimo espacio en el disco duro. Partimage también tiene la funcionalidad de realizar el proceso contrario; teniendo un archivo imagen, instalarlo en una partición vacía. **[37]**

2.2.3.4.3. PXE

PXE hace referencia al entorno de ejecución de prearranque (Preboot eXecution Environment) es un entorno para arrancar e instalar el sistema operativo en computadores desde una red. PXE está relacionado con varios protocolos de red como IP, UDP, DHCP y TFTP.**[33]**

2.2.3.5. Herramientas para el trabajo con los sistemas de archivos

Un disco duro puede estar estructurado en varias particiones, cada una de estas puede tener un sistema de archivos distinto, por lo que se hace necesario de contar con herramientas que permitan manipular la información de las particiones y los datos contenidos en cada una de ellas para los sistemas de archivos más utilizados, estos son algunos de ellos: ext2/ext3, reiserfs, xfs, jfs, ntfs y fat.

2.2.3.5.1. E2fsprogs

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el sistema de archivos ext2/ext3. El programa dumpe2fs se utiliza para obtener el espacio libre y utilizado de la partición, e2label para obtener y modificar el nombre, mkfs.ext2 y mkfs.ext3 para crear el sistema de archivos, resize2fs para redimensionar y con e2fsck se chequea y repara una partición.

2.2.3.5.2. Reiserfsprogs

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el sistema de archivos reiserfs. El programa debugreiserfs se utiliza para obtener el espacio libre, el utilizado y el nombre de la partición, reiserfstune para modificar el nombre, mkfs.reiserfs para crear el sistema de archivos, resize_reiserfs para redimensionar y con reiserfsck se chequea y repara una partición.

2.2.3.5.3. Xfsprogs

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el sistema de archivos xfs. El programa xfs_db se utiliza para obtener el espacio libre y el utilizado, y para obtener y modificar el nombre de la partición, mkfs.xfs para crear el sistema de archivos, xfs_growfs para redimensionar y con xfs_repair se chequea y repara una partición.

2.2.3.5.4. Jfsutils

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el

sistema de archivos jfs. El programa jfs_debugfs se utiliza para obtener el espacio libre y utilizado de la partición, jfs_tune para obtener y modificar el nombre, mkfs.jfs para crear el sistema de archivos, mount para redimensionar y con jfs_fsck se chequea y repara una partición.

2.2.3.5.5. Ntfsprogs

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el sistema de archivos ntfs. El programa ntfsresize se utiliza para obtener el espacio libre, el utilizado, para redimensionar, chequear y reparar la partición, y ntfslabel para obtener y modificar el nombre.

2.2.3.5.6. Dosfstools

Es un paquete que proporciona los programas necesarios para realizar diferentes operaciones sobre el sistema de archivos fat. El programa dosfsck se utiliza para obtener el espacio libre, el utilizado y para chequear y reparar la partición, mkdosfs ademas es usado para modificar el nombre y para crear el sistema de archivos.

2.2.3.6. Lenguajes de programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

Un lenguaje de programación permite a uno o más programadores especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico. Una característica relevante de los lenguajes de programación es precisamente que más de un programador puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

2.2.3.6.1. Bash

Bash es un intérprete de comandos de tipo Unix (shell) escrito para el proyecto GNU. Es el shell por defecto en la mayoría de las distribuciones de GNU/Linux de hoy en día. Se encarga de interpretar las órdenes que le demos para su proceso por el kernel. Es el encargado de leer los caracteres tecleados por los usuarios, los interpreta y los ejecuta. Al escribir un comando, es el Shell y no el S.O., quien se encarga de interpretarlo y ordenar que se ejecute. **[16]**

2.2.3.6.2. Perl

Perl (Lenguaje Práctico para la Extracción e Informe) es un lenguaje de programación que fue originalmente desarrollado para la manipulación de texto y en la actualidad es ampliamente utilizado en la administración de sistemas y en el desarrollo Web. Perl tiene una gran potencia en la manipulación de textos debido a que incluye expresiones regulares que facilitan el trabajo con textos. [16]

2.2.3.6.3. Lenguaje C/C++

C es un lenguaje de programación creado en 1969 por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

El C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C. Las principales características del C++ son el soporte para programación orientada a objetos y el soporte de plantillas o programación genérica (templates). Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. [16]

2.2.3.6.4. CSS

Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML. El W3C es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los navegadores. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

2.2.3.6.5. Javascript

JavaScript es un lenguaje de programación interpretado, es decir, que no requiere compilación, es utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y lenguaje C. Es un lenguaje orientado a objetos, ya que dispone de Herencia, la cual se realiza siguiendo el paradigma de programación basada en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad. JavaScript se ejecuta en el cliente al mismo tiempo que las sentencias van descargándose junto con el código HTML.

2.2.3.6.6. Ajax

AJAX, acrónimo de Asynchronous JavaScript And XML es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncronica con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

2.2.3.7. Protocolos de red

Un protocolo es el conjunto de normas que regulan la comunicación (establecimiento, mantenimiento y cancelación) entre los distintos componentes de una red informática. Los protocolos de red son una norma standard -conjunto de normas standard- que especifican el método para enviar y recibir datos entre varios ordenadores.

Los protocolos de red proporcionan lo que se denominan «servicios de enlace». Estos protocolos gestionan información sobre direccionamiento y encaminamiento, comprobación de errores y peticiones de retransmisión. Los protocolos de red también definen reglas para la comunicación en un entorno de red particular.

2.2.3.7.1. DHCP

DHCP (sigla en inglés de Dynamic Host Configuration Protocol) es un protocolo de red que permite a los nodos de una red IP obtener sus parámetros de configuración automáticamente. Se trata de un protocolo de tipo cliente/servidor en el que generalmente un servidor posee una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas van estando libres, sabiendo en todo momento quién ha estado en posesión de esa IP, cuánto tiempo la ha tenido y a quién se la ha asignado después. **[34]**

2.2.3.7.2. TFTP

TFTP son las siglas de Trivial File Transfer Protocol (Protocolo de transferencia de archivos trivial). Es un protocolo de transferencia muy simple semejante a una versión básica de FTP. TFTP a menudo se utiliza para transferir pequeños archivos entre computadoras en una red. **[35]**

2.2.3.7.3. NFS

Network File System (Sistema de archivos de red), o NFS, es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de locales. **[36]**

2.2.3.8. Modelado de sistema

Un modelo es una representación de un objeto, sistema o idea, deforma diferente al de la entidad misma. El

propósito de los modelos es ayudarnos a explicar, entender o mejorar un sistema. Un modelo de un objeto puede ser una réplica exacta de éste o una abstracción de las propiedades dominantes del objeto.

En la actualidad no existen tantas herramientas de modelado de arquitectura, no hay un estándar y el uso de las existentes implica el estudio de una sintaxis especializada. Los lenguajes de descripción arquitectónicos suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos.

En los últimos años se han publicado diversos estudios y estándares en los que se exponen los principios que se deben seguir para la mejora de los procesos de software. Una metodología para el desarrollo de un proceso de software es un conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de Sistemas Informáticos. Por ello escoger la metodología que va a guiar el proceso de desarrollo del sistema es un paso primordial.

Se decidió utilizar como metodología para controlar y planificar trabajo la metodología SXP, por sus características (*ver sección 1.3.1.4*) y las facilidades que aporta a todo el proceso. Como herramienta de modelado de software se propone la herramienta Case Visual Paradigm Community.

2.2.3.8.1. Lenguaje Unificado de Modelado (UML)

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo de software. UML posee formas de modelar conceptos como lo son procesos de negocio y funciones de sistema, además de aspectos concretos como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables. **[27]** Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software pero no especifica en sí mismo qué metodología o proceso usar.

2.2.3.8.2. Visual Paradigm Community

Fue seleccionada la edición Community de Visual Paradigm que es gratuita, soporta la versión 2.0 de UML y permite su extensión mediante la conexión de módulos conectables (plugin) o usando plantillas (templates). Es una herramienta CASE que utiliza UML como lenguaje de modelado. Visual Paradigm Community es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. **[32]**

2.3. Estándares de codificación de SISTCLON

Un estándar de codificación son reglas que se siguen para la escritura del código fuente. De tal manera que otros programadores se les facilite entender tu código (como identificar las variables, las funciones o métodos, etc). Los estandares de codificación elevan la mantenibilidad de nuestro código, sirven como punto de referencia para los programadores, mantienen un estilo de programación y ayudan a mejorar el proceso de codificación, haciéndolo entre otras cosas más eficiente.

2.3.1. Estándar de código para el desarrollo de SISTCLON

➢ **Nomenclatura de clases y métodos**

Los nombres de clases, sus constructores y métodos deben tener su primera letra en mayúscula. Las múltiples palabras deben comenzar con mayúscula a continuación de la anterior. Ejemplo:

```
class Cliente
void CalcularTotal()
```

➢ **Nomenclatura de variables**

Las variables deben contener solo minúsculas, múltiples palabras se unen comenzando por letra mayúscula. Nombres que no sean palabras solo deben ser usadas como iteradores en for() loops. Ejemplo:

```
int variable;
```

```
float CantidadPrimarias;
```

➢**Constantes**

Las constantes siguen las mismas reglas que las variables con la excepción de que son escritas en mayúsculas. Ejemplo:

```
const int MINIMO = 10;
```

➢**Identación**

Se utliza el estilo Allman, también conocido como “ANSI style”. Este estilo ubica las llaves asociadas a una sentencia de control en la siguiente línea, las sentencias son identadas al siguiente nivel de las llaves. La cantidad de espacios con que se identa es de 4 o un TAB. Ejemplo:

```
if(Vacio())
{
    Operacion1();
    Operacion2();
}
```

➢**Comentarios**

Los comentarios dentro de los métodos se realiza con doble backslash. Los comentarios que describen el objetivo de un método o función, clase o archivo, se realiza abriendo el comentario con un backslash seguido de dos asteriscos y para cerrar dos asteriscos seguido de un backslash. Ejemplo:

```
//Comentario sencillo
/**Descripcion mas detallada**/
```

2.3.2. Estándar de Código para el desarrollo de WEB SISTCLON

➢**Nomenclatura de clases y métodos**

Los nombres de clases y sus constructores deben tener su primera letra en mayúscula. Múltiples palabras deben ser separadas con un underscore, no con CamelCased. El resto de los métodos de la clase deben

debe ser escritos enteramente en minúscula.

➢**Nomenclatura de variables**

Las variables deben contener solo minúsculas, múltiples palabras son separadas por underscore. Nombres que no sean palabras solo deben ser usadas como iteradores en for() loops.

➢**Constantes**

Las constantes siguen las mismas reglas que las variables con la excepción de que son escritas en mayúsculas.

➢**Identación**

Se debe usar el estilo Allman style indenting. Este estilo posiciona las llaves asociadas a una entencia de control en la siguiente línea, se identa en el mismo nivel que la sentencia de control. Las sentencias dentro de las llaves se posicionan en el siguiente nivel.

2.4. Estructuración de los componentes

En este epígrafe se describe la arquitectura de software del sistema, explicando el patrón o patrones arquitectónicos a utilizar, la estructuración de los componentes y la interacción entre ellos, así como el modo de integración de los mismos.

2.4.1. Fundamentación de los patrones utilizados

La descripción de la arquitectura es importante y a la vez necesaria para el desarrollo de un sistema de software, pues constituye una expresión del sistema y de su evolución, permite una mejor comunicación entre los clientes y los desarrolladores, así como la evaluación y comparación de arquitecturas de forma consistente. Facilita la planificación, gestión y ejecución de las actividades del desarrollo del sistema ya que es una forma de verificar la correcta implementación de acuerdo a la descripción arquitectural. SISTCLON un software que permite la clonación y distribución de imágenes de sistemas operativos y funciona como

cliente-servidor donde el cliente inicia un sistema operativo usando la tecnología de clientes ligeros y el servidor administra de forma remota todos los clientes.

2.4.1.1. Patrones arquitectónicos

Los patrones arquitectónicos expresan una organización estructural para un sistema de software. Proveen un conjunto de subsistemas predefinidos e incluyen reglas y lineamientos para conectarlos. **[7]**

2.4.1.1.1. Arquitectura cliente/servidor

Esta arquitectura consiste en varios clientes distribuidos en diferentes nodos, conectados en red a uno o varios nodos servidores, donde el servidor puede servir a varios clientes a la vez (*ver sección 1.5.1*). En los nodos clientes generalmente encontramos presentación de usuario y en los nodos servidores la lógica del negocio. La arquitectura cliente/servidor es una forma de dividir y especializar programas y equipos de cómputo de forma que la tarea que cada uno de ellos realiza se efectúa con la mayor eficiencia posible y permita simplificar las actualizaciones y mantenimiento del sistema. **[17]**

Esta arquitectura se propone para SISTCLON en general, este sistema se basa en un clásico modelo cliente/servidor, donde los clientes son los equipos que se van a instalar, y el servidor es la computadora que le proveerá a los clientes las imágenes que se les instalará (*ver sección 1.5.1*). Por lo tanto en estos sistemas se tienen dos tipos de aplicaciones diferentes: una para el cliente y otra para el servidor, donde a cada una de ellas se le define una arquitectura, como se describe en epígrafe siguiente.

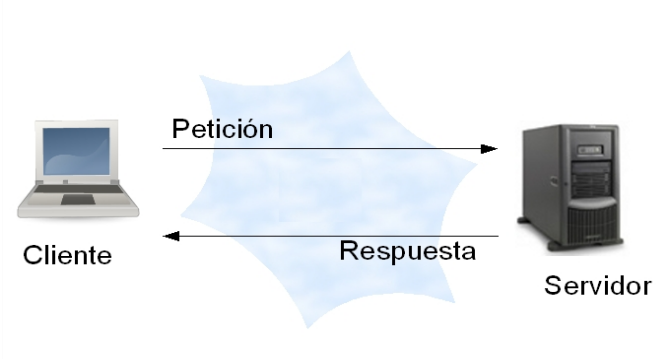


Figura 1. Arquitectura Cliente/Servidor

2.4.1.1.2. Arquitectura en capas

La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La arquitectura de un sistema es la vista conceptual de la estructura de este. Toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos. Un elemento importante dentro de la arquitectura es la forma en que se va a integrar todo el negocio del sistema, para lo cual se debe definir la estructura del software y establecer una estrategia de integración de las diferentes partes o componentes.

Buscando la flexibilidad, reusabilidad y escalabilidad del sistema SISTCLON para el ambiente de escritorio, se propone el diseño de una arquitectura de software distribuida, dicha propuesta es la utilización de una arquitectura cliente/servidor en tres capas, lo cual simplifica la comprensión y la organización del sistema, reduciendo las dependencias de forma que las capas más bajas no sean conscientes de ningún detalle o interfaz de las superiores. La arquitectura propuesta permitirá la interoperabilidad en entornos distribuidos con un nivel de abstracción superior, además de lograr una interfaz de usuario más flexible para el ambiente de escritorio permitiendo que la aplicación sea más simple y escalable.

La calidad tan especial de este estilo arquitectónico consiste en separar la lógica de la aplicación y convertirla en una capa intermedia bien definida. Esta separación entre la lógica de aplicación de la interfaz de usuario añade una enorme flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en absoluto la lógica de aplicación siempre que esté presente una interfaz claramente definida a la capa de presentación.

Para el sistema de clonación SISTCLON las tres capas que se definieron fueron: **Presentación** (Interfaz de usuario), **Lógica de Negocio o Dominio** (tareas y reglas que rigen el proceso), **Acceso a Datos o Gestión de Datos** (Mecanismos de almacenamiento persistente). La capa de Presentación no presenta una complejidad considerable. En la capa de Lógica de Negocio se asocian varios componentes arquitectónicamente significativos para el sistema: RESC-Server, Paquete Particionador, Paquete Imagen, Paquete Configuraciones, Paquete Envío de Mensajes, Paquete Gestión de Información. La capa de Acceso a Datos está definida para la gestión de la persistencia de la información de las computadoras clientes así como para almacenar las imágenes de los sistemas operativos y una descripción de las mismas, para ello se cuenta con dos servidores de base de datos.

Cada capa va a contener un paquete de componentes. Cada paquete agrupa a su vez un conjunto de componentes que son los que van a dar solución al problema. La arquitectura del sistema del lado del servidor quedara como se muestra a continuación:

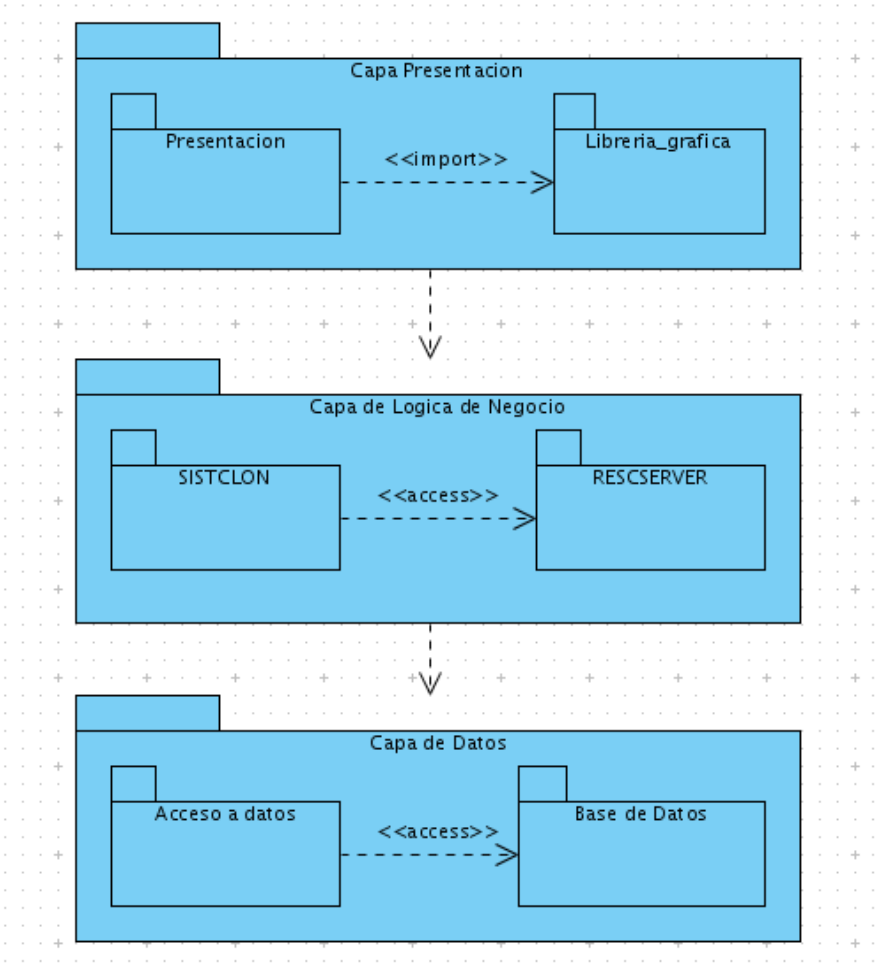
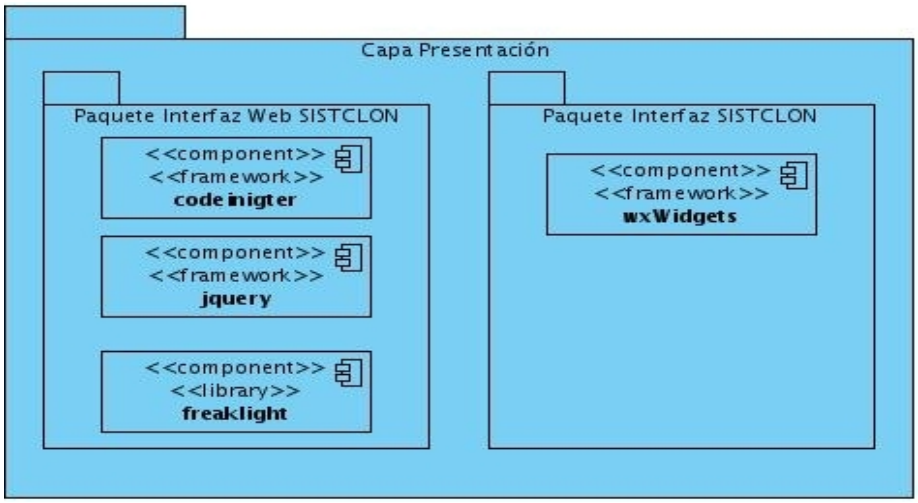


Figura 2. Arquitectura de SISTCLON servidor.

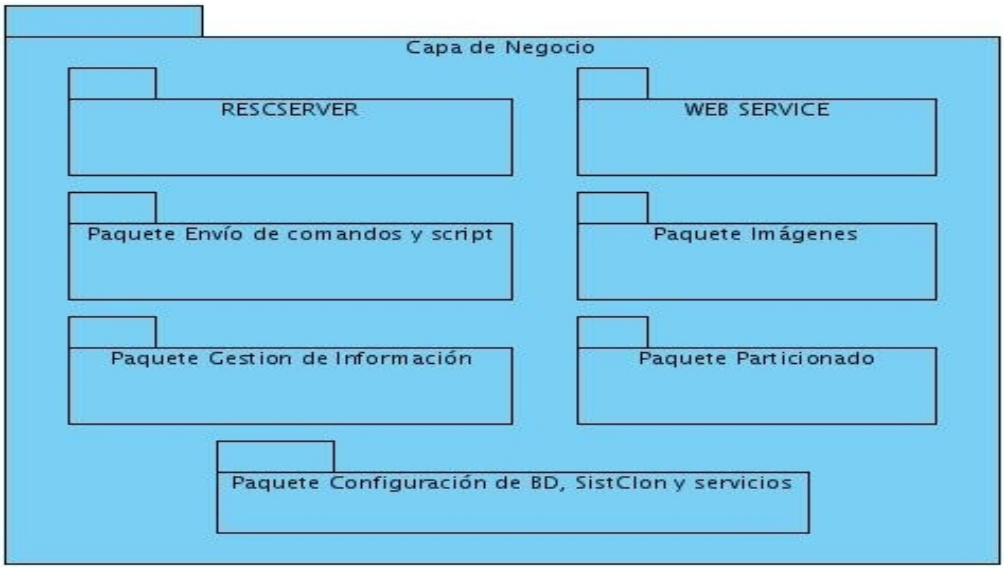
Capa de Presentación

Es la que interactúa directamente con el usuario, captura la información entrada por éste y hace las peticiones a la capa inferior mostrando al usuario la respuesta proveniente de ésta. Únicamente se comunica con la capa de lógica de negocio (*ver figura 2*).



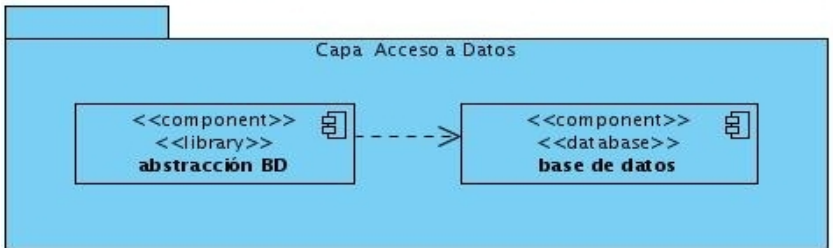
Capa de Lógica de Negocio

Está conformada por los paquetes que integran el sistema, los cuales se ajustan a las historias de usuarios arquitectónicamente significativas y a los requisitos. Desde el punto de vista del diseño, esta capa es contenedora de las clases entidades y controladoras. Únicamente se comunica con la capa de Acceso a Datos (*ver figura 2*).



Capa de Acceso a Datos

Contiene componentes que interactúan con las base de datos y permiten, utilizando los procedimientos almacenados generados, realizar todas las operaciones con las bases de datos de forma transparente para la capa de negocio.



La arquitectura del sistema del lado del cliente quedaría conformada por dos capas: la capa de presentación y la capa de lógica de negocio como se muestra a continuación:

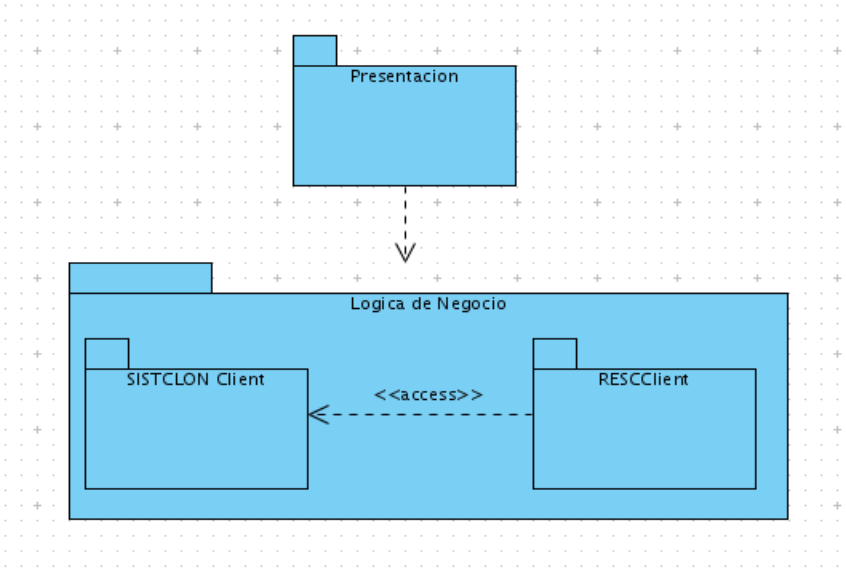


Figura 3. Arquitectura de SISTCLON cliente.

2.4.1.1.3. Modelo Vista Controlador

Para el diseño del ambiente web se utilizó el framework CodeIgniter ya que es verdaderamente liviano, requiere pocas configuraciones, no precisa el uso de herramientas de línea de comando (consola), se recomienda cuando no se necesita de librerías de gran escala como el Repositorio de Extensiones y aplicaciones de PHP. No se precisa ser forzado a aprender un lenguaje de plantilla para su utilización (aunque dispone de uno si se quiere utilizar), además se evita la complejidad, arribando a soluciones más simples. La curva de aprendizaje es sumamente corta. Perfectamente se puede comenzar a utilizar habiendo leído el tutorial de CodeIgniter o las guías online.

Este framework propone el uso de este patrón, quedando la arquitectura para este ambiente de esta forma:

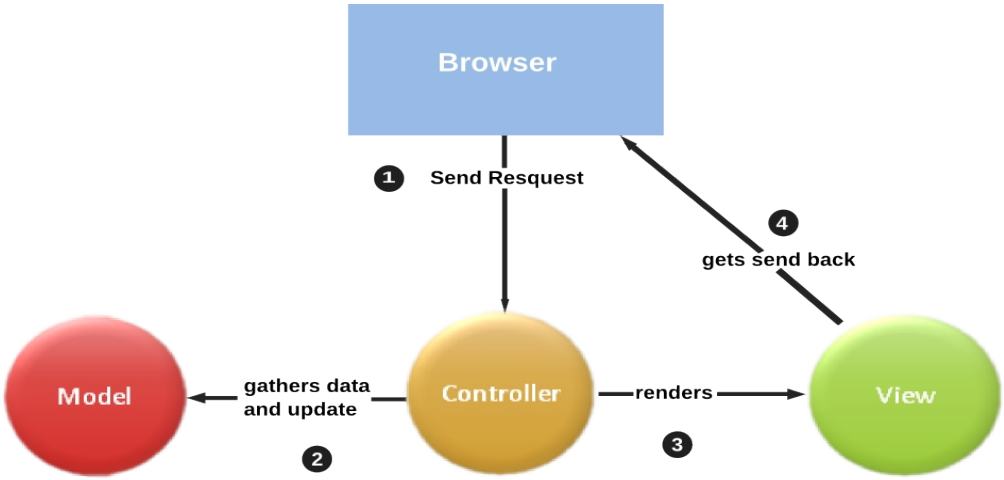


Figura 4. Patrón MVC en el framework CodeIgniter.

- **Modelo:** Es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de datos.
- **Vista**: Presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario.
- **Controlador**: Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo

y probablemente en la vista.

2.4.1.2. Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe 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. **[30]**

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error.

En el modelo de diseño de SISTCLON se tuvieron en cuenta los patrones de asignación de responsabilidades (GRASP) y los patrones GOF que se verán a continuación.

2.4.1.2.1. Experto

El patrón experto en información es el principio básico de asignación de responsabilidades. Nos indica que la responsabilidad de la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Con la utilización de este patrón se definió donde colocar en cada clase las funcionalidades que necesitan de esa información, ya que esta clase sería nuestro experto en información.

2.4.1.2.2. Creador

El patrón creador nos ayuda a identificar quién debe ser el responsable de la instanciación de nuevos objetos o clases. Este patrón se utilizó para identificar qué clase A debe crear elementos de una clase B, apoyándose en que la clase A debería: contener, agregar, registrar, utilizar y tener los datos de inicialización

de la clase B.

2.4.1.2.3. Alta Cohesión

Este patrón nos dice que la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase. En SISTCLON es necesario controlar la complejidad de cada clase utilizada para mantener un buen comportamiento de las mismas, por esto, las clases que fueron identificadas con una gran cantidad de funcionalidades se dividieron en otras clases de manera que se repartiera equitativamente el peso de la complejidad, manteniendo además la coherencia de las clases.

2.4.1.2.4. Bajo Acoplamiento

Este patrón se utilizó con la idea de tener las clases lo menos ligadas entre sí posibles. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

2.4.1.2.5. Controlador

El patrón controlador se utilizó para que sirviera como intermediario entre cada una de las capas, de forma tal que se garantice la comunicación entre los eventos externos del sistema en la capa de presentación y los componentes de la capa de negocio.

2.4.1.2.6. Patrón Solitario (Singleton)

El patrón de diseño se utilizó para garantizar que una clase sólo tenga una única instancia y proporcionar un punto de acceso global a ella. Reduce el espacio de nombres, es una mejora sobre las variables globales, además de que es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos.

2.4.1.2.7. Patrón Fachada (Facade)

El patrón de diseño fachada se empleó para proveer de una interfaz unificada y sencilla, que haga de intermediaria entre el cliente y la interfaz o grupo de interfaces más complejas, permitiendo así una mayor

flexibilidad en el desarrollo del sistema. Permite reducir la complejidad y minimizar las dependencias. Nuestro sistema cuenta con dos interfaces, una para un ambiente de escritorio y otra para un ambiente web por lo que sirvió de mucha ayuda el empleo de este patrón.

2.4.1.2.8. Patrón Observador (Observer)

El patrón Observador define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador se encarga de notificar este cambio a todos los otros dependientes de forma automática. El objetivo de utilizar este patrón es desacoplar la clase de los objetos clientes del objeto, aumentando la modularidad del lenguaje**.**

En el caso del ambiente web, que usa el patrón arquitectónico MVC, se emplea cuando el controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo, aquí se usa el patrón observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio.

2.4.2. Protocolos de comunicación entre nodos físicos

Se conoce como protocolo de comunicación a un conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre sistemas. Los nodos son interconectados entre ellos y a través de estas conexiones navegan paquetes de información. En una red, tienen que trabajar juntos varios protocolos. Al trabajar juntos, aseguran que los datos se preparan correctamente, se transfieran al destino correspondiente y se reciban de forma apropiada.

El trabajo de los distintos protocolos tiene que estar coordinado de forma que no se produzcan conflictos o se realicen tareas incompletas. Los protocolos de comunicación que se establecen son los siguientes:

- **ADO**: Entre el nodo Servidor BD SISTCLON y el nodo Servidor SISTCLON, se encarga de transportar los paquetes que gestionan los datos en el nodo de Servidor SISTCLON.

- **TCP/IP**: Entre los nodos Servidor SISTCLON, el nodo Servidor Web SISTCLON, entre el nodo PC Cliente y el nodo Servidor SISTCLON, este protocolo se encarga de que éstos nodos puedan comunicarse entre sí, así como manejar los datos y proporcionar la fiabilidad necesaria en el transporte de los mismos.
- **HTTP**: Entre los nodos PC Cliente y el nodo Servidor Web SISTCLON, este protocolo es el que empaqueta todo el flujo de interacción de información entre ambos nodos, solicitados por el nodo PC Cliente.

2.4.2.1. Estructura de la distribución física del sistema

La distribución física del sistema define la estructuración física que tendrá el mismo, estas estructuras son representadas a través de nodos que responden a los elementos hardware que intervienen en el negocio y sobre los cuales se ejecutarán los elementos software.

Los nodos son interconectados entre ellos, estableciendo la lógica de transportación de los datos o información. Esta distribución del sistema se representa con un diagrama que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. Los nodos internos indican ambientes, que es un concepto más amplio que el hardware propiamente dicho, ya que un ambiente puede incluir al lenguaje de programación, a un sistema operativo, un ordenador o un clúster de terminales.

La mayoría de las veces la distribución física del sistema implica modelar la topología del hardware sobre el que se ejecuta un sistema a un nivel suficiente para que un ingeniero de software pueda especificar la plataforma sobre la que se ejecuta el software del sistema.

A continuación se muestra la estructura de la distribución física correspondiente al sistema de clonación SISTCLON:

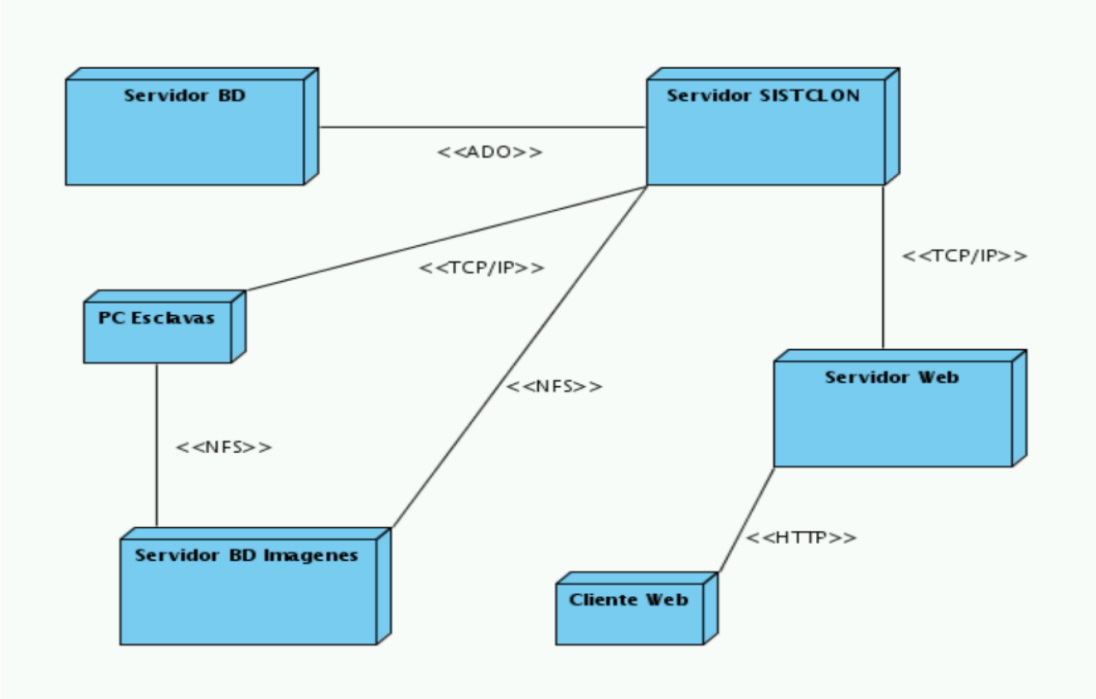


Figura 5. Modelo de despliegue de SISTCLON.

2.5. Estrategia de integración de los componentes de SISTCLON

Todos los componentes cumplen una tarea importante dentro del sistema en general; cada uno de ellos se interrelaciona entre sí de acuerdo a su propósito final en el sistema. Para lograr la integración de los diferentes componentes de un sistema se debe tener en cuenta los componentes creados por el equipo de desarrollo además de los ya existentes.

La utilización de componentes ya existentes es de gran importancia ya que ahorra tiempo de trabajo. Existen componentes independientes que se pueden utilizar sin tener que pasar por el proceso de su construcción, sólo integrarlos con los demás componentes de SISTCLON. Dentro de los componentes existentes que se utilizan en este sistema se encuentran: el PXE, Partimage, RESC-Client, Sfdik, entre otros.

Un componente importante es el PXE *(ver sección 2.5.7)* el cual es usado para iniciar todos los clientes por la red enviándoles un mini sistema operativo con los scripts y aplicaciones necesarias para gestionar, configurar y clonar el ordenador cliente. La aplicación RESC-Cliente se encuentra entre las aplicaciones que posee el sistema de archivos exportado y se encarga de mantener al cliente conectado constantemente al servidor para recibir órdenes del mismo.

Internamente el PXE hace uso de aplicaciones como son el DHCP *(ver sección 2.5.10.1)* para obtener un ip de manera dinámica, el TFTP *(ver sección 2.5.10.2)* para poder exportar el kernel del mini sistema operativo y finalmente el NFS *(ver sección 2.5.10.3)* para exportar el sistema de archivo que cada cliente obtendrá en el inicio por la red y el cual lleva consigo todas las aplicaciones que servirán para realizar el clonado total del sistema.

Otro componente importante es el sfdisk *(ver sección 2.5.4)* el cual se usa para gestionar el particionado de cada computadora cliente. El sfdisk recibe como parámetro de entrada un fichero que contiene la estructura de la tabla de particiones que el administrador del sistema configuró previamente de manera remota.

Después de particionado el cliente, se usa cualquiera de las opciones del mkfs para darle formato a las particiones de acuerdo al tipo de sistema de archivo especificado para cada partición. También se hace uso del componente partimage *(ver sección 2.5.5)* como la aplicación encargada de clonar el sistema operativo que el administrador sirve por NFS. El partimage obtiene la dirección de la imagen que se desea clonar en una partición y se encarga de restablecerla en la partición especificada a clonar. Todos los componentes cumplen una tarea importante dentro del sistema en general; cada uno de ellos se interrelaciona entre sí de acuerdo a su propósito final en el sistema.

El PXE brinda el mini sistema operativo que incluye un grupo de aplicaciones entre ellas el sfdisk y el partimage, además de otras como el RESC-Cliente. Este es el encargado de obtener todas las órdenes del administrador y ejecutarlas. Cuando el administrador envíe la estructura de la tabla de particiones que el desea para cada cliente el RESC-Client se lo comunica al sfdisk haciendo una llamada al mismo y

pasándole por parámetro el fichero con la tabla de particiones. Después de particionar se pasa a formatear la partición atendiendo al sistema de ficheros seleccionado por el administrador haciendo uso de la opción correspondiente del mkfs.

Después de haber realizado el particionado del disco duro de cada cliente el administrador procede a enviar la ubicación de la imagen a clonar y en que partición desea clonarla, esta orden llega al RESC-Client y este a su vez ejecuta el partimage pasándole estos datos mediante comandos y el mismo procede a realizar el restablecimiento del sistema operativo en la partición seleccionada.

Para llevar a cabo la integración de los diferentes componentes ya existentes que se utilizan en el sistema se probaron primeramente de forma independiente, lo cual se conoce como pruebas unitarias o individuales, comprobando el comportamiento de los mismos de forma externa.

Los componentes se organizan de acuerdo a ciertos criterios, que representan decisiones de diseño. En este sentido, diferentes bibliografías apuntan que la arquitectura de software incluye justificaciones referentes a la organización y el tipo de componentes, garantizando que la configuración resultante satisface los requerimientos del sistema.

Para realizar la integración de las diferentes partes se realizó un análisis de las historias de usuarios que aportan mayor valor a la arquitectura del sistema dejando definido el orden de integración de estas unidades de forma tal que se le dé cumplimiento a los requisitos funcionales correspondientes a cada una de éstas. Se establecieron las relaciones e implementaciones necesarias para integrar:

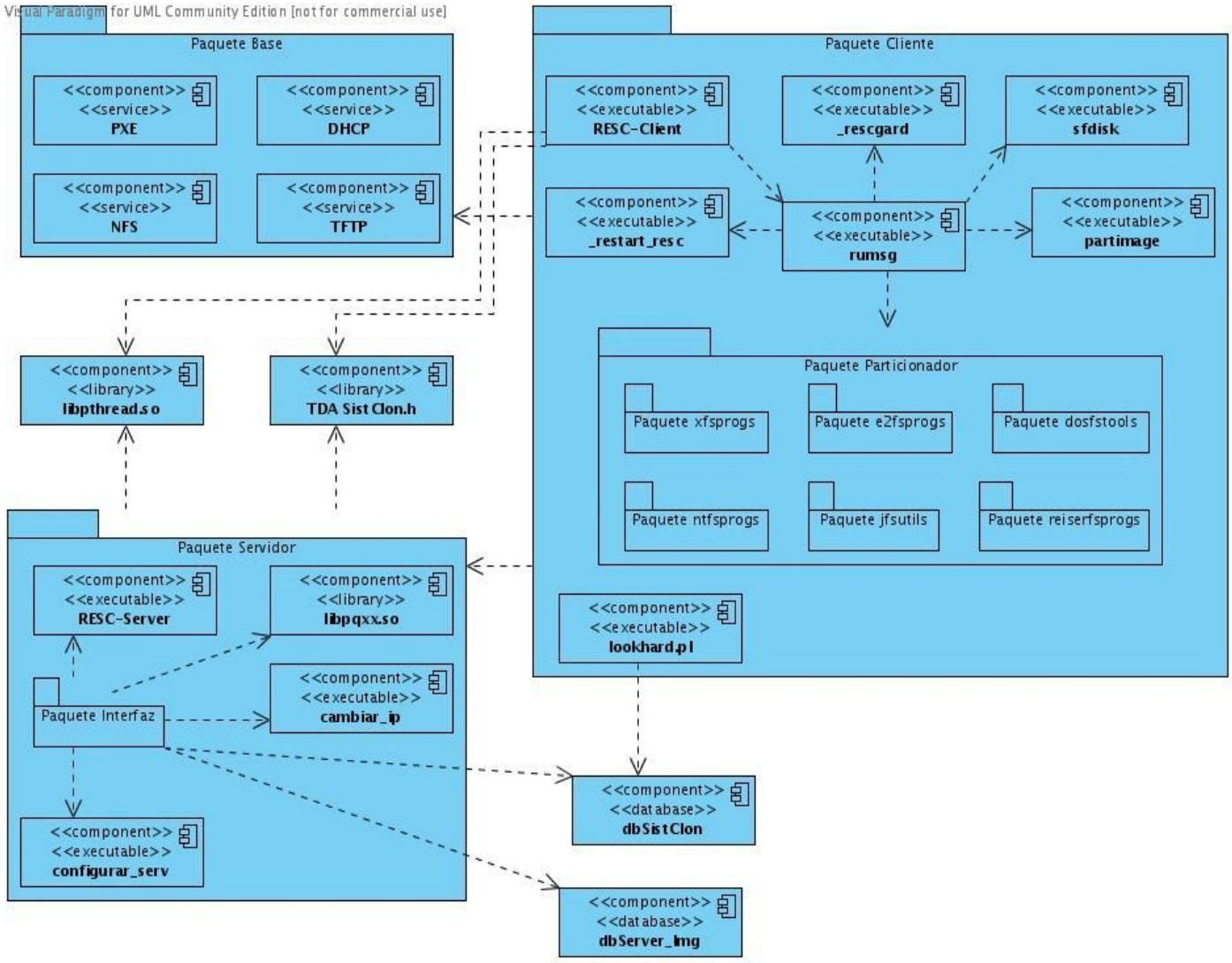
- El RESC-Cliente es el encargado de obtener todas las órdenes del administrador y ejecutarlas. Cuando el administrador envíe la estructura de la tabla de particiones que el desea para cada cliente el RESC-Client se lo comunica al sfdisk haciendo una llamada al mismo y pasándole por parámetro el fichero con la tabla de particiones.
- Después de particionar se pasa a formatear la partición atendiendo al sistema de ficheros

seleccionado por el administrador haciendo uso de la opción correspondiente del mkfs.

- Después de haber realizado el particionado del disco duro de cada cliente el administrador procede a enviar la ubicación de la imagen a clonar y en que partición desea clonarla, esta orden llega al RESC-Client y este a su vez ejecuta el partimage pasándole estos datos mediante comandos y el mismo procede a realizar el restablecimiento del sistema operativo en la partición seleccionada.

2.5.1. Diagrama de componentes

Los diagramas de componentes describen los elementos físicos de un sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos de software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente. A continuación se muestra el diagrama de componentes correspondiente al sistema de clonación SISTCLON.



Firura 5. Diagrama de componentes de SISTCLON.

2.6. Configuración de los puestos de trabajo por roles

Tabla 3. Configuración de los puestos de trabajo por roles.

Rol	Miembro
Gerente (Management)	Yenner Joaquín Díaz Núñez
Cliente (Customer)	Dirección de los laboratorios.
Programadores (Programmers)	Yenner Joaquín Díaz Núñez Angel Guillermo Borjas Almaguer
Analista (Analyst)	Yenner Joaquín Díaz Núñez Angel Guillermo Borjas Almaguer
Diseñadores (Designers):	Yenner Joaquín Díaz Núñez Angel Guillermo Borjas Almaguer
Encargado de Pruebas (Tester)	Yenner Joaquín Díaz Núñez Angel Guillermo Borjas Almaguer
Arquitecto (Architect)	Irina Gonzalez Oduardo

2.7. Conclusiones

En este capítulo se ha definido la arquitectura de software para la propuesta arquitectónica de SISTCLON. Esta solución esta basada en un profundo análisis de los protocolos de comunicación, frameworks, lenguajes de programación, tecnologías que serán utilizadas, además de las herramientas estipuladas que hacen que la arquitectura de software sea robusta. Conjuntamente se plasmaron los requisitos funcionales y requisitos no funcionales de SISTCLON, con los cuales no sería posible llevar a cabo el desarrollo del programa. Además se explicó como se aplica el patrón de arquitectura en capas, MVC y cliente/servidor para dar solución del problema planteado. Finalmente quedó establecida la arquitectura del sistema de clonación.

3. Capítulo 3. Análisis de resultados y validación de la propuesta

3.1. Introducción

La medición es la base de la ingeniería, la ciencia y los negocios. La ingeniería del software ha comenzado con el establecimiento de métricas standard para el desarrollo de software. La medición de software consiste en la colección significativa y precisa de información que tiene valor práctico para el personal de software. El objetivo es proporcionar a los profesionales y administradores de software un conjunto de datos útiles y tangibles para dimensionar, estimar y controlar proyectos de software con rigor y precisión. Entonces, podemos definir las métricas de software o medidas de software como:

"La aplicación continua de técnicas basadas en las medidas de los procesos de desarrollo del software y sus productos, para producir una información de gestión significativa y a tiempo. Esta información se utilizará para mejorar esos procesos y los productos que se obtienen de ellos"

El objetivo del presente capítulo es enumerar un conjunto de métricas del software, dando una breve descripción de los aspectos básicos de cada una.

3.2. Métricas de diseño de alto nivel

Las métricas de alto nivel nos ayudan a localizar los módulos más complejos y, por lo tanto, aquellos en los que debemos poner especial atención. También es utilizada para saber el número de módulos asignados a cada trabajador.

Las métricas de diseño de alto nivel se concentran en las características de la arquitectura del programa, con énfasis en la estructura arquitectónica y en la eficiencia de los módulos. Estas métricas son de caja negra en el sentido que no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema.

3.2.1. Métrica de Card y Glass

Card y Glass definen las siguientes tres medidas de complejidad:

- ***La complejidad estructural***. S(i), de un módulo i se define de la siguientes manera.

$S(i) = f^2_{out}(i)$

donde **$f_{out}(i)$** es la expansión del módulo i. La expansión indica el número de módulos que son invocados directamente por el módulo i.

- ***La complejidad de datos***. D(i) proporciona una indicación de la complejidad en la interfaz interna de un módulo i y se define como :

$D(i) = v(i) / [f_{out}(i) + 1]$

donde **v(i)** es el número de variables de entrada y salida del módulo i.

- ***La complejidad del sistema***. C(i), se define como la suma de las complejidades estructural y de datos, y se define como.

$C(i) = S(i) + D(i)$

A medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas.

3.2.2. Métrica de Henry y Kafura

Henry y Kafura proponen la métrica de complejidad de expansión – concentración, que considera las estructuras de datos que recoge (concentran) o actualizan (expansión). Esta métrica de complejidad se expresa de la siguiente forma:

$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2$$

donde la **longitud(i)** es el número de sentencias en lenguaje de programación en el módulo (i) y **$f_{in}(i)$** es la concentración del módulo i.

Henry y Kafura amplían la definición de concentración y expansión no sólo el número de conexiones de control del módulo (llamadas al módulo), sino también el número de estructuras de datos del que el módulo **i** recoge (concentración) o actualiza (expansión) datos. Como en las métricas de Card y Glass mencionadas anteriormente, un aumento en la métrica de Henry-Kafura conduce a una mayor probabilidad de que también aumente el esfuerzo de integración y pruebas del módulo.

3.3. Resultado de las métricas empleadas

En el presente epígrafe se dan solución a los cálculos planteados en el epígrafe anterior, sobre las métricas propuestas para la evaluación de la arquitectura de software.

3.3.1. Métrica de Card y Glass

Para para efectuar los cálculos la métrica de Card y Glass es necesario conocer de cada módulo la expansión, las variables de entrada y las variables de salida. Para obtener la complejidad del sistema se suman la complejidad estructural y la complejidad de datos (*ver sección 3.2.1*) y si el valor es menor o igual que 32, hay carencia de complejidad; se manifiesta complejidad en el rango de mayor que 32 y menor o igual que 50 y muy complejo para aquellos valores por encima de 50.

Tabla 4. Umbrales de complejidad.

Complejidad	**S(i)**	**D(i)**	**C(i)**
No complejo	1,4,9,16,25	≤ 7	≤ 32
Complejo	36,49,64	> 7 y ≤ 12	> 32 y ≤ 50
Muy complejo	81...	> 12	> 50

Para el cálculo de **S(i)** y **D(i)** es preciso conocer la expansión que estará reflejada por un número entero, puesto que la expansión de un módulo no puede ser expresada por un número real; lo mismo sucede con la cantidad de variables de entrada y salida para el cálculo de **D(i)**. A continuación se definen los módulos que presentan relaciones y se calculan para cada uno de estos, las complejidades anteriormente definidas:

3.3.1.1. Módulo Envío de comandos y script

$S(i) = f^2_{out}(i) = 0$

$D(i) = v(i) / [f_{out}(i) + 1] = 2 / 1 = 2$

$C(i) = S(i) + D(i) = 0 + 2 = 2$

3.3.1.2. Módulo Configuración de BD, SISTCLON y servicios

$S(i) = f^2_{out}(i) = 0$

$D(i) = v(i) / [f_{out}(i) + 1] = 1/ 1 = 1$

$C(i) = S(i) + D(i) = 0 + 1 = 1$

3.3.1.3. Módulo Gestión de información

$S(i) = f^2_{out}(i) = 0$

$D(i) = v(i) / [f_{out}(i) + 1] = 2 / 1 = 2$

$C(i) = S(i) + D(i) = 0 + 2 = 2$

3.3.1.4. Módulo Particionado

$S(i) = f^2_{out}(i) = 1$

$D(i) = v(i) / [f_{out}(i) + 1] = 4 / 2 = 2$

$C(i) = S(i) + D(i) = 1 + 2 = 3$

3.3.1.5. Módulo Imágenes

$S(i) = f^2_{out}(i) = 1$

$D(i) = v(i) / [f_{out}(i) + 1] = 3 / 2 = 1$

$C(i) = S(i)+D(i) = 1 + 1 = 2$

Resultado:

A continuación se muestra una gráfica con el resumen de la sustitución de los valores correspondientes a los módulos previamente vistos, por tanto, se ha podido comprobar que para ninguno de los módulos existe una complejidad estructural considerable.

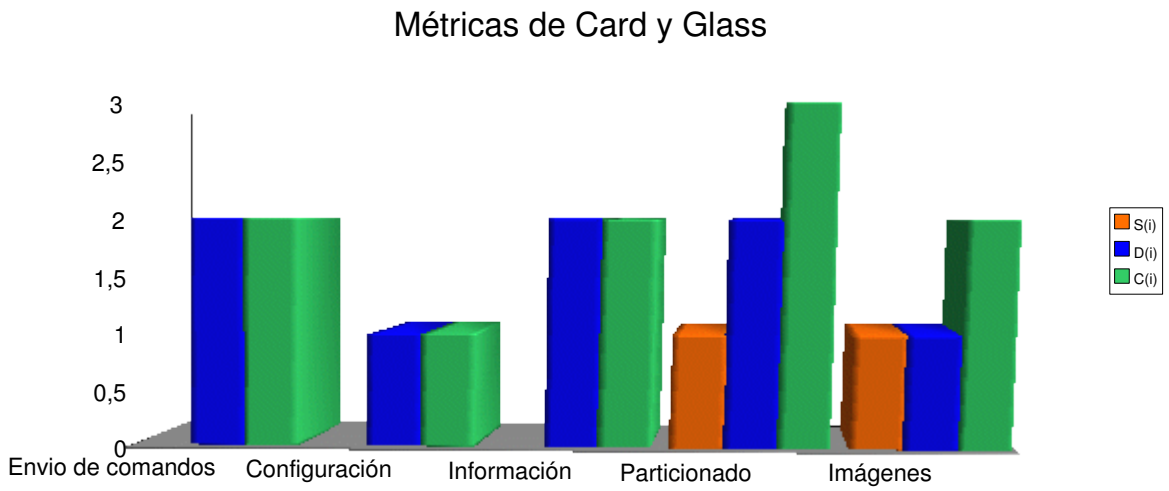


Figura 6 Valores de complejidad para la métrica de Card y Glass.

Según el gráfico se puede comprobar que el módulo más complejo del sistema de clonación es el módulo de Particionado.

3.3.2. Resultado de la métrica de Henry y Kafura

Para desarrollar esta métrica, es necesario conocer una serie de valores esenciales, que al sustituirlos en la fórmula (*ver epígrafe 3.2.2*) aporta el conocimiento del esfuerzo de integración y pruebas que requiere el módulo.

Para calcular el **MHK** durante el diseño puede emplearse el diseño procedimental para estimar el número de sentencias del lenguaje de programación del módulo **i**. El valor referente a $\mathbf{f_{in}(i)}$ responde a la concentración

del módulo **i**, es decir, es la cantidad de módulos de los que **i** recoge datos y para el $f_{out}(i)$ se cumple el mismo principio del epígrafe anterior.

3.3.2.1. Módulo Envío de comandos y script

MHK = longitud(i) x $[f_{in}(i) + f_{out}(i)]^2$

MKH = 215 * $[1 + 0]^2$

MKH = 215 * 1

MKH = 215

3.3.2.2. Módulo Configuración de BD, SISTCLON y servicios

MHK = longitud(i) x $[f_{in}(i) + f_{out}(i)]^2$

MKH = 355 * $[1 + 0]^2$

MKH = 355 * 1

MKH = 355

3.3.2.3. Módulo Gestión de información

MHK = longitud(i) x $[f_{in}(i) + f_{out}(i)]^2$

MKH = 524 * $[1 + 0]^2$

MKH = 524 * 1

MKH = 524

3.3.2.4. Módulo Particionado

MHK = longitud(i) x $[f_{in}(i) + f_{out}(i)]^2$

MKH = 2314 * $[1 + 1]^2$

MKH = 2314 * 4

MKH = 9256

3.3.2.5. Módulo Imágenes

MHK = longitud(i) x [f_{in}(i) + f_{out}(i)]²

MKH = 1060 * [1 + 1]²

MKH = 1060 * 4

MKH = 4240

Resultado:

Esta métrica refleja que el módulo de Particionado es el más complejo y por tanto el que más esfuerzo de integración y pruebas va a requerir dentro del sistema.

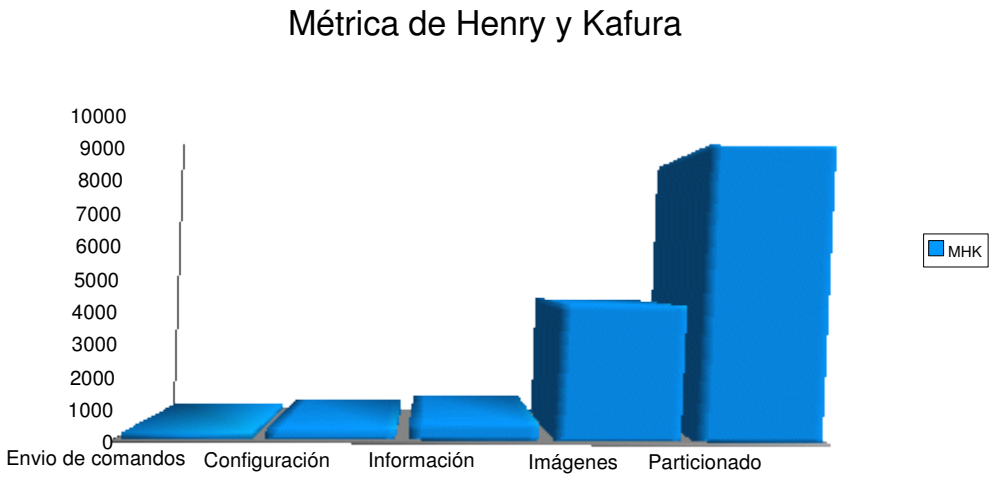


Figura 7 Valores para la métrica de Henry y Kafura.

3.4. Impacto de la arquitectura en el sistema SISTCLON

En los primeros años de la construcción de software no existía el diseño del sistema como una etapa independiente de la programación. A principios de la década del 70 se comienza a pensar en proponer, investigar y aplicar la descomposición de los sistemas antes de programar y cuando distinguir entre

pequeños y grandes programas o sistemas. En el desarrollo de un proyecto de software intervienen muchos factores y si bien la arquitectura tiene un papel protagónico no es la única que decide el éxito del proyecto, pero por esto no deja de ser uno de los pilares que sustenta y lleva al traste un buen producto final.

En la UCI donde cada facultad se orienta a un perfil determinado, la formación docente está vinculada a la producción, esta se desarrolla en los laboratorios docentes y a su vez estos se encuentran equipados con diferentes tipos de computadoras que son atendidas por un grupo de técnicos quienes les dan el mantenimiento e instalan el software necesario. Teniendo en cuenta que para cada facultad las necesidades en cuanto a software que utilizan son diferentes y que las imágenes a instalar en las computadoras tienen que ser personalizadas, lo que provoca que se tengan que hacer varias imágenes.

En un ambiente como este se hace muy engorroso clonar, distribuir las imágenes de los sistemas operativos y administrar las PC clientes manualmente. A raíz de este problema se desarrolló un sistema de clonación y distribución de imágenes de sistemas operativos con el objetivo de automatizar este proceso. Dicho sistema no contaba con una arquitectura bien definida que brindara una flexibilidad, reusabilidad y mantenimiento al mismo. La arquitectura propuesta brinda soporte a la integración de nuevos o futuros componentes y facilita el mantenimiento del sistema.

3.5. Conclusiones

En el transcurso del capítulo se evaluaron las métricas del diseño arquitectónico, ilustrando cuales serían los módulos más complejos o de mayor envergadura tanto en complejidad del sistema como en esfuerzos de integración y pruebas, además de realizar una valoración del impacto de la solución propuesta en SISTCLON.

Conclusiones generales

- Se cumplieron todos los objetivos trazados en esta investigación y a la vez se cumplió con el problema científico a partir de definir una propuesta arquitectónica para SISTCLON.
- Es de vital importancia que todo sistema de software esté respaldado por una arquitectura sólida que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.
- Se realizó un estudio profundo acerca de varios elementos arquitectónicos que existen en la actualidad, para hacer uso de las mejores técnicas de diseño arquitectónico.
- Se desarrolló una arquitectura de tres capas que facilitó el desarrollo paralelo del sistema, el mantenimiento y soporte de la aplicación al tratarse cada capa de forma individual.
- Se definieron componentes y funcionalidades reusables que fueron empleados en cada uno de los módulos permitiendo acortar el tiempo de desarrollo de la aplicación.

Recomendaciones

Con el objetivo de mejorar y ampliar la documentación que se presenta en este trabajo se recomienda:

- Efectuar un continuo refinamiento de la arquitectura de software.
- Aumentar el número de arquitectos a la hora de desarrollar otro proyecto, logrando así una mejor división del trabajo, la cual ayude a realizar un trabajo más rápido, profundo y abarcador sobre la arquitectura de un sistema.
- Valorar la posibilidad de hacer extensiva esta propuesta arquitectónica a otros proyectos.

Referencias Bibliográficas

[1] Shaw, G.D. y M., An Introduction to software architecture. 2000.

[2] Zapata Sanchez, A.F., Arquitectura de Software.

[3] Kruchten, Philippe (1995). The 4+1 View Model of Architecture. IEEE Software.

[4] Penalver Romero, GM. 2008. MA-GMPR-UR2_MA-SXP. 23.10.2008.

[5] IEEE. (2000). "IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems-Description." [Disponible en: http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html]

[6] Mary Shaw, D. G. (1996). Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall.

[7] Carlos Reynoso, N. K. (2004) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [Disponible en: http://www.willydev.net/descargas/prev/Estiloypatron.pdf]

[8] Kent Beck, "Una explicación de la Programación Extrema: Aceptar el cambio", Addison Wesley, 2002.

[9] Garlan M.S.y.D., Software Architecture: Perspectives on an emerging discipline.1996.

[10] Martin, Robert & Newkirk, James: La Programación Extrema en la práctica, 2002 Addison Wesley

[11] Jeffries, R., Anderson, A., Hendrickson, C. "Extreme Programming Installed". Addison-Wesley. 2001

[12] Arquitectura 3 capas. [Disponible en: http://www.slideshare.net/Decimo/arquitectura-3-capas]

[13] Schwaber K., Beedle M., Martin R.C. "Agile Software Development with SCRUM". Prentice Hall. 2001.

[14] Sánchez, M. A. M. Metodologías De Desarrollo De Software. 2002, 25/02/2007. 5 p. [Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf

[15] Kent Beck, "Una explicación de la programación extrema. Aceptar el cambio", Pearson Education, 1999. Addison Wesley, 2000.

[16] Perez Roldan, D. (2008). Sistema de Clonación y Distribución de Imágenes de Sistemas Operativos. La Habana.

[17] Ríos GIL, J.J. Sistemas de Información. [en línea]. http://www.it.uc3m.es/: Departamento de Ingeniería Telemática Escuela Politécnica Superior Universidad Carlos III de Madrid, 13 de octubre de 2005 [citado 23 de febrero de 2007]. [Disponible en: http://www.it.uc3m.es/mcfp/docencia/si/material/1_cli-ser_mcfp.pdf]

[18] Duque Méndez, Néstor D. Conceptos de arquitectura Cliente/Servidor. [Disponible en: http://www.virtual.unal.edu.co] Universidad Nacional de Colombia, 27 de noviembre de 2006 [citado 25 de

abril de 2007]. [Disponible en: http://www.it.uc3m.es/mcfp/docencia/si/material/1_cli-ser_mcfp.pdf]

[19] Jacobson, I., Grady Booch, and James Rumbaugh., El Proceso Unificado de Desarrollo de Software. 1999.

[20] F. DeRemer and H. Cron, "Programming-in-the-large versus programming-in-the-small"

[21] Larman, C., UML y Patrones Introducción al análisis y diseño orientado a objetos. La Habana: Editorial Félix Varela, 2004.

[22] Rising, L., Janoff, N.S. (2000). The Scrum Software Development Process for Small Teams. IEEE Software. [Disponible en: http://members.cox.net/risingl1/Articles/IEEEScrum.pdf]

[23] PostgreeSQL. 2004. [Disponible en:
http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html]

[24] Craig Larman, UML y Patrones: una Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado, Segunda Edición, Tomo I Capítulo 18, Páginas 185-215. Madrid, Prentice Hall, 2003

[25] Roger S Pressman, Ingeniería de Software: Un Enfoque Práctico, McGraw-Hill, 2001

[26] Anastacio Velásquez, Miguel M. "Model View Controller (MVC)". [Disponible en: http://www.informatizate.net/articulos/model_view_controller_mvc_20040324.html], 18 de febrero de 2006.

[27] Larman, Craig. "UML y Patrones. Introducción al análisis y diseño orientado a objetos", Editorial Prentice Hall Hispanoamericana, S.A. México, 1999.

[28] Anjuta Team. Anjuta. [Disponible en: <http://anjuta.org/].

[29] Fowler, M. "Is Design Dead", 2001. [Disponible en: www.martinfowler.com/articles/designDead.html]

[30] Craig Larman. UML y Patrones. 2a edición, Madrid, Prentice Hall, 2003.

[31] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, Patrones de diseño, Pearson Educación, 2003

[32] Visual Paradigm. 2007 [Citado; Disponible en: http://www.visual-paradigm.com/product/vpuml/]

[33] Preboot Execution Environment (PXE) Specification. Septiembre de 1999. [Disponible en: http://www.pix.net/software/pxeboot/archive/pxespec.pdf .

[34] Dynamic Host Configuration Protocol (DHCP). [Disponible en: http://www.ietf.org/rfc/rfc2131.txt].

[35] Trivial File Transfer Protocol (TFTP). [Disponible en: http://www.tftp-server.com/ , http://tools.ietf.org/html/rfc1350].

[36] Network File System Protocol Specification (NFS). [Disponible en: http://tools.ietf.org/html/rfc3530 , http://nfs.sourceforge.net/].

[37] Partimage. [Disponible en: http://www.partimage.org/].

[38] Cristina Gómez, Enric Mayol Sarroca, Antoni Olivé, Ernest Teniente LópezDiseño de sistemas software en UML. Publicado por Edicions UPC, 2003.

[39] Code::Blocks Team. Code::Blocks. [Disponible en: http://www.codeblocks.org]

[40] Wxwidgets [Disponible en: http://www.wxwidgets.org]

[41] Sfdisk [Disponible en: http://linux.die.net/man/8/sfdisk]

[42] CodeIgniter [Disponible en: http://codeigniter.org/]

[43] Arquitectura Orientada a Servicios (SOA) [Disponible en: http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.asp]

[44] PostgreeSQL. 2004 [Citado; Disponible en: http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html]

[45] John Wiley: Introduction to Client / Server Systems: A Practical Guide for Systems Professionals.

[46] Morgan Kaufmann: Transaction Processing Concepts and Techniques.

[47] PATARIN, S., HALES, D., “Why Does BitTorrent Work So Well” “ Some Ideas from Evolutionary Game Theory”, European Conference on Complex Systems, Paris, Francia, Noviembre de 2005, http://cfpm.org/~david/posters/patarin-hales-delis-poster6.pdf [07/09/2008]

[48] BANGEMAN, E., “Study: BitTorrent sees big growth, LimeWire still #1 P2P app”, Ars Technica, Abril de 2008, http://arstechnica.com/news.ars/post/20080421-study-bittorren-sees-big-growth-limewire-still-1-p2p-app.html [28/09/2008]

[49] J.Rosenberg,H.Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[50] B. Quinn, "IP Multicast Applications: Challenges and Solutions", draft-quinn-multicast-apps-00.txt, Internet Draft, November 1998.

[51] R. Hinden, M. O'Dell, S. Deering: IETF 2374: An IPv6 Aggregatable Global Unicast Address Format, Julio 1998.

Bibliografía

Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschman, PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Volumen 1.

Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschman, PATTERN-ORIENTED SOFTWARE ARCHITECTURE. Volumen 2.

Carlos E. Cuesta. Arquitecturas de Software. Ingeniería del Software I, Universidad Rey Juan Carlos.

Society, S. E. S. C. o. t. I. C. (2000). "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems."

Bosch, J. (2000). Design & Use of Software Architectures. Addison Wesley.

Carlos Reynoso, N. K. (2004) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.

Clements, P. (1996). A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design.

Kruchten, P. (1995). The 4+1 View Model of Architecture. IEEE Software.

Mary Shaw, D. G. (1996). Software Architecture. Perspectives on an Emerging Discipline, Prentice Hall.

Robert Monroe, A. K., Ralph Melton y David Garlan (1997) Architectural Styles, design patterns, and objects. IEEE Software.

Roger S Presuman. 2005. Ingeniería del Software, Un enfoque práctico. 5 ed.

Intel Corporation. *Preboot Execution Environment (PXE) Specification.* September 1999. [Disponible en: http://www.pix.net/software/pxeboot/archive/pxespec.pdf]

Sanchez, M. A. M. Metodologías de Desarrollo de Software. 2002. [Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf

F. DeRemer and H. Cron, “Programming-in-the-large versus programming-in-the-small”

Anexos

Lista de reserva de producto (LRP).

Prioridad	Item	Descripción	Estimación	Estimado por
Muy alta				

1	Obtener direcciones IP de los clientes conectados.	0.5	ANA
2	Organizar listado de IP de acuerdo a su hardware.	1	ANA
3	Mostrar listado de IP de los clientes conectados.	0.5	ANA
4	Crear Particiones.	2	ANA
5	Definir sistema de ficheros de la partición creada.	1	ANA
6	Mostrar espacio disponible para particionar.	1.5	PRO
7	Definir tamaño de la partición a crear.	1	ANA
8	Definir tipo de partición a crear (Extendida, lógica, primaria).	1	ANA
9	Eliminar partición.	2	PRO
10	Mostrar estado del disco duro durante el proceso de particionado.	5	PRO
11	Obtener direcciones IP de los clientes conectados(RESC-Client).	3	PRO
12	Mostrar direcciones IP de los clientes conectados(RESC-Client).	1	PRO
13	Enviar comando a cliente(s).	2	PRO
14	Enviar script a cliente(s).	2	PRO
15	Buscar imagen a instalar.	0.5	PRO
16	Escoger la partición donde se instalara la imagen.	1	ANA
17	Enviar imagen a cliente(s).	5	PRO

Alta				
	1	Mostrar información del disco duro de un cliente.	8	ANA
	2	Mostrar información de las particiones del disco duro de un cliente.	2	ANA
	3	Mostrar información de la motherboard de un cliente.	1	PRO
	4	Mostrar la información de la interface de red de un cliente.	1	PRO
	5	Mostrar información del display de un cliente.	1	PRO
	6	Mostrar información de la multimedia de un cliente.	1	PRO
	7	Soporte para Windows.	8	PRO
	8	Configurar los archivos principales del sistema operativo después de instalado. (solo para Linux).	2	PRO
Media				
	1	Permitir realizar una imagen de una partición específica de un cliente de forma remota.	3	PRO
	2	Gestionar tareas pendientes.(RESC)	2	PRO
	3	Soporte para las P5LD2-VM	2	PRO
Baja				
	1	Retroalimentación con el servidor. (RESC)	2	PRO
	2	Listado de aplicaciones instaladas en el sistema operativo.	4	PRO
Requerimientos no funcionales				
	1	Diseñar interfaz gráfica para seleccionar la imagen a clonar en la pc cliente.	2	DIS

	2	Diseñar interfaz gráfica para realizar imagen de un cliente.	3	DIS
	3	El sistema permitirá su extensibilidad, permitiendo agregar nuevas funcionalidades en un futuro.	2	PRO
	4	El sistema debe ser bien documentado de forma tal que en caso de mantenimiento el tiempo que se requiera sea el mínimo.	1	ANA
	5	El sistema deberá funcionar sobre plataforma GNU/Linux.	2	PRO

Plan de Releases.

Release	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	Se obtiene la versión 0.1 de SISTCLON con la terminación de las HU descritas.	3, 1, 2, 5, 4, 6	8
2	Se obtiene la versión 0.2 de SISTCLON con la terminación de las HU descritas	8, 9, 12, 13, 14, 7, 11, 10,15,16	8
3	Se obtiene la versión 0.3 de SISTCLON con la terminación de las HU descritas	17,18,19,20,21,22,23,24,25,26,27 ,28,29	7
4	Se obtiene la versión 0.4 de SISTCLON con la terminación de las HU descritas	30,31,32,33,34	10
5	Se obtiene la versión 0.5 de SISTCLON con la terminación de las HU descritas		8
6	Se obtiene la versión 0.6 de SISTCLON con la terminación de las HU descritas		8

Historias de Usuarios.

Tarea de Ingeniería

Número Tarea: U-SCDSW-24-02	**Número Historia de Usuario:** U-SCDSW-24
Nombre Tarea: Verificar conexiones entrantes.	
Tipo de Tarea : Desarrollo (Desarrollo / Corrección / Mejora / Otra)	**Puntos Estimados:** 0.1
Fecha Inicio: 9/03/2008	**Fecha Fin:** 9/03/2008
Programador Responsable: Angel Guillermo Borjas Almaguer.	
Descripción: Verificar que no exista una doble conexión desde el mismo ip.	

Historia de Usuario	
Número: U-SCDSW-24	**Nombre Historia de Usuario:** Obtener direcciones IP de los clientes conectados.
Modificación de Historia de Usuario Número:	
Usuario: Angel G Borjas Almaguer.	**Iteración Asignada: 5**
Prioridad en Negocio: Media (Alta / Media / Baja)	**Puntos Estimados:** 2
Riesgo en Desarrollo: Bajo (Alto / Medio / Bajo)	**Puntos Reales:** 1
Descripción: Se realiza al iniciar el sistema y consiste en obtener las direcciones IP de los PC clientes que están conectados al RESC-Server.	
Observaciones: El cliente debe el RESC-Client ejecutándose y del lado del servidor debe existir el RESC-Server ejecutándose también.	
Prototipo de interfaces:	

Historia de Usuario

Número: *U-SCDSW-31*	**Nombre Historia de Usuario:** *Gestión del disco duro de los clientes.*
Modificación de Historia de Usuario Número: *ninguna*	
Usuario: *Jorge M. Vázquez Paredes*	**Iteración Asignada:** *6*
Prioridad en Negocio: *Alta*	**Puntos Estimados:** *6*
Riesgo en Desarrollo: *Alto Alto Medio o Bajo*	**Puntos Reales:** _
Descripción: *Gestiona la estructura de la tabla de particiones de uno, varios o todos los clientes conectados, ya sea iniciando una nueva tabla borrando los datos existentes, o a partir de la estructura y datos actuales.*	
Observaciones: *Tiene una interfaz amigable e intuitiva para el usuario.*	
Prototipo de interfaz:	

Historia de Usuario	
Número: *U-SCDSW-34*	**Nombre Historia de Usuario:** *Soportar la clonación de Windows.*
Modificación de Historia de Usuario Número: *ninguna*	
Usuario: *Jorge M. Vázquez Paredes*	**Iteración Asignada:** *6*
Prioridad en Negocio: *Alta*	**Puntos Estimados:** *6*
Riesgo en Desarrollo: *Alto Alto Medio o Bajo*	**Puntos Reales:** _
Descripción: *Soporte para clonar particiones de Windows en los clientes, ya sea tener un cliente con Windows solamente, o acompañado con uno o varios GNU/Linux.*	

Observaciones: *[Alguna acotación importante de señalar acerca de la historia.]*
Prototipo de interfaz:

Historia de Usuario	
Número: *U-SCDSW-33*	**Nombre Historia de Usuario:** *Configuración del sistema operativo clonado en los clientes.*
Modificación de Historia de Usuario Número: *ninguna*	
Usuario: *Jorge M. Vázquez Paredes*	**Iteración Asignada:** *6*
Prioridad en Negocio: *Alta*	**Puntos Estimados:** *6*
Riesgo en Desarrollo: *Alto Alto Medio o Bajo*	**Puntos Reales:** _
Descripción: *Una vez terminada la clonación en un cliente, se procede a la configuración de diferentes archivos fundamentales para garantizar un correcto funcionamiento del sistema operativo clonado.*	
Observaciones: *[Alguna acotación importante de señalar acerca de la historia.]*	
Prototipo de interfaz:	

Glosario de Términos

Red: Sistema de interconexión de ordenadores que permite compartir información y recursos.

TCP/IP: Conjunto de protocolos de red en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras.

Dirección IP: Dirección de un ordenador dentro de una red con protocolo TCP/IP.

Protocolo: Conjunto de normas que rigen un determinado proceso de comunicación.

HTTP: El protocolo por el que se transmiten páginas web a un navegador.

DHCP: Protocolo que permite que un proveedor de servicios de Internet asigne una dirección IP dinámica a un ordenador.

FTP / TFTP: Servicios y protocolos de transferencia de ficheros. TFTP es usado junto con BOOTP en el arranque remoto de sistemas empotrados.

Scrum: Es un proceso de desarrollo de software iterativo e incremental utilizado comúnmente en entornos basado en las metodología Ágiles de desarrollo de software. Está enfocado a la gestión de procesos de desarrollo de software, pero puede ser utilizado en equipos de mantenimiento de software, o en una aproximación de gestión de programas.

XP: (Extreme Programming) la programación extrema o Extreme Programming es el más destacado de los procesos ágiles de desarrollo de software. La misma se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Según su concepto, ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

SXP: Es una metodología compuesta por las metodologías SCRUM y XP que ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permiten actualizar los procesos de software logrando el mejoramiento de la actividad productiva

Perl: (Practical Extraction and Report Languaje) es un lenguaje de programación creado para la administración de sistemas Unix.