Universidad de las Ciencias Informáticas
Facultad 1

“Diseño del Mecanismo de Comunicación de Xerberos”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

- Indira Garcés Pérez

Tutores:

- Ing. Raydel Miranda Gómez
- Ing. Dayron Pérez Roldán

Co-tutor:

- Msc. Alexeis Companionis Guerra

La Habana, junio de 2012
“Año 54 de la Revolución”

DECLARACIÓN DE AUTORÍA

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

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

_____

Indira Garcés Pérez

_____	_____	_____
Tutor	Tutor	Co-tutor
Ing. Raydel Miranda Gómez	Ing. Dayron Pérez Roldán	Msc. Alexeis Companionis

"Sean capaces de sentir, en lo más hondo, cualquier injusticia realizada contra cualquiera, en cualquier parte del mundo. Es la cualidad más linda de un revolucionario".

Ernesto Ché Guevara

AGRADECIMIENTOS

A mi mamita linda, por darme la vida y enseñarme a estudiar, por ser mi fuente de inspiración en todo lo que hago, estar siempre a mi lado y complacerme en todos mis caprichos.

A mi papá, por enseñarme a hacer las cosas bien, guiarme por el camino correcto y motivarme al sentirse siempre orgulloso de su niña linda.

A los dos les dedico mis logros en la vida y este sueño hecho realidad. Les agradezco por permitirme la dicha de crecer en el seno de una familia unida y feliz. Si no fuese por ustedes hoy no estaría escribiendo estas líneas y no tengo palabras para expresarles lo feliz que soy de tenerlos en mi vida.

A mis abuelos, en especial a esa Estrella brillante que con su enorme corazón ha iluminado siempre mi vida, ha endulzado mi niñez y juventud y me ha enseñado que en la vida uno recoge lo que siembra.

A mis hermanos Robe y Jose, por ser los mayores tesoros de mi vida y deseo que este sueño sea algún día el de ellos.

A mi cosi, mi novio Yudisbel, por formar parte de la etapa más importante de mi vida. Por todo su amor, paciencia, apoyo y comprensión. Por secarme las lágrimas y consolarme, por compartir mis alegrías y por todos sus consejos. Por ser partícipe de todas mis locuras, por complacerme en todos mis caprichos, por enseñarme a disfrutar las cosas lindas de la vida y demostrarme que el amor es el arma que nos da la fuerza necesaria para vencer todos los obstáculos.

A mi inmensa familia, por todos los momentos alegres y tristes que hemos vivido juntos y por estar siempre pendientes de mí, en especial, a mis tíos Elisa y Estebita, y a mis primos Nely, Ricky, Luisi, Mandy y Roly.

A la familia de Yudisbel, por acogerme como parte de ellos y por todas las alegrías que hemos compartido, en especial a Cristina, Coralia, Barbarita, Clara, Yilian y Kamila.

A Claudia, por ser mi amiga de la vida, por su confianza y por todas las cosas lindas que vivimos juntas.

A la UCI por darme la oportunidad de formarme como ingeniera y como persona y por ser la principal protagonista de la etapa más importante de mi vida.

A la FEU, esa organización de grandes que jugó un papel fundamental en mi vida universitaria, que sembró en mí un gran sentido de pertenencia, que me demostró que con deseo y dedicación se pueden hacer muchas cosas.

A todos los chicos del secretariado de la FEU y del comité primario de la UJC, a todos los que estuvieron junto a mí en los momentos más difíciles y me ayudaron a que las

cosas salieran bien. En especial a Lisandra, por creer en mí y demostrarme que puedo hacer lo que me proponga, por su apoyo y sus consejos, por escucharme siempre y por su inmensa ayuda en la recta final.

A Reidiel, por ser mi amigo incondicional durante estos 5 años y por todo su apoyo cuando más lo necesitaba.

A todos mis compañeros de aula, en especial a Guille, Machín, Héctor, Ricardo, Rosana, Sandra, Susana, Goar, Adys, Pablo y el Rober, por formar parte importante de todos mis días.

A los chicos del apartamento, Nana, Ana Evis (la patí), Mairim y Angelito (mi sobrino) por todo lo que compartimos estos últimos meses, por secarme las lágrimas y darme mucha fuerza para vencer todos los obstáculos a los que me enfrenté en este tiempo. Muchas gracias por su hospitalidad, apoyo y dedicación incondicional.

A todas mis amistades, Roland, Yeni, Iván, Rosa, Orlenis, Celia, Yoan, Cristina, Silvano, Alionuska, Heney, Daira, Nestor, Elizabeta, que me ayudaron de alguna forma a hacer este sueño realidad.

A mis tutores, Miranda, Dayron y Alexeis, por su ayuda incondicional. En especial a Miranda por acogerme a última hora con los brazos abiertos, por todo el conocimiento que me transmitió y darme la seguridad que necesité para defender este trabajo.

A Haniel por su inmensa ayuda en la confección del ppt.

A todos mis profesores, por siempre creer en mí.

A todos los que de una forma u otra han contribuido a mi formación profesional y humana.

DEDICATORIA

A mis padres y a mi abuela, a ustedes dedico este sueño.

A Yudisbel, mi príncipe azul.

A mi familia y mis amigos.

RESUMEN

Xerberos es un sistema informático que integra gran cantidad de funcionalidades entre las que se encuentran: detectar amenazas de seguridad, auditar acciones, procesos y recursos dentro de la empresa, y realizar actividades de soporte y mantenimiento en estaciones de trabajo. En este resulta de vital importancia que la gestión de la comunicación entre los usuarios y los clientes, se realice de forma rápida y eficaz para lograr que el sistema sea un todo integrado. De la capacidad de establecer la interconexión e interoperabilidad entre los nodos del sistema dependerá la gestión de todas las funcionalidades que brindan sus servicios.

En la actualidad, Xerberos se encuentra en una fase inicial de su desarrollo y el diseño de un mecanismo de comunicación es el primer paso para que el sistema cumpla satisfactoriamente los objetivos que persigue. Este mecanismo se define como: El conjunto de componentes de software que intervienen en la comunicación y son comunes a todos los elementos interconectables de Xerberos.

En la presente investigación se realiza un estudio de los protocolos de comunicación entre procesos y ordenadores existentes, enfatizando en las características de cada uno, para seleccionar cuáles se utilizarán en la comunicación de Xerberos. Además, se realiza el diseño de un mecanismo que proporcionará un marco de trabajo (framework) para la implementación del subsistema de comunicación en cada uno de los nodos que conforman el sistema.

Palabras Claves: mecanismo de comunicación, protocolos, Xerberos.

ÍNDICE

INTRODUCCIÓN

En la actualidad la informática avanza vertiginosamente. Entre los elementos importantes de este avance se encuentran el desarrollo de las redes y de las comunicaciones, que permiten conectar ordenadores con posibilidad de transferencia de datos a alta velocidad.

A medida que las redes utilizadas por empresas y organizaciones se expanden y con ello se adquieren nuevas tecnologías informáticas, la cantidad de puntos de acceso a la red es mayor, a causa de esto, el servicio de soporte y mantenimiento a los sistemas operativos instalados en las estaciones de trabajo se vuelve tedioso. En las empresas, las personas encargadas de realizar esta labor son los técnicos o administradores de red.

En la Universidad de las Ciencias Informáticas existen muchos laboratorios, aulas y oficinas que cuentan con varias computadoras, las cuales son utilizadas en el proceso docente educativo y en la producción de software, conectadas en su mayoría a una red de área local. Además, trabaja un grupo de técnicos informáticos que son los encargados de instalar y brindar soporte a las computadoras de los laboratorios. Para realizar este trabajo, se utilizan software propietarios, que no tienen la calidad requerida y no están adaptados a la diversidad de hardware y software que están presentes en las computadoras de los laboratorios. [1]

Para dar solución a esta problemática, un grupo de proyecto perteneciente al Centro de Software Libre (CESOL) de la facultad 1, desarrolló un sistema libre de instalación remota con el nombre de SistClon, lanzando su primera versión estable en el año 2008. Después de su lanzamiento, el equipo de desarrollo continuó el perfeccionamiento del sistema y se desarrollaron nuevas versiones, pero con el desarrollo de la tecnología, fue surgiendo la necesidad de añadirle nuevas funcionalidades, que SistClon no pudo asumir porque no contaba con una arquitectura lo suficientemente robusta para ello. Por esta razón, se proyectó un nuevo software al que denominaron Xerberos.

Xerberos es un sistema de gestión de ordenadores bajo licencia LGPL, que integra gran cantidad de funcionalidades, las cuales, implementadas en forma de servicios, se encuentran organizadas en diferentes subsistemas, permitiendo una mayor robustez y escalabilidad del software en general. El sistema está compuesto por un servidor central y varios servidores auxiliares (cada uno con características específicas), las estaciones clientes y las estaciones de administración (usuarios) desde las cuales se gestionan todas las estaciones clientes mediante una interfaz que puede ser web o de escritorio.

Por las características de la arquitectura cliente-servidor presentes en Xerberos, los subsistemas que lo conforman se encuentran distribuidos tanto en los servidores como en las estaciones clientes, en dependencia de la función que realizan cada uno de los servicios. En Xerberos resulta de vital importancia que la gestión de la comunicación entre los usuarios y los clientes se realice de forma rápida y eficaz, para lograr que el sistema sea un todo integrado. De la capacidad de establecer la

interconexión e interoperabilidad entre los nodos del sistema dependerá la gestión de todas las funcionalidades que brindan sus servicios.

En la actualidad, el sistema se encuentra en una fase inicial de su desarrollo. Por esta razón, es necesario definir para Xerberos un mecanismo que garantice la gestión de las comunicaciones (envío y recepción de información) entre los usuarios y los clientes, que cumpla con los requerimientos de la arquitectura definida y sea transparente a la interfaz de usuario que se utilice.

Partiendo de esta situación surge el siguiente **problema científico:** ¿Cómo garantizar en Xerberos el envío y recepción de información entre los usuarios y las estaciones clientes?

Para la presente investigación se define como **objeto de estudio:** los mecanismos y protocolos de comunicación entre procesos y ordenadores, quedando enmarcado el **campo de acción** en el mecanismo de comunicación de Xerberos.

Para darle solución al problema científico planteado, se ha trazado como **objetivo general:** "Diseñar un mecanismo para Xerberos que permita la gestión de la comunicación en cada uno de los nodos que lo conforman".

Desglosando el objetivo general en los siguientes **objetivos específicos:**

- Analizar los mecanismos de comunicación entre procesos y ordenadores existentes.
- Realizar el diseño del mecanismo de comunicación de Xerberos.

La **idea a defender** en el presente trabajo de diploma radica en: El diseño del mecanismo de comunicación de Xerberos proporcionará a los desarrolladores una visión correcta del mismo para su futura implementación.

Para dar cumplimiento a los objetivos específicos planteados se definen las siguientes **tareas de investigación:**

- Investigación acerca de los mecanismos de comunicación entre procesos y ordenadores, enfatizando en las características, ventajas y desventajas que tiene el empleo de cada uno de ellos.
- Selección de la metodología de desarrollo y las tecnologías adecuadas para el desarrollo de la aplicación. Proponer
- Elaboración del diseño del mecanismo de comunicación de Xerberos.

Los métodos científicos de investigación son la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. Se clasifican en teóricos y empíricos, los cuales están dialécticamente relacionados. [2]

En el proceso investigativo se emplean como métodos teóricos el **Histórico-Lógico** y el **Analítico-Sintético**. El Histórico-lógico plantea que los fenómenos no suceden al azar, sino que son producto de un proceso que los origina, motiva o simplemente da lugar a su existencia. Este método posibilitó el estudio de cómo se realiza la comunicación en el sistema informático SistClon, que fue el antecedente de Xerberos, lo que permitió conocer de forma general el funcionamiento y desarrollo del mismo.

El Analítico-sintético permite la división mental del todo en sus múltiples relaciones y componentes. Además establece mentalmente la unión entre las partes previamente analizadas, posibilita descubrir las relaciones esenciales y características generales entre ellas. La síntesis se produce sobre la base de los resultados obtenidos previamente en el análisis. Este método permitió el estudio de los mecanismos y protocolos de comunicación existentes para identificar cuáles pueden ser útiles en la solución del problema planteado.

El documento se estructura en tres capítulos donde se recoge todo el proceso de investigación y diseño del mecanismo propuesto, los cuales se describen a continuación:

Capítulo 1. Actualidad de Xerberos. Mecanismos de comunicación entre procesos y ordenadores

En este capítulo se hace un análisis de la estructura general de Xerberos y se realiza un estudio profundo sobre los mecanismos de comunicación entre procesos y ordenadores. Además, se describe el lenguaje de programación, las librerías y herramientas que se utilizarán en el diseño y futura implementación del mecanismo.

Capítulo 2. Características del mecanismo de comunicación de Xerberos

Este capítulo expone las características del sistema. En él se plantea el modelo de dominio y los requisitos funcionales y no funcionales del mecanismo de comunicación. Además, se realiza la definición de las Historias de Usuario y las Tareas de Ingeniería.

Capítulo 3. Diseño del mecanismo de comunicación de Xerberos

En este capítulo se realiza el diseño del mecanismo de comunicación de Xerberos, utilizando una metodología de desarrollo ágil. Además, se describe la arquitectura a utilizar y los patrones arquitectónicos y de diseño, se muestra el diagrama de clases del diseño y se realiza la descripción detallada de cada una de las clases que componen el mencionado diagrama.

CAPÍTULO 1. ACTUALIDAD DE XERBEROS. MECANISMOS DE COMUNICACIÓN ENTRE PROCESOS Y ORDENADORES.

En el presente capítulo se precisan los elementos teóricos que sustentan la investigación y el desarrollo del tema propuesto. Se realiza un análisis de las características del software SistClon, que fue el antecedente de Xerberos, y se describen las funcionalidades de este último. Se estudian profundamente los mecanismos y protocolos de comunicación entre procesos y ordenadores existentes, enfatizando en las ventajas, desventajas y características de cada uno de ellos, con el objetivo de seleccionar posteriormente cuáles se utilizarán en la comunicación de Xerberos. Además, se describe el lenguaje de programación, las librerías y herramientas que se utilizarán para el diseño y futura implementación de la propuesta.

1.1. Antecedentes de Xerberos

SistClon es un Sistema de Clonación y Administración Centralizada de Imágenes de Sistemas Operativos Libres desarrollado con el objetivo de dar solución a una problemática existente en la UCI hace un tiempo, donde no era posible instalar y administrar las imágenes de las computadoras utilizadas para la docencia de manera automática con aplicaciones libres, que tuvieran las mismas o más prestaciones que las aplicaciones propietarias existentes en el mercado.

SistClon en sus inicios fue concebido como un software orientado al mantenimiento y la administración remota de un conjunto de estaciones de trabajo. Su principal objetivo es la clonación de imágenes de sistemas operativos, pero además, integra funcionalidades para la realización de auditorías de hardware. Posee soporte para diferentes tipos de tarjetas madres (motherboards), usa la tecnología de clientes ligeros para iniciar las estaciones de trabajo que deben ser clonadas y es capaz de gestionar particiones e instalar 9 sistemas de archivos diferentes.

Entre las principales funcionalidades de SistClon se encuentran:

- Auditoría de Hardware.
- Clonación de Sistemas Operativos.
- Gestión y configuración de discos duros.
- Gestión remota de estaciones de trabajo.
- Gestión de configuraciones de estaciones de trabajo.
- Gestión de imágenes de sistemas operativos.
- Servicio de cliente ligero. [1]

En SistClon la interfaz gráfica se encuentra en el servidor. La comunicación entre esta y el RESC_Server se realiza mediante una cola de mensajes. El RESC_Server es la aplicación en el

servidor que establece la comunicación con los clientes utilizando socket por TCP/IP. Las estaciones clientes tienen corriendo un demonio llamado RESC-Client, el cual se encarga de gestionar la comunicación y utiliza una cola de mensajes para comunicarse con el rumsg, aplicación que tiene la responsabilidad de ejecutar todas las órdenes que se reciben del servidor. En el *anexo 1,* se muestra de forma gráfica lo descrito.

La forma en que se realiza la comunicación en SistClon, anteriormente explicada, presenta algunas desventajas que atentan contra el correcto funcionamiento del sistema, entre ellas:

- La comunicación entre el servidor y la interfaz gráfica se realiza utilizando cola de mensajes, por esta razón, es obligatorio que estos estén instalados en el mismo ordenador. De esta forma, la interfaz gráfica solo puede ser instanciada una sola vez, reduciendo la posibilidad de que el sistema sea administrado al mismo tiempo por varios usuarios. Además, no existe una forma de autenticación con el servidor.
- La comunicación entre las estaciones y el servidor se realiza sobre un canal no cifrado y no existe un certificado digital que permite identificar que cada cliente es quien realmente dice ser.
- Del lado del cliente, todas las órdenes son ejecutadas por una misma aplicación (rumsg) lo cual obliga a construir las órdenes a manera de scripts del lado del servidor, siendo mucho más beneficioso, separar la ejecución de las órdenes en servicios individuales con funcionalidades específicas y de esta forma poder consultar los resultados de las mismas y llevar un registro (log) con las operaciones realizadas.

La primera versión estable de SistClon fue liberada en el año 2008. Posteriormente, el equipo de proyecto continuó el perfeccionamiento del sistema y se desarrollaron nuevas versiones, que han sido utilizadas por el grupo de técnicos informáticos de la UCI para instalar los ordenadores de los laboratorios docentes. Con el desarrollo de la tecnología y la experiencia adquirida por los desarrolladores en este campo, fue surgiendo la necesidad de añadirle nuevas funcionalidades para que el sistema de instalación cumpliera con las prestaciones actuales que tiene este tipo de software en el mercado. Sin embargo, la principal desventaja de SistClon es que no posee una arquitectura suficientemente robusta como para integrarle nuevas funciones que le permitan compararse con estos. Para evitar este inconveniente, sus desarrolladores proyectaron un nuevo software denominado Xerberos, con muchas más funcionalidades y con una arquitectura mejor diseñada, que lo hace muy flexible a la hora de añadirle nuevas opciones.

1.2. Actualidad de Xerberos

Xerberos es un sistema informático en proceso de desarrollo que permite detectar amenazas de seguridad como: uso indebido de los recursos de la red, acceso ilegal a estaciones o terminales, daños maliciosos y fuga de información sensible. Integra funcionalidades para auditar acciones, procesos y

recursos dentro de la empresa brindando notificaciones de posibles modificaciones y actividad indebida en los mismos. También permite realizar actividades de soporte y mantenimiento en estaciones de trabajo, gracias a un sistema de clonación y administración remota de imágenes de sistemas operativos, permitiendo gestionar de forma centralizada las particiones y configuraciones, así como la instalación y desinstalación de software en cada estación. Su interfaz amigable y sencilla permite un fácil acceso a cada funcionalidad del sistema y a los reportes. [3]

En este nuevo sistema, el resultado del trabajo anterior no se desecha. SistClon pasa a ser un subconjunto de Xerberos y se convierte en el subsistema de clonación, lo que constituye un gran adelanto en la implementación.

Entre los diferentes subsistemas que agrupan y organizan todo el conjunto de funcionalidades que brinda Xerberos se encuentran:

1. **Subsistema de Registros:** Permite almacenar información en archivos de texto tanto en el servidor como en los clientes. Otros subsistemas hacen uso del mismo para hacer depuraciones o dar información de salida. La utilidad radica en no usar la consola o salida estándar para dar mensajes, además, permite realizar inspecciones después de la ejecución de cualquier servicio; facilitando de esta forma la detección de posibles fallas o errores en los mismos.
2. **Subsistema de Base de Datos:** Constituye un puente entre la aplicación de administración central (Desktop y Web), los servicios, y las bases de datos que los mismos utilizan. Otros módulos se valen de él para recuperar o actualizar información en las bases de datos.
3. **Subsistema de Clonación:** Permite el mantenimiento y la administración remota de clientes en cuanto a hardware y software. Su principal objetivo es la clonación de sistemas operativos GNU/Linux, además, permite particionar los discos duros.
4. **Subsistema de Monitoreo y Control:** Se encarga de la gestión de seguridad de la información interna. Permite detectar amenazas de seguridad como uso indebido de los recursos de la red, acceso ilegal a estaciones o terminales, daños maliciosos y fuga de información sensible. Además, permite el monitoreo constante sobre el uso de aplicaciones y dispositivos externos de almacenamiento por los usuarios en las estaciones clientes. Utiliza el Módulo de Registros y el Módulo de Base de Datos.
5. **Subsistema de Auditoría de Recursos:** Encargado de la auditoría de seguridad de los activos que se implementa recopilando información sobre los activos de software y hardware de la empresa en forma dinámica, rastreando los cambios de los activos y permitiendo que los administradores comprendan el estado de los activos, mejorando así la capacidad de gestión informática de la empresa. Utiliza el Módulo de Registros y el Módulo de Base de Datos.
6. **Subsistema de Reportes:** Encargado del diseño y elaboración de listados o reportes basados en las distintas tablas de la base de datos. Utiliza el Módulo de Registros y el Módulo de Base de Datos. Los reportes, además de auxiliar a la empresa u organización proporcionando información específica sobre cada área, también pueden utilizarse como estadísticas para la

toma de decisiones.

7. **Subsistema de Comunicación:** Eje central en la comunicación (envío y recepción de información) entre el/los programa(s) servidor (y sus módulos) y los clientes. Su funcionalidad radica en facilitar el envío y recepción de información entre el/los programa(s) servidor (y sus módulos) y los clientes (y sus módulos) a través de la red. [3]

Funcionamiento de Xerberos

Xerberos está compuesto por varios nodos, entre ellos: un servidor central y varios servidores auxiliares (cada uno con características específicas), las estaciones clientes y las estaciones de administración (usuarios), desde las cuales se gestionan todas las estaciones clientes desde una interfaz que puede ser web o de escritorio. El sistema se ejecutará en un número indefinido de nodos, los cuales únicamente se clasificarán en la arquitectura como clientes y servidor. El servidor central es único, mientras que la cantidad de clientes puede ser numerosa (el número máximo de clientes está limitado principalmente, por el ancho de banda de la conexión del servidor). Los usuarios se comunican con el servidor a través de la red, pero dos clientes nunca pueden comunicarse entre ellos. [3]. La figura 1.1 muestra la relación entre los nodos y el funcionamiento del sistema.

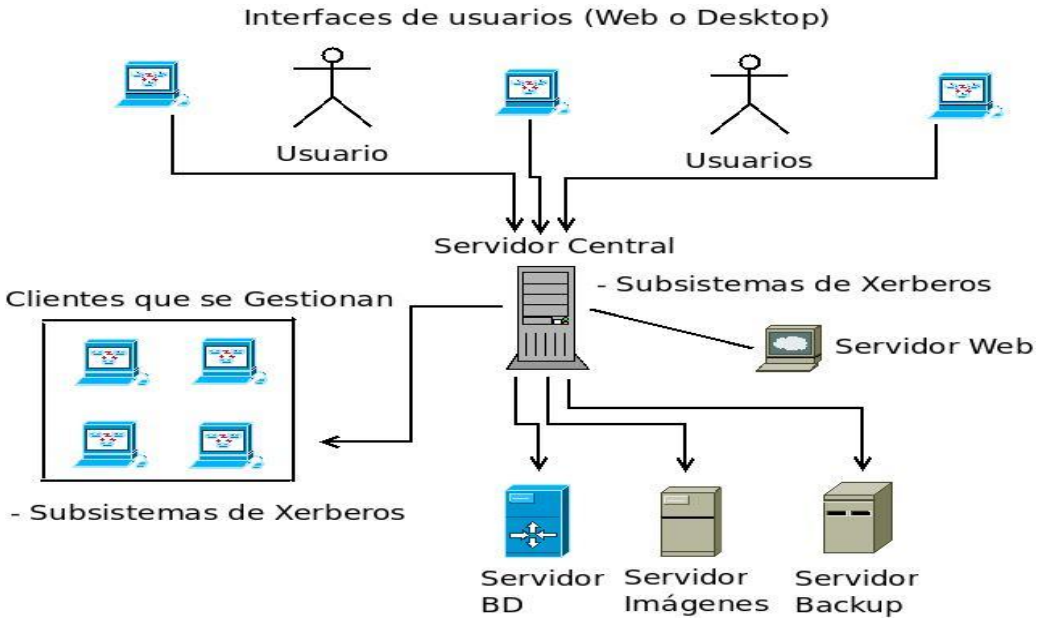


Figura 1.1: Componentes físicos de Xerberos y sus relaciones.

Por las características de la arquitectura cliente-servidor presentes en Xerberos, los subsistemas que lo conforman se encuentran distribuidos tanto en los servidores como en las estaciones clientes, en dependencia de la función que realizan cada uno de los servicios que lo integran. La arquitectura del sistema se divide en varios plugins. En Xerberos, un plugin no es más que una librería dinámica con extensión .so, la cual contiene el código necesario para establecer la comunicación con uno de los servicios del sistema, de manera que, cuando se desee ejecutar alguna funcionalidad, se carga el

plugin correspondiente y se hace uso de los métodos o funcionalidades que brinda el mismo, sin tener en cuenta las características de la aplicación desde la que se accede.

1.3. Mecanismos y protocolos de comunicación entre procesos y ordenadores

La **comunicación entre procesos**, en inglés IPC (*Inter-process Communication*) es una función básica de los sistemas operativos la cual provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí. [4]
Un **mecanismo de comunicación entre procesos** es un conjunto de elementos de software definidos que se relacionan y posibilitan que los procesos se comuniquen entre sí.
La **comunicación entre ordenadores** es la transmisión de datos e información a través de un canal de comunicación entre dos computadoras y se logra mediante la utilización de redes. [5]
Los **protocolos de comunicación** son un conjunto de reglas que permiten la transferencia e intercambio de datos entre los distintos dispositivos que conforman una red. [6]

Existen distintos mecanismos y protocolos de comunicación entre procesos y ordenadores, entre ellos:

Tuberías (Pipes)

Son un mecanismo para la comunicación de datos entre procesos que permiten conectar automáticamente la salida estándar de un programa con la entrada estándar de otro programa. [7] Para mejorar el rendimiento, la mayoría de los sistemas operativos implementan las tuberías utilizando espacios en memoria (buffers), lo que permite al proceso proveedor, generar más datos que los que el proceso consumidor puede atender inmediatamente. Por esta razón, son muy usadas en los sistemas operativos para garantizar la multitarea de los mismos y en lenguajes de programación como Perl, Bash, C, Python, entre otros.
Las tuberías son la manera más fácil de comunicación entre procesos y son muy utilizadas mediante el símbolo "|", el cual permite comunicar dos procesos por medio de una tubería desde la terminal o línea de comando de Linux o Windows.

Ventaja de las tuberías:

- Además de permitir la comunicación, funcionan para transferir información entre procesos o hilos.

Desventajas de las tuberías:

- Son unidireccionales, es decir, el proceso que envía una salida a otro proceso, no puede a la vez, recibir una entrada de este, por lo que la comunicación que se establece es en una sola

dirección.

- El envío de información que permiten es muy básico.
- Solo pueden ser utilizadas en la comunicación entre procesos en un mismo ordenador.

Cola de mensajes

Las colas de mensajes son uno de los mecanismos más utilizados para comunicar procesos y pueden ser consideradas como una lista de mensajes enlazados. Permiten que un número de mensajes, cada uno de una longitud variable, sean ordenados en una estructura de cola (FIFO o basada en prioridades) a la espera de ser leídos. Una vez que la cola ha sido creada, cualquier proceso puede escribir o leer mensajes de esta. [8]

Características de las colas de mensajes:

- Las colas de mensajes permiten efectuar comunicación asíncrona, indirecta y simétrica entre hilos o procesos.
- Una cola de mensajes puede tener varios escritores y lectores.
- Las colas tienen nombre.
- Cada cola tiene un tampón asociado:
  - si se llena, se bloquea el emisor (se puede anular).
  - el receptor se bloquea si el tampón está vacío.

Ventaja de las colas de mensajes:

- Permiten transferir información entre procesos o hilos.
- Pueden tener varios escritores y lectores.

Desventaja de la comunicación mediante mensajes:

- Solo pueden ser utilizadas en la comunicación entre procesos en un mismo ordenador.

Socket

Los sockets son un mecanismo de comunicación que constituyen un componente básico de la comunicación interprocesos e intersistemas, representan un extremo de una comunicación bidireccional y tienen asociada una dirección IP, un protocolo y un número de puerto. Su función principal es permitir a dos programas (posiblemente situados en computadoras distintas) intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada. Su característica más importante es permitir la implementación de una arquitectura cliente-servidor, pues la comunicación es iniciada por uno de los programas que se denomina programa cliente y el segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor.[9]
El puerto que utiliza un socket debe ser mayor de 1024 y menor de 49151, pues estos son los llamados puertos registrados y son los de libre utilización.

Para comunicarse con otro proceso usando sockets debe conocerse:

1. Dirección IP (32 bits) de la máquina donde se ejecuta el proceso.
    - Alternativamente: su nombre para consultar el servicio de nombre DNS (traducción dominio-IP).
2. Número de puerto que utiliza el proceso en su máquina.

Existen varios tipos de sockets, los más importantes son:

1. Socket de Flujo (Stream): utiliza el protocolo de transporte TCP y establece una conexión entre los dos extremos que permite enviar flujos de bytes en ambas direcciones. Es fiable pues se asegura el orden de entrega de los mensajes.
   Existen dos papeles definidos implícitamente:
    - sockets de servidor: esperan recibir conexiones.
    - sockets de tráfico: inician conexiones (en los clientes) y envían/reciben datos (en clientes y servidor).
2. Socket de Datagrama (Datagram): utiliza protocolo de transporte UDP y no se establece conexión, cada datagrama es independiente. No es fiable pues no se asegura el orden en la entrega y mantiene la separación entre mensajes.

Ventajas del uso de Sockets:

- Proporcionan una comunicación de dos vías entre dos ordenadores.
- Permite maximizar la flexibilidad a la hora de desarrollar aplicaciones porque se tiene el control total de los datos enviados.
- Tiene gran disponibilidad pues la interfaz Socket está disponible en múltiples sistemas operativos y lenguajes de programación.

Desventaja del uso de Sockets:

- Se necesita realizar un manejo explícito de los datos a transmitir y es complejo de implementar.

RPC (Remote Procedure Call / llamada a procedimiento remoto)

El RPC fue inventado por la empresa SUN Microsystems y puede ser usado por medio de los protocolos TCP/IP o UDP/IP. Es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. [10]

Ejemplos de entornos RPC:

- Sun-RPC (ONC-RPC: Open Network Computing-RPC): RPC muy extendido en entornos Unix, infraestructura sobre la que se ejecuta NFS (servicio de sistema de ficheros en red), NIS (servicio de directorio).
- DCE/RPC (Distributed Computing Environmen RPC): RPC definido por la Open Software Foundation.

- Java-RMI: invocación de métodos remotos en Java.
- CORBA (Common Object Requesting Broker Architecture): soporta la invocación de métodos remotos bajo un paradigma orientado a objetos en diversos lenguajes.
- SOAP (Simple Object Access Protocol): protocolo RPC basado en el intercambio de datos (parámetros + resultados) en formato XML.
- DCOM (Distributed Component Object Model): Modelo de Objetos de Componentes Distribuidos de Microsoft, con elementos de DCE/RPC.
- .NET Remoting: infraestructura de invocación remota de .NET. [11]

Ventajas de RPC:

- El protocolo es un gran avance sobre los sockets usados hasta el momento.
- El programador no tiene que estar pendiente de las comunicaciones, estando estas encapsuladas dentro de las RPC, pues el objetivo del mismo es ocultar/abstraer los detalles relativos a la comunicación que son comunes a diversas aplicaciones.
- Las RPC son muy utilizadas dentro del paradigma cliente-servidor.

Desventaja de RPC:

- Las comunicaciones RPC se basan en la idea de que el receptor está operativo para poder invocar una cierta función, no se puede suponer que el receptor siempre estará operativo y esperando a comunicarse.

XML-RPC

Es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes. Fue creado por Dave Winer de la empresa UserLand Software en asociación con Microsoft en el año 1998. XML-RPC utiliza las peticiones POST de HTTP para enviar un mensaje, en formato XML, señalando el procedimiento que se va a ejecutar en el servidor y los parámetros, y el servidor devuelve el resultado en formato XML.

Ventajas del XML-RPC:

- Es un protocolo muy simple ya que solo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión.
- La simplicidad del XML-RPC está en contraste con la mayoría de los protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso. [12]
- Es sencillo de implementar.

Desventaja del XML-RPC:

- Funciona sobre Internet.

Algunas implementaciones de XML-RPC:

- XMLRPC Apache - para el lenguaje "Java".
- Frontier::RPC - para el lenguaje "Perl" .
- XMLRPC-PHP - para el lenguaje "PHP".
- XMLRPC-Python - para el lenguaje "Python".
- XMLRPC-C - para los lenguajes "C" y "C++".
- XMLRPC-ASP - para el lenguaje "COM/VBasic".

XML-RPC que fue el primer protocolo de comunicación bajo HTTP mediante XML. Con este protocolo se pueden realizar llamadas a procedimientos remotos, es decir, se puede bien en un cliente o un servidor realizar peticiones mediante HTTP a un servidor web. Los mensajes deben tener un formato determinado empleando XML para encapsular los parámetros de la petición. Con el paso del tiempo el proyecto iniciado por David Winer interesó a importantes multinacionales entre las que se encuentran IBM y Microsoft y de este interés por XML-RPC se desarrolló SOAP. [13]

SOAP (Simple Object Access Protocol)

Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP no define en sí mismo ninguna semántica de la aplicación como un modelo de programación o la semántica de aplicación específica, sino que define un mecanismo simple para expresar la semántica de aplicaciones, proporcionando un modelo de paquetes modulares y la codificación de los mecanismos de codificación de datos dentro de los módulos. Una diferencia fundamental es que en los procedimientos en SOAP los parámetros tienen nombre y no interesa su orden, no siendo así en XML-RPC. [14]

Ventajas de SOAP:

- Protocolo abierto: SOAP es una especificación abierta, construido sobre tecnologías también abiertas como XML y HTTP. Por estas razones ha sido aceptado uniformemente por la industria, lo cual incrementa sus posibilidades de transformarse en estándar.
- La especificación de SOAP está bien definida y es sumamente simple.
- Es independiente de plataformas y lenguajes.
- No se encuentra fuertemente asociado a ningún protocolo de transporte: La especificación de SOAP no describe cómo se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- No está atado a ninguna infraestructura de objeto distribuido.
- SOAP presenta un bajo acoplamiento con los sistemas y las aplicaciones internas de una

organización, situación que no se da con otras tecnologías distribuidas (CORBA, DCOM, etc.) donde existe un alto acoplamiento con la arquitectura subyacente de las aplicaciones, lo que implica que tanto el emisor como el receptor deben usar el mismo sistema o por lo menos sistemas compatibles.

- Aprovecha los estándares existentes en la industria, ejemplo, aprovecha XML para la codificación de los mensajes y los protocolos de transporte.

Desventajas de SOAP:

- Los mensajes SOAP, por estar en formato XML, son mucho más grandes que sus equivalentes en las tecnologías CORBA y DCOM con formatos NDR o CDR respectivamente.
- Como está basado en XML (formato ASCII) el parseador (parser) requiere más recursos de CPU, que otros protocolos basados en formato binario.
- Es más complejo que los demás de su tipo en lo referente a su programación y uso.
- No define la semántica de los mensajes, lo cual significa que la aplicación cliente y la aplicación servidor deben acordar la semántica de los mensajes.

D-Bus (*Desktop Bus*)

D-Bus es un software libre y de código abierto, especializado en el mecanismo de comunicación entre procesos (IPC) ampliamente utilizado hoy día. Es parte del proyecto freedesktop.org y se utiliza en amplio rango de aplicaciones. Se divide fundamentalmente en tres capas:

- Una biblioteca, libdbus, que permite a dos aplicaciones conectarse e intercambiar mensajes.
- Un demonio ejecutable que funciona como bus de mensajes, construido sobre libdbus, al cual pueden conectarse a varias aplicaciones. El demonio puede encaminar mensajes desde una aplicación a una o más aplicaciones.
- Bibliotecas adaptadas (wrappers) para su uso en marcos de trabajo (frameworks) concretos.

Principales usos de D-Bus:

- Comunicación entre aplicaciones de escritorio en la misma sesión, facilitando la integración de aplicaciones dentro de un mismo entorno de escritorio y el tratamiento de asuntos relativos al ciclo de vida de los procesos.
- Comunicación entre el sistema operativo y la sesión de escritorio, incluyendo dentro del sistema operativo al núcleo y algunos demonios o procesos. [15]

Desventaja de D-Bus:

- Solo pueden ser utilizadas en la comunicación entre procesos en un mismo ordenador.

FTP (File Transfer Protocol / Protocolo de Transferencia de Archivos)

La implementación del FTP se remonta a 1971, cuando se desarrolló un sistema de transferencia de

archivos (descrito en RFC141) entre equipos del Instituto Tecnológico de Massachusetts. Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor, pues desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

FTP define la manera en que los datos deben ser transferidos a través de una red TCP/IP y este permite:

- La conexión a un sistema remoto.
- Observar los directorios remotos.
- Cambiar de directorio remoto.
- Copiar uno o varios archivos hacia el directorio local.
- Copiar uno o varios archivos hacia el directorio remoto.

El protocolo FTP emplea los permisos de ejecución, lectura y escritura debido a que se desarrolló en entornos de tipo UNIX similares a los populares GNU/Linux. Es el más conocido y utilizado para la transferencia de archivos en Internet pues es el ideal para transferir grandes bloques de datos por la red. [16]

TCP/IP (Transfer Control Protocol / Internet Protocol // Protocolo de Control de Transmisión/Protocolo de Internet)

TCP/IP fue desarrollado y demostrado por primera vez en 1972 por el Departamento de Defensa de los Estados Unidos, ejecutándolo en ARPANET, una red de área extensa de dicho departamento. Son los protocolos que se utilizan en Internet para transmitir datos, el TCP está orientado a la conexión que establece una línea de diálogo entre el emisor y el receptor antes de que se transfieran los datos, y el IP trata cada paquete de forma independiente e incluye en la cabecera información adicional para así controlar la información. Estos protocolos garantizan que la comunicación entre dos aplicaciones sea precisa. [17]

Se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que lo componen, los cuales fueron los primeros en definirse y son los más utilizados. Pero existen más de 100 protocolos diferentes en este conjunto.

Ventajas de TCP/IP:

- Consume pocos recursos de red.
- Trabaja sobre una gran variedad de hardware y sistemas operativos.
- Está diseñado para enrutar y tiene un grado muy elevado de fiabilidad.
- Es adecuado para redes grandes y medianas, así como en redes empresariales.
- Es compatible con las herramientas estándar para analizar el funcionamiento de la red.
- Ofrece mayor fiabilidad pues envía información adicional en el paquete para asegurar la

conexión permitiendo que los datos vayan correctamente del emisor al receptor, en el orden estipulado, y completos.

- Debe ser utilizado en aplicaciones donde es más importante que los datos lleguen correctamente a que lleguen rápidamente.

Desventajas de TCP/IP:

- Es más difícil de configurar y de mantener.
- Al enviar información adicional en el paquete para asegurar la conexión, como el tamaño del paquete es limitado, pierde espacio útil para esta función.
- Es más lento que UDP.

UDP (User Datagram Protocol / Protocolo de datagrama de usuario)

Es un protocolo del nivel de transporte no orientado a conexión que permite el envío de datagramas a través de la red, sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. No tiene control de flujo, por lo que los paquetes pueden adelantarse unos a otros y tampoco se sabe si el mensaje ha llegado correctamente, ya que no hay confirmación de entrega o recepción. [18]

Ventaja de UDP:

- Es más rápido que TCP.

Desventajas de UDP:

- No es fiable pues los datos son enviados sin saber si van a ser recibidos correctamente, en orden o completos. En otras palabras, no garantiza el envío correcto de los datos.
- Solo debe ser utilizado en aplicaciones donde es más importante la recepción rápida de los datos que la verificación de los mismos.

HTTP (Hypertext Transfer Protocol / Protocolo de transferencia de hipertexto)

Es el protocolo más utilizado en Internet. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force. El propósito de este protocolo es permitir la transferencia de archivos entre un navegador (el cliente) y un servidor web localizado, mediante una cadena de caracteres denominada dirección URL.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usa la información que un servidor puede almacenar en el sistema cliente (cookies). [19]

S-HTTP (Secure HTTP)

Fue posiblemente el primer protocolo de seguridad implementado para Internet. Se dio a conocer por primera vez en forma de borrador en junio de 1994. S-HTTP es una extensión del protocolo HTTP cuya finalidad es la transmisión de datos de forma segura sobre la web entre un cliente y un servidor. Para ello, trabaja sobre la capa de aplicación, cifrando el contenido de los mensajes entre las dos máquinas, usando para ello un sistema de cifrado basado en una pareja de claves pública y privada.

S-HTTP permite la negociación inicial de la fortaleza del cifrado a usar (algoritmo y longitud de claves), permite la autenticación de ambos extremos mediante el uso de firmas digitales y verifica la integridad de los datos usando las MAC (en español, código de autenticación de mensajes). Es un protocolo muy flexible que permite usar estas opciones (firma y cifrado) de forma opcional.

SSL (Secure Socket Layer / Protocolo de conexión segura)

Es un protocolo criptográfico que proporciona comunicaciones seguras por una red. Fue desarrollado por Netscape Communications Corporation, la cual en noviembre de 1995 publicó la especificación para SSL 3.0, que desde entonces se ha convertido en el estándar para comunicaciones seguras entre clientes y servidores en Internet.

El objetivo de Netscape era crear un canal de comunicaciones seguro entre un cliente y un servidor que fuese independiente del sistema operativo usado por ambos y que se beneficiara de forma dinámica y flexible de los nuevos adelantos en materia de cifrado a medida que estos estuvieran disponibles. SSL fue diseñado como un protocolo seguro de propósito general y no teniendo en mente las necesidades específicas del comercio electrónico. SSL trabaja sobre el protocolo TCP y por debajo de protocolos como HTTP, IMAP, LDAP, etc., y puede ser usado por todos ellos de forma transparente para el usuario.

El uso de SSL no se limita solo a cifrar datos importantes que se envían y se reciben a través de la red, también permite autentificar tanto a los clientes como a los servidores que realizan conexiones TCP/IP.

SSL ofrece algunas características de interés:

- Separación de responsabilidades: Utiliza algoritmos independientes para la encriptación, autenticación e integridad de datos, con claves diferentes (claves secretas) para cada función.
- Eficiencia: Aunque la fase de saludo utiliza algoritmos de clave pública, la operativa de intercambio de datos se realiza mediante encriptación y desencriptación de clave privada. Los algoritmos de clave privada son más rápidos. Además, la fase de saludo no tiene que repetirse para cada comunicación entre un cliente y un servidor, la “clave secreta” negociada puede conservarse entre conexiones SSL. Esto permite que las nuevas conexiones SSL inicien la comunicación segura de inmediato, sin necesidad de realizar lentas operaciones de clave pública.
- Autenticación con base en certificados: Se utilizan certificados X.509 para la autenticación. Los

certificados de servidores son obligatorios, mientras que los del cliente son opcionales.

- Independiente de protocolos: Aunque SSL se diseñó para funcionar sobre TCP/IP, puede hacerlo sobre cualquier protocolo confiable orientado a conexiones. En cambio, no funcionará sobre un protocolo no confiable, como por ejemplo UDP.
- Protección contra ataques: SSL protege contra ataques de hombre en el camino o de reproducción. En un ataque de hombre en el camino, el atacante intercepta todas las comunicaciones entre las dos partes, haciendo a cada una de ellas creer que se comunica con la otra. SSL protege contra estos ataques mediante certificados digitales que nos garantizan que cada parte es quien dice ser.

Ventajas de SSL:

- Confidencialidad: Mediante el uso de la encriptación se garantiza que los datos enviados y recibidos no pueden ser interpretados por ninguna otra persona que no sea ni el emisor ni el receptor.
- Integridad: Se garantiza que los datos recibidos son exactamente iguales a los datos enviados.
- Autentificación: El emisor se autentifica utilizando un certificado digital emitido por una empresa llamada Autoridad Certificadora. Este documento es totalmente infalsificable y garantiza que el receptor es quien dice ser.

Desventaja de SSL:

- SSL solamente asegura que mientras los datos viajan desde el cliente hasta el servidor no serán modificados ni espiados. Lo que el servidor haga con ellos, está ya más allá de la competencia de este protocolo. Los datos podrían ser manipulados irresponsablemente o caer en manos de un atacante que asalte el servidor.

SSH (Secure Shell / intérprete de órdenes segura)

Es un protocolo para crear conexiones seguras entre dos sistemas usando una arquitectura cliente-servidor. SSH es tanto una aplicación como un protocolo, que permite conectar dos ordenadores a través de una red, ejecutar comandos de manera remota y mover ficheros entre los mismos. Proporciona autentificación fuerte y comunicaciones seguras sobre canales no seguros y pretende ser un reemplazo seguro para aplicaciones tradicionales no seguras, como telnet, rlogin, rsh y rcp. Su versión 2 (SSH2) proporciona también sftp, un reemplazo seguro para FTP.

SSH permite establecer conexiones seguras a servidores X-Windows, realizar conexiones TCP seguras y realizar acciones como sincronización de sistemas de ficheros (rsync) y copias de seguridad por la red, de manera segura. Además, permite solventar problemas de seguridad que pueden derivarse del hecho de que usuarios tengan acceso como superusuario (root) a ordenadores de la red o acceso al cable de comunicaciones, de modo que podrían interceptar contraseñas que se transmiten

por la red. Esto no puede ocurrir con SSH, ya que nunca envía texto plano, sino que la información siempre viaja cifrada.

El protocolo SSH proporciona los siguientes tipos de protección:

- Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- El cliente transmite su información de autenticación al servidor usando una encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos durante la conexión se transfieren por medio de encriptación de 128 bits, lo cual los hacen extremamente difícil de descifrar y leer.
- El cliente tiene la posibilidad de enviar X11 aplicaciones lanzadas desde el intérprete de comandos de la shell. Esta técnica proporciona una interfaz gráfica segura (llamada reenvío por X11) y un medio seguro para usar aplicaciones gráficas sobre una red.

Ya que el protocolo SSH encripta todo lo que envía y recibe, se puede usar para asegurar protocolos inseguros. El servidor puede convertirse en un conducto para convertir en seguros los protocolos inseguros mediante el uso de una técnica llamada reenvío por puerto, como por ejemplo POP, incrementando la seguridad del sistema en general y de los datos.

Análisis de la selección de los protocolos

Después de realizado el estudio de los mecanismos y protocolos de comunicación entre procesos y ordenadores existentes, se decidió cuáles se van a utilizar en la comunicación de Xerberos, teniendo en cuenta las características y ventajas y desventajas que brindan cada uno de ellos.
Se utilizarán los sockets para la comunicación entre los procesos situados en computadoras distintas. Se decidió el uso de socket porque estos proporcionan una comunicación de dos vías entre dos ordenadores, permiten implementar una arquitectura cliente-servidor (arquitectura propuesta para el sistema) y tienen gran disponibilidad, pues la interfaz Socket está disponible en múltiples sistemas operativos y lenguajes de programación. Además, el uso de los sockets permite mantener la compatibilidad con lo que ya está implementado del sistema. Entre los tipos de sockets estudiados se determinó utilizar el socket de flujo pues es el más fiable y trabaja sobre el protocolo de transporte TCP/IP.
También se utilizará SSL para proporcionar una comunicación segura por la red. SSL se diseñó para funcionar sobre TCP/IP, que es el protocolo de transporte que se empleará para el trabajo con los sockets. SSL garantiza la integridad de los datos, brinda un canal de comunicaciones seguro y permite autenticar a las partes que intervienen en la comunicación, lo cual es muy importante para el cumplimiento de los objetivos del sistema.
El protocolo SSH anteriormente explicado, tiene características que lo hacen muy compatible con los objetivos de la comunicación en Xerberos, pues permite conectar dos ordenadores a través de una

red, ejecutar comandos en ordenadores remotos y mover ficheros entre los mismos, todo esto de forma segura. La utilización del mismo se tuvo en cuenta pero se decide no usarlo debido a que:

- El proceso de comunicar dos puntos o más en una subred interna se hace un poco complejo usando SSH ya que se debe configurar cada cliente de forma manual para que acepte comandos y scripts del servidor y el objetivo principal es lograr interactuar lo menos posible con cada cliente manualmente.
- El sistema necesita llevar un control de cada cliente conectado al servidor para gestionarlos, mostrarlos en la interfaz gráfica y permitirle al usuario una forma fácil de interactuar con cada cliente de manera individual. Empleando SSH se hace un tanto difícil integrarlo a una interfaz amigable al usuario final.
- El envío de comandos y scripts a una o dos computadoras de una subred con SSH es relativamente rápido, no siendo así cuando se tienen 30 o más clientes conectados al servidor.

Para argumentar lo anteriormente expresado, se efectuó un estudio de cómo se realiza la comunicación en sistemas de gestión de ordenadores existentes en el mercado, el cual no fue del todo satisfactorio, pues la mayoría de los fabricantes de estos sistemas no revelan su tesoro más preciado que es la comunicación entre los nodos. Sin embargo, del software Capristano, que es una herramienta de gestión de tareas automatizadas, sí se divulga que utiliza SSH para ejecutar comandos en máquinas remotas. Precisamente su desventaja más notable consiste en que el nivel de escalabilidad de este sistema se comporta bien en entornos pequeño-medios, pero cuando el volumen de los ordenadores crece, se hace demasiado difícil su mantenimiento. [20]

1.4. Herramientas, lenguajes y metodología a utilizar

Visual Paradigm

Visual Paradigm es un software de modelado y una herramienta de diseño UML diseñada para ayudar en el desarrollo de software. Utiliza el lenguaje de modelado estándar UML y ha sido adoptada por muchos proyectos de código abierto y grupos comunitarios. 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. Ayuda de forma rápida a la construcción de aplicaciones de calidad, mejor y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. [21]

Entre sus principales características se encuentran:

- Entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.

- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs (Entornos de Desarrollo Integrados).
- Disponibilidad en múltiples plataformas.

Visual Paradigm puede generar código hasta en 15 lenguajes de programación, entre ellos C++ el cual será utilizado en la implementación del mecanismo de comunicación. Es la herramienta CASE que se utiliza en el modelado de los diagramas.

RapidSVN

Es una herramienta para el control de versiones de un proyecto o aplicación. Proporciona una interfaz fácil de usar. Es sencillo para principiantes, pero suficientemente flexible como para aumentar la productividad de los usuarios experimentados de subversión, ocasionando que sea eficiente. Es portable y multiplataforma. Es posible emplearlo en cualquier plataforma en la que se puedan ejecutar wxWidgets xlix, por ejemplo en: Linux, Windows, Mac OS/X, Solaris, etc. Es rápido, está completamente escrito en C++ y es multilingüe (ha sido traducido a muchos idiomas: alemán, francés, italiano, portugués, ruso, ucraniano, chino simplificado, japonés). Cuenta con soporte completo para el estándar de codificación Unicode. [22]

Framework LibPoco

Es una colección de marcos de trabajo (frameworks) y bibliotecas de código abierto para C++ que facilitan la construcción de aplicaciones centradas en la red, además simplifican y aceleran el desarrollo de aplicaciones en C++. Permite el desarrollo tanto de sistemas servidores como aplicaciones de escritorio. La libpoco posee un diseño e implementación eficiente, que lo hacen un marco de trabajo potente y robusto para utilizarlo en aplicaciones que posean un gran número de funcionalidades.

Este framework posee características que facilitan el trabajo de los programadores como por ejemplo, permite el trabajo e integración con varios gestores de base de datos, permite el trabajo con hilos, gestiona la encriptación y desencriptación de datos, además facilita en gran medida el trabajo con XML, entre otras. [23]. En la siguiente figura se muestra la diversidad de paquetes que contiene la Libpoco, en los que están distribuidas sus principales funcionalidades.

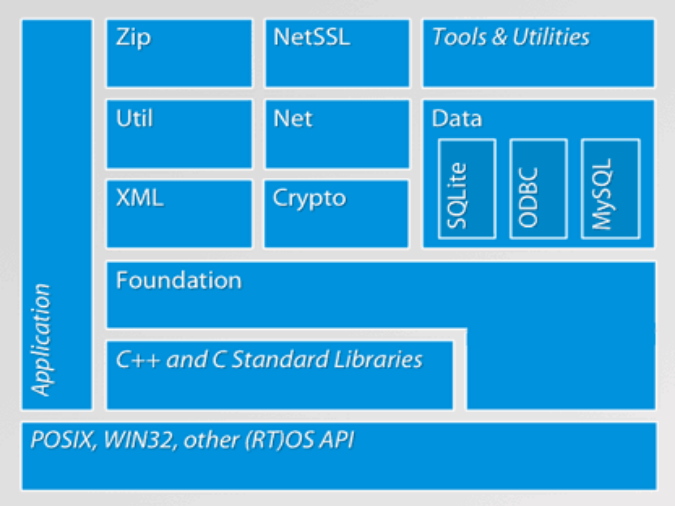


Figura 1.2: Estructura de los paquetes de la libpoco.

Framework SistClon

El framework SistClon es un conjunto de clases desarrolladas en C++ e implementadas por el Ing. Dayron Pérez Roldán apoyado en el equipo de trabajo del proyecto de desarrollo Nova-Xerberos. Es muy sencillo y está basado en el framework Libpoco. Su principal objetivo es establecer un marco de trabajo común y fácil para todos los desarrolladores del sistema. Además, facilita y permite el trabajo con XML y el uso de funciones ya implementadas para la conversión de tipos. Recopila aquellas clases y métodos que pueden resultar de utilidad en la implementación de nuevos servicios y funcionalidades del sistema Xerberos haciendo de la programación un proceso más fácil, rápido y menos sobrecargado. [24]

Lenguaje de programación C/C++

El lenguaje de programación C fue creado en 1969 por Ken Thompson. No es un lenguaje fuertemente tipado, permitiendo casi cualquier conversión de tipo, bastando solamente con que estos sean parecidos y no necesariamente iguales. Constituye un lenguaje estructurado de nivel medio aunque posee muchas características de bajo nivel, permitiéndole una mayor flexibilidad y potencia a cambio de menor abstracción, aunque posee mucho.

Dispone de una biblioteca estándar que contiene numerosas funciones y que siempre está disponible, además de las extensiones que puedan proporcionar los compiladores o entornos de desarrollo. Es un lenguaje muy eficiente debido a que es posible la realización de implementaciones óptimas. Como la mayor cantidad de código en los sistemas UNIX está escrito en C, la utilización de este lenguaje en el desarrollo del mecanismo de comunicación facilitará el uso de numerosas librerías del sistema operativo que son útiles para el logro de los objetivos planteados.

El lenguaje de programación C++ constituye un superconjunto del lenguaje C y fue desarrollado en 1980, por Bjarne Stroustrup. Su característica fundamental es el soporte para la programación orientada a objetos y de plantillas, o programación genérica, así como la programación estructurada.

Su versatilidad y portabilidad permite la realización desde aplicaciones muy simples hasta muy complejas, compilando las mismas en cualquier sistema sin cambio alguno en el código. C++ posibilita la redefinición de operadores, más conocida como sobrecarga de operadores, también nos permite la identificación de tipos en tiempo de ejecución. [25]

C++ es el lenguaje de programación que se utilizará en la implementación de los subsistemas de Xerberos.

Metodología de desarrollo

La metodología de desarrollo de software son los procedimientos, las técnicas y las herramientas utilizadas para el desarrollo de un sistema. Provee al programador de una guía para desarrollar una aplicación pues enseña a un equipo de trabajo a dividir un proyecto en etapas, muestra qué tareas se llevan a cabo en cada etapa, qué restricciones deben aplicarse, qué técnicas y herramientas se emplean y cómo se controla y gestiona un proyecto. Las metodologías se han dividido en dos grandes grupos: Ágiles/Ligeras y Pesadas/Tradicionales.

Metodologías Pesadas: Se centran en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar y requiere una extensa documentación, ya que pretende prever todo de antemano. Son más eficaces y necesarias cuanto mayor es el proyecto que se pretende realizar, respecto a tiempo y recursos que son necesarios emplear y donde se requiere una gran organización.

Ejemplos:

- METRICA
- OPEN
- RUP (Rational Unified Process).

Metodologías Ágiles: Ayudan a entregar un producto de calidad en tiempo y costo estimados. Combina una filosofía y un conjunto de directrices de desarrollo. La filosofía busca la satisfacción del cliente y la entrega temprana de software incremental; equipos de proyectos pequeños y con alta motivación; métodos informales; un mínimo de productos de trabajo de ingeniería de software y una simplicidad general del desarrollo. Las directrices del desarrollo resaltan la entrega sobre el análisis y diseño (aunque estas actividades no se descartan) y la comunicación activa y continua entre los desarrolladores y los clientes. Las metodologías ágiles representan una opción razonable a las metodologías pesadas para ciertas clases de software y ciertos tipos de proyectos de software. [26]

Sus elementos claves son:

- Poca documentación.
- Simplicidad.
- Análisis como una actividad constante.
- Diseño evolutivo.
- Integraciones.

- Testeos diarios.

Ejemplos:

- Feature-Driven Development (FDD)
- Crystal
- XP (Extreme Programming)
- SCRUM
- SXP

Metodología a utilizar:

Para el desarrollo de Xerberos se definió el uso de las metodologías ágiles, por todas las ventajas que estas brindan a un equipo de desarrollo pequeño, expuestas anteriormente. Por esta razón, el desarrollo individual de todos los subsistemas, aplicaciones y servicios que lo conforman, debe realizarse respetando la metodología definida para el sistema. Entre las diversas metodologías ágiles que existen, el equipo de desarrollo decidió utilizar SXP porque es la metodología que se adopta en el centro de desarrollo y además tiene características que guían correctamente al equipo de proyecto en la obtención de un producto final de calidad.

SXP: Es una metodología compuesta por dos metodologías, Scrum y XP. Es un híbrido cubano de metodologías ágiles, que 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. 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 al éxito del proyecto.

SXP está 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. [27]

Consta de cuatro fases principales:

1. **Planificación-Definición:** En esta fase lo fundamental es la realización de entrevistas con el cliente, el levantamiento de los requisitos, así como todos los artefactos de metodología ágil que estos generan. Es donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto. Además, en esta fase se definen las Historias de Usuario y las Tareas de Ingeniería que se realizarán para la implementación de estas y se realiza el modelado de diseño del sistema.
2. **Desarrollo:** Se definen las historias de usuarios que se implementarán. También se realiza la programación propia de los requisitos funcionales de acuerdo con lo expresado en los artefactos generados, cumpliendo con los estándares de programación que se hayan definido

para el desarrollo del software. La esencia de esta fase es la implementación del sistema hasta que esté listo para entregarlo.

3. **Entrega:** Se procede a la entrega de la documentación y del producto elaborado.
4. **Mantenimiento:** Se brinda soporte al sistema entregado y se realiza la gestión de cambios.

CAPÍTULO 2. CARACTERÍSTICAS DEL MECANISMO DE COMUNICACIÓN DE XERBEROS

El presente capítulo expone las características del sistema y se plantean el Modelo de Dominio y los requisitos funcionales y no funcionales del mecanismo de comunicación. Además, se realiza la definición de las Historias de Usuario y las Tareas de Ingeniería.

2.1. Descripción de la solución propuesta

En Xerberos, la definición y diseño de un mecanismo de comunicación es el primer paso para que el sistema funcione como un todo y exista una adecuada gestión de los servicios. Este mecanismo se define como: el conjunto de componentes de software que intervienen en la comunicación y son comunes a todos los elementos interconectables de Xerberos.

Los componentes de software comunes a todos los elementos interconectables son: el emisor, el receptor y el despachador (dispatcher), que tienen que existir en todos los nodos de Xerberos para establecer una comunicación. De esta forma, el emisor de un ordenador envía información mediante un canal de comunicación, la cual es recibida por el receptor de otro ordenador, manipulada por el despachador y en caso de requerir una respuesta, esta es enviada por el emisor de la misma forma. La figura 2.1 muestra el mecanismo descrito:

Figura 2.1: Descripción del Mecanismo de Comunicación de Xerberos

El sistema está compuesto por varios nodos, entre ellos, los que interactúan con el mecanismo de comunicación son: la estación de administración (usuario), la estación cliente y el servidor central. *(Ver anexo 2)*. Por esta razón, el diseño que se propone debe contemplar que se realizarán operaciones

distintas en cada nodo y debe garantizar la uniformidad en la recepción y envío de los datagramas generados por dichas operaciones. Para establecer la conectividad y el intercambio de información se propone el empleo sockets con el protocolo TCP/IP y se proporciona una comunicación segura por la red utilizando SSL.

Se espera como resultado, el diseño de un mecanismo de comunicación que proporcionará un marco de trabajo (framework) para la implementación del subsistema de comunicación de Xerberos en cada uno de los nodos.

2.2. Modelo de Dominio

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los principales conceptos u objetos del mundo real, significativos para un problema o área de interés. Este es de gran ayuda para desarrolladores y usuarios, ya que de esta forma se utiliza un vocabulario común y pueden entender el contexto en que se enmarca el sistema. [28]

Se llama "de dominio" para distinguirlo del modelo de negocio que es un concepto más amplio. En la definición del mecanismo de comunicación no se realiza el modelo del negocio, pues, debido a la relativa simplicidad del entorno donde está enmarcado el sistema y al conocimiento que se posee acerca de su funcionamiento, no es necesario modelarlo para comprender la problemática que ha de resolverse, siendo suficiente realizar un modelo de dominio o conceptual.

La figura siguiente muestra el Modelo de Dominio del mecanismo de comunicación de Xerberos:

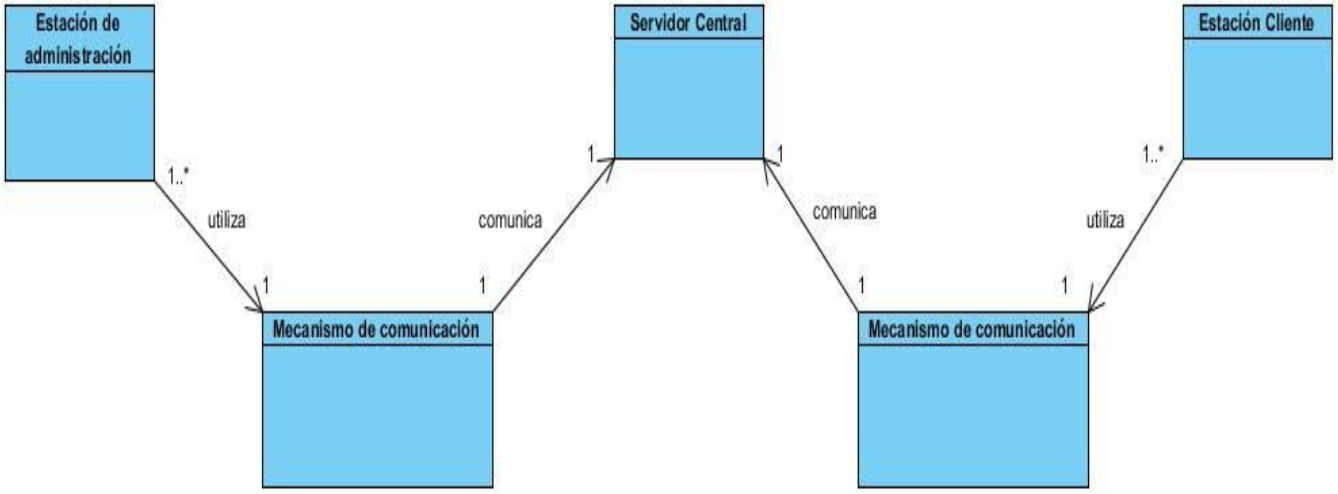


Figura 2.2: Modelo de Dominio del Mecanismo de Comunicación de Xerberos.

Fundamentación del Modelo de Dominio

Estación de Administración: Son las estaciones desde las cuales el técnico brinda soporte y mantenimiento a las estaciones clientes, comúnmente llamadas usuarios.

Servidor Central: Es el servidor principal de Xerberos por el que pasa toda la comunicación del sistema. En él ocurren los procesos centrales del software.

Estación cliente: Son las estaciones a las cuales se les puede brindar soporte y mantenimiento de forma remota.

Mecanismo de comunicación: Es la propuesta que se realiza.

Para garantizar la comunicación en Xerberos, en cada uno de los nodos anteriormente descritos se implementará un subsistema de comunicación basado en el marco de trabajo (mecanismo de comunicación que se propone) y aterrizado a las características de cada uno. El marco de trabajo solo tiene en cuenta lo que es común e imprescindible para garantizar la comunicación en cualquiera de los nodos.

2.3. Requisitos Funcionales

Los requerimientos funcionales son la determinación exacta de qué debe ser capaz de hacer el sistema, estas se corresponden con opciones que ejecutará el software, operaciones realizadas de forma oculta o condiciones extremas a determinar por el sistema. La metodología de desarrollo ágil SXP establece que en la fase de definición han de identificarse los requisitos claves del software. [29]

Lista de reserva del Producto

Los requisitos que se capturan son priorizados y expresados en una Lista de Reserva del Producto (LRP) la cual define el trabajo que se va a realizar en el proyecto. El objetivo es asegurar que el producto definido al terminar la lista es el más correcto, útil y competitivo posible.

Prioridad	Ítem	Descripción	Estimación (Semanas)	Estimado por
Muy Alta				
	1	Enviar datos a un ordenador.	2	Indira Garcés Pérez
	2	Recibir datos de un ordenador.	2	
	3	Ejecutar plugins.	4	
	4	Proporcionar una interfaz para establecer la conexión entre dos ordenadores.	4	
	5	Proporcionar una interfaz para la recepción de datos desde un	4	

		ordenador.		
	6	Proporcionar una interfaz para el envío de datos a un ordenador.	4	
Alta				
	7	Verificar estado de la conexión.	2	
	8	Registrar eventos del sistema.	3	
RNF (Requisitos No Funcionales)				
	1	Establecer conexiones seguras.	6	

2.4. Historias de usuario y Tareas de Ingeniería

Las Historias de Usuario (HU) son la técnica utilizada en SXP para especificar los requisitos del software, lo que equivaldría a los casos de uso en el proceso unificado. Las HU guían la construcción de las pruebas de aceptación y son utilizadas para estimar tiempos de desarrollo. En este sentido, solo proveen detalles suficientes para hacer una estimación razonable del tiempo que llevará implementarlas. En el momento de implementar una historia de usuario, se debe detallar a través de la comunicación con el cliente, siendo estas la base para las pruebas funcionales. [30]

Las Tareas de Ingeniería definen cada una de las actividades asociadas a las HU y permiten organizar el proceso de implementación, así como conocer el grado de complejidad de cada HU, teniendo en cuenta la cantidad de tareas asociadas.

A continuación se detallan las Historias de Usuario y las Tareas de Ingeniería del mecanismo de comunicación de Xerberos:

Historia de Usuario	
Número: 1	**Nombre Historia de Usuario:** Establecer conexión.
Modificación de Historia de Usuario Número: Ninguna	
Usuario:	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 2 semanas
Descripción: Permitirá establecer conexión entre dos ordenadores.	
Observaciones: Para establecer una comunicación entre dos ordenadores, uno se comporta como maestro, que inicia la conexión, y otro como esclavo, que la acepta. En Xerberos, las estaciones de administración y clientes se comportan como ordenadores maestros y el servidor central se comporta como esclavo.	

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Estudio para trabajo con sockets.	

Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Los sockets son el mecanismo de comunicación que se utilizará para establecer la comunicación entre dos ordenadores. Conocer a profundidad sus características y funcionamiento es imprescindible para la implementación de las clases encargadas de establecer las conexiones. Además, se estudia la implementación de los sockets en el framework Libpoco, con el objetivo de utilizar estas clases para que el desarrollo del sistema sea más rápido y fácil.	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 1
Nombre Tarea: Implementar la interfaz para establecer conexión entre dos ordenadores.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se desarrolla una interfaz genérica llamada *Conexion* que crea un objeto socket el cual permitirá establecer la comunicación por TCP/IP entre dos ordenadores. Existen dos formas de establecer la comunicación: Conectar, que es la que utiliza un ordenador maestro y Aceptar, que es la que utiliza un ordenador esclavo. Esta clase gestiona todas las operaciones que se realizan con el socket y que son comunes a los dos tipos de conexión.	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** 1
Nombre Tarea: Implementar la clase Conectar.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se desarrolla la clase *Conectar*, la cual hereda de la clase *Conexion* y gestiona todas las operaciones a realizarse con el socket en los ordenadores maestros, que se encargan de establecer la conexión.	

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:** 1
Nombre Tarea: Implementar la clase Aceptar.	

Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se desarrolla la clase *Aceptar*, la cual hereda de la clase *Conexion* y gestiona todas las operaciones a realizarse con el socket en los ordenadores esclavos, que se encargan de aceptar la conexión.	

Historia de Usuario	
Número: 2	**Nombre Historia de Usuario:** Enviar datos a un ordenador.
Modificación de Historia de Usuario Número: Ninguna	
Usuario:	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo:	**Puntos Reales:** 1 semana
Descripción: Permitirá enviar información hacia un ordenador.	
Observaciones: El objetivo de establecer una comunicación entre dos ordenadores es realizar el intercambio de información. Un ordenador envía información y otro la recibe.	

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 2
Nombre Tarea: Implementar la interfaz para enviar datos.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se implementa la clase *Emisor* que se encarga de enviar todos los datos que se encuentran disponibles a un ordenador.	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 2
Nombre Tarea: Crear datagrama de la información que se enviará.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se implementa la clase *ConstruirDatagrama* que se encarga de procesar los datagramas que se van a enviar hacia un ordenador, o sea, si la información es muy grande, se deben construir varios datagramas de 256 bytes con la información.	

Historia de Usuario	
Número: 3	**Nombre Historia de Usuario:** Recibir datos de un ordenador.
Modificación de Historia de Usuario Número: Ninguna	
Usuario:	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo:	**Puntos Reales:** 1 semana
Descripción: Permitirá recibir información desde un ordenador.	
Observaciones: El objetivo de establecer una comunicación entre dos ordenadores es realizar el intercambio de información. Un ordenador envía información y otro la recibe.	

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 3
Nombre Tarea: Implementar la interfaz para recibir datos.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se implementa la clase *Receptor* que se encarga de recibir todos los datos que se envían desde otro ordenador.	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 3
Nombre Tarea: Procesar datagrama recibido.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se implementa la clase *ConstruirData* que se encarga de procesar los datagramas que recibe un ordenador, o sea, si se reciben uno o varios datagramas para una misma información, esta clase se encarga de construir la información.	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** 3
Nombre Tarea: Implementar la clase Despachador.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Esta clase se encarga de manejar las peticiones, o sea, distribuir las	

tareas para que sean ejecutadas por el plugin que le corresponde.

Historia de Usuario	
Número: 4	**Nombre Historia de Usuario:** Registrar eventos.
Modificación de Historia de Usuario Número: Ninguna	
Usuario:	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo:	**Puntos Reales:** 2 semanas
Descripción: Permitirá guardar un registro de los eventos del sistema.	
Observaciones: Uno de los requerimientos del sistema es guardar un registro de las acciones más importantes que se realizan.	

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 4
Nombre Tarea: Estudiar patrón de diseño Observador.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: El patrón de diseño Observador es un patrón de comportamiento, el cual define una dependencia de uno a muchos entre objetos, de forma que cuando el objeto observado cambia de estado, se notifican y actualizan automáticamente todos los objetos que están observándolo.	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 4
Nombre Tarea: Estudiar implementación del patrón Observador en la Libpoco.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se estudia cómo se implementa el patrón Observador en la Libpoco con el objetivo de conocer si es oportuno utilizar dicha implementación en el desarrollo del mecanismo de comunicación de Xerberos.	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** 4
Nombre Tarea: Estudiar la clase LogMsg del Framework SistClon.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**

Programador Responsable:
Descripción: Se estudia la clase que permite manejar registros para conocer si es oportuno utilizarla para el registrar los eventos del sistema.

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:** 4
Nombre Tarea: Implementación del patrón Observador.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0,1
Fecha Inicio:	**Fecha Fin:**
Programador Responsable:	
Descripción: Se desarrollan las clases que se comportan como observadoras.	

CAPÍTULO 3. DISEÑO DEL MECANISMO DE COMUNICACIÓN DE XERBEROS

En el presente capítulo se realiza el diseño del mecanismo de comunicación de Xerberos, utilizando la metodología de desarrollo ágil SXP. Además, se describe la arquitectura a utilizar y los patrones arquitectónicos y de diseño, se muestra el diagrama de clases del diseño y se realiza la descripción detallada de cada una de las clases que componen el mencionado diagrama.

3.1. Descripción de la arquitectura

En el diseño se modela el sistema y se establece su estructura (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. En la metodología SXP no se enfatiza en la definición temprana de una arquitectura estable para el sistema, sino que esta se asume de forma evolutiva. Para solventar los inconvenientes que pudiera generar el no contar con dicha arquitectura, se utiliza el diseño con metáforas que da como resultado el modelo de diseño. [30]

La arquitectura de software es la vista conceptual de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos. La descripción de la arquitectura es importante y a la vez necesaria para el desarrollo de un software pues facilita la comunicación entre todas las partes (partícipes) interesadas en el desarrollo de un sistema basado en computadora. [31]

El Patrón Arquitectónico define la estructura básica de un sistema, la cual tiene que ver con la especialización que adoptan los objetos y clases de acuerdo con el tipo de aplicación o entorno tecnológico. Representa una plantilla de construcción que provee un conjunto de subsistemas aportando las normas para su organización. [32]

El diseño del mecanismo de comunicación debe acoplarse a la arquitectura de Xerberos, la cual utiliza el patrón cliente-servidor. Este patrón arquitectónico define una relación entre dos aplicaciones, en las cuales una de ellas (cliente) envía peticiones a la otra (servidor), que le da respuestas.

3.2. Diagrama de Clases del Diseño

El diagrama de clases del diseño representa de forma gráfica las clases que serán implementadas en el sistema. En él se definen sus atributos, funcionalidades y las relaciones de composición, agregación y asociación existentes entre ellas. Es realizado por el diseñador del sistema y está orientado a los programadores. De una buena estructuración de las clases a implementar depende la posibilidad y facilidad de soporte y mantenimiento del producto. A continuación, la figura 3.1 muestra el diagrama de clases del mecanismo de comunicación de Xerberos. En los anexos 2 y 3 se muestra el diagrama

ampliado por partes.

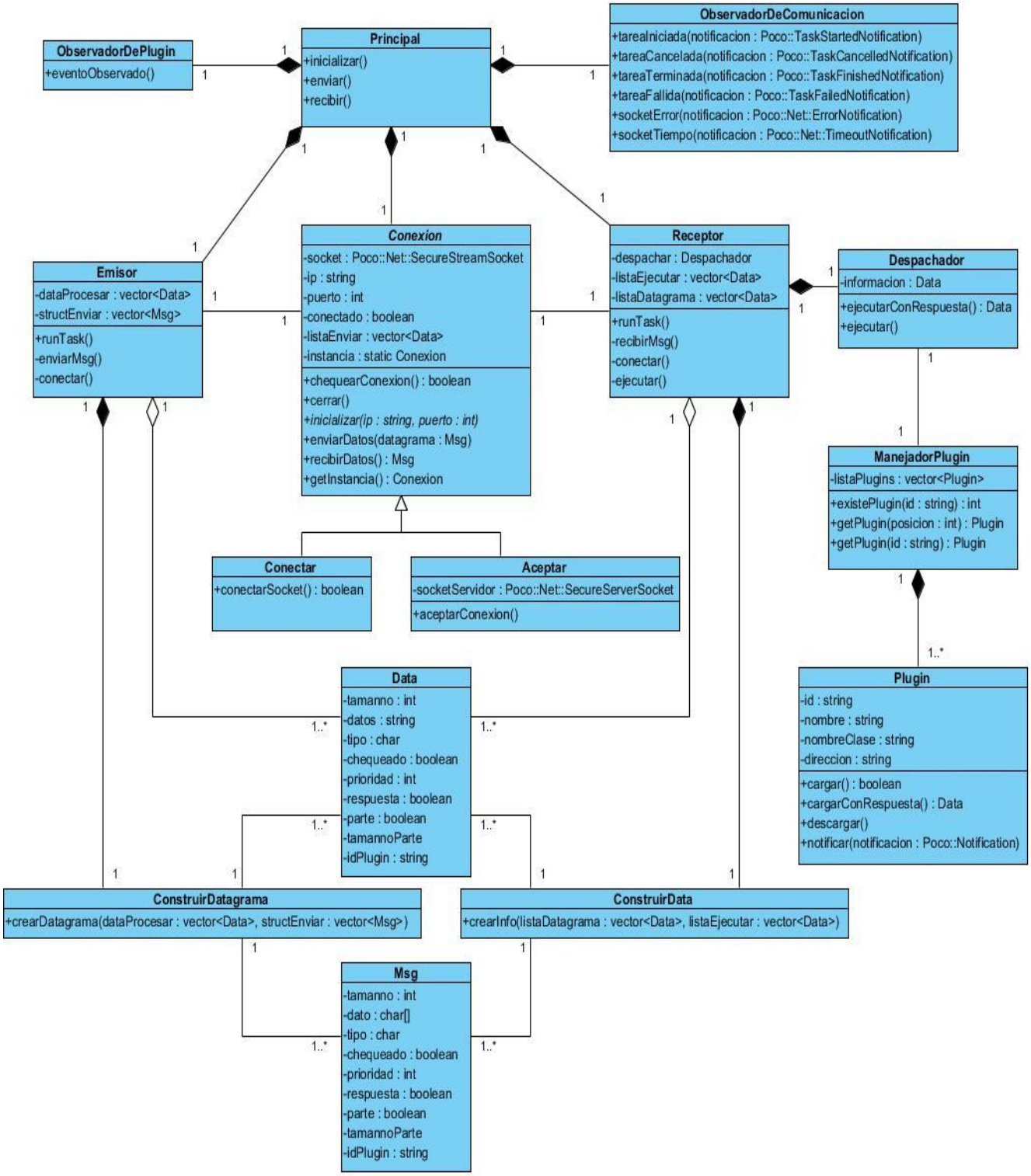


Figura 3.1: Diagrama de Clases del mecanismo de comunicación de Xerberos.

3.3. Descripción de las Clases del Diseño

Para el mecanismo de comunicación que se propone se han diseñado 15 clases que cubren todos los requerimientos del mismo, a continuación se describen cada una de estas clases:

Principal

Documentación	Clase que interactúa directamente con el actor del sistema y se comporta como la fachada del mecanismo de comunicación ante este.

Métodos

public inicializar ()	
Documentación	Método que se encarga de: - Configurar los parámetros de los registros. - Manejar clases observadoras. - Inicializar los parámetros de la conexión. Todas estas configuraciones se pueden obtener de un XML o de cualquier otro contenedor de información.
public enviar ()	
Documentación	Método que ejecuta la tarea de enviar, crea una instancia de la clase *Emisor*.
public recibir ()	
Documentación	Método que ejecuta la tarea de recibir, crea una instancia de la clase *Receptor*.

Conexion

Documentación	Es una clase genérica encargada de crear un objeto socket para realizar una conexión entre dos ordenadores y realizar todas las acciones con los sockets que son comunes para los dos tipos de conexiones: *Conectar* y *Aceptar*.

Atributos

private socket : Poco::Net::SecureStreamSocket	
Documentación	Este tipo de objeto se recomienda que sea del tipo Poco::Net::SecureStreamSocket, clase de la Libpoco encargada de crear sockets de flujos (stream) y asegurados con SSL que son los que se utilizarán en la comunicación de Xerberos.
Tipo	Poco::Net::SecureStreamSocket
private ip : string	
Documentación	IP de la PC esclava, la que acepta la conexión.
Tipo	string
private puerto : int	
Documentación	Puerto de la PC esclava, la que acepta la conexión.

Tipo	int
private conectado : boolean	
Documentación	Atributo booleano que dice si el socket está conectado.
Tipo	boolean
Private listaEnviar : vector<Data>	
Documentación	Lista con la información que se va a enviar.
Tipo	vector<Data>
Private instancia : static Conexion	
Documentación	Atributo que tiene una única instancia de la clase *Conexion*, de forma tal que, todas las clases que utilicen un objeto de esta, utilicen una única instancia.
Tipo	static Conexion

Métodos

public chequearConexion () : boolean	
Documentación	Método que devuelve si la conexión está abierta.
public cerrar ()	
Documentación	Método que cierra la conexión del socket. Guarda un registro del evento realizado.
public inicializar (ip : string, puerto : int)	
Parámetros	**ip** Documentación: IP del ordenador esclavo. Tipo: string **puerto** Documentación: Puerto por donde está escuchando el socket del ordenador esclavo. Tipo: int
Documentación	Método que inicializa los parámetros para la conexión entre los socket.
public getInstancia () : Conexion	
Documentación	Este método retorna una única instancia de la clase *Conexion*, la cual puede ser utilizada por múltiples clases.

Conectar

Documentación	Clase especializada de *Conexion*. Es la encargada de crear un objeto Poco::Net::SecureStreamSocket para establecer una conexión con un ordenador esclavo y realizar todas las acciones con los sockets para este tipo de conexión.

Métodos

public conectarSocket () : boolean	
Documentación	Método que establece la conexión con el Socket Servidor de un ordenador esclavo. Guarda un registro del evento realizado.

Aceptar

Documentación	Clase especializada de *Conexion*. Es la encargada de crear un objeto Poco::Net::SecureServerSocket para permitir una conexión desde un ordenador maestro y realizar todas las acciones con los sockets para este tipo de conexión.

Atributos

private socketServidor : Poco::Net::SecureServerSocket	
Documentación	Este objeto se recomienda que sea del tipo Poco::Net::SecureServerSocket, clase de la Libpoco encargada de crear un socket servidor asegurado con el protocolo SSL, el cual es necesario para permitir la conexión segura desde los ordenadores maestros.
Tipo	Poco::Net::SecureServerSocket

Métodos

public aceptarConexion ()	
Documentación	Método encargado de aceptar las conexiones de los ordenadores maestros que estén conectándose al socket servidor. Utiliza el método acceptConnection() de la clase Poco::Net::SecureServerSocket, el cual devuelve un nuevo socket del tipo StreamSocket con el cual se maneja el envío de datos entre los ordenadores maestros y el esclavo (entre clientes y servidor). Guarda un registro cuando llega una petición de conexión.

Receptor

Documentación	Es la clase encargada de recibir los datagramas que se enviaron desde un ordenador y ejecutarlos. Se recomienda que esta clase herede de la clase Poco::Task, clase de la Libpoco que es manejada por la clase Poco::TaskManager, la cual, permite correr varios Task al mismo tiempo, o sea, implementa un escenario multihilo, muy importante para el

	adecuado funcionamiento del sistema.

Atributos

private despachar : Despachador	
Documentación	Atributo que crea una instancia de la clase *Despachador* para mandar a ejecutar la información que se recibe desde otro ordenador.
Tipo	Despachador
private listaEjecutar : vector <Data>	
Documentación	Lista con los datos disponibles a ejecutar.
Tipo	vector <Data>
private listaDatagrama : vector <Data>	
Documentación	Lista con los datos que se reciben y se van a procesar.
Tipo	vector <Data>

Métodos

public runTask ()	
Documentación	Método principal de la clase, se encarga de verificar la conexión, si está abierta, llama al método recibirMsg(). En caso contrario llama al método conectar(). Además, llama al método ejecutar() para ejecutar las órdenes que estén disponibles en la lista.
private recibirMsg ()	
Documentación	Método que recibe un datagrama que fue enviado desde otro ordenador y manda a que la clase *ConstruirData* lo procese y lo coloque en la lista de disponibles para ser ejecutado. Guarda un registro del evento realizado.
private conectar ()	
Documentación	Método que intenta establecer una conexión en caso de que esta no exista. Guarda un registro del evento realizado.
public ejecutar ()	
Documentación	Método que manda a ejecutar las órdenes que se reciben. Para esto utiliza el atributo despachar.

ConstruirData

Documentación	Es la clase encargada de construir la información que se ejecutará a partir de uno o varios datagramas que se reciben.

Métodos

public crearInfo (listaDatagrama : vector <Data>, listaEjecutar : vector <Data>)

Parámetros	**listaDatagrama**	
	Documentación	Lista que contiene todos los datagramas necesarios para construir una información.
	Tipo	vector<Data>
	listaEjecutar	
	Documentación	Lista que contiene los datos que están disponibles para ser ejecutados.
	Tipo	vector<Data>
Documentación	Este método se encarga de construir la información a partir de uno o varios datagramas recibidos.	

Despachador

Nombre	Valor
Documentación	Es la clase encargada de manejar las peticiones, o sea, a partir de una información recibida, ejecutar el plugin que le corresponde.

Atributos

private informacion : Data	
Documentación	Información que se va a ejecutar.
Tipo	 Data

Métodos

public ejecutarConRespuesta () : Data	
Documentación	Método que ejecuta el plugin correspondiente y da una respuesta.
public ejecutar ()	
Documentación	Método que ejecuta el plugin correspondiente y no da respuesta.

Emisor

Documentación	Es la clase encargada de enviar los datagramas a un ordenador. Al igual que la clase *Receptor*, se recomienda que esta clase herede de la clase Poco::Task, clase de la Libpoco que es manejada por la clase Poco::TaskManager, la cual, permite correr varios Task al mismo tiempo, o sea, implementa un escenario multihilo, muy importante para el adecuado funcionamiento del software.

Atributos

private dataProcesar : vector <Data>	
Documentación	Lista con los datos disponibles a enviar antes de procesados.
Tipo	vector <Data>
private structEnviar : vector <Msg>	
Documentación	Lista con los datos que se van a enviar después de procesados.
Tipo	vector <Msg>

Métodos

public runTask ()	
Documentación	Método principal de la clase, se encarga de verificar la conexión, si está abierta, llama al método enviarMsg(). En caso contrario, llama al método conectar().
private enviarMsg ()	
Documentación	Método que envía un datagrama a otro ordenador. Guarda un registro del evento realizado.
private conectar ()	
Documentación	Método que intenta establecer una conexión en caso de que esta no exista. Guarda un registro del evento realizado.

ConstruirDatagrama

Documentación	Es la clase encargada de construir uno o varios datagramas de la información que se enviará hacia un ordenador, o sea, si la información a enviar es muy grande, se deben construir varios datagramas de 256 bytes con la información a enviar.

Métodos

public crearDatagrama (dataProcesar : vector <Data>, structEnviar : vector <Msg>)		
Parámetros	**dataProcesar**	
	Documentación	Información disponible para enviar antes de procesada.
	Tipo	vector <Data>
	structEnviar	
	Documentación	Información que se va a enviar después de procesada.
	Tipo	vector <Msg>
Documentación	Este método se encarga de construir uno o varios	

	datagramas a partir de la información que se desea enviar.

Data

Documentación	Es la clase encargada de crear el objeto que tendrá la información que se desea enviar, antes de ser procesada y que se recibe después de ser procesada y por tanto es la información que se ejecuta.

Atributos

private tamanno : int	
Documentación	Tamaño del mensaje completo.
Tipo	int
private datos : string	
Documentación	Información que se envía o recibe.
Tipo	string
private tipo : char	
Documentación	Tipo de mensaje: c-comando, f-fichero, v-vida r-respuesta.
Tipo	char
private chequeado : boolean	
Documentación	Indica si el mensaje se ha chequeado.
Tipo	boolean
private prioridad : int	
Documentación	Indica la prioridad del mensaje.
Tipo	int
private respuesta : boolean	
Documentación	Indica si se debe dar respuesta del mensaje.
Tipo	boolean
private parte: boolean	
Documentación	Dice si el mensaje es parte de otro.
Tipo	boolean
private tamannoParte : int	
Documentación	Tamaño de la información que se envía.
Tipo	int

private idPlugin : string	
Documentación	Especifica el ID del plugin que ejecuta dicha tarea.
Tipo	string

Métodos

public get_____ ()	
Documentación	Los métodos de esta clase son los Set y los Get de todos los atributos.

Msg

Documentación	Esta no es una clase como tal, sino que es un objeto tipo estructura (struct). Este objeto se utiliza para crear los datagramas que permitirán el intercambio de datos entre dos ordenadores, pues este tipo de dato es el que se envía por la red y tiene un tamaño de 256 bytes.

Atributos

private tamanno : int	
Documentación	Tamaño del Mensaje completo, Max = 65536 (2 bytes).
Tipo	int
private dato : char[]	
Documentación	Fracción de datos que se envía o recibe. Tiene un tamaño máximo de 250 bytes.
Tipo	char[]
private tipo : char	
Documentación	Tipo de mensaje: c-comando, f-fichero, v-vida r-respuesta (1 byte)
Tipo	char
private chequeado : boolean	
Documentación	Indica si el mensaje se ha chequeado.
Tipo	boolean
private prioridad : int	
Documentación	Indica la prioridad del mensaje.
Tipo	int
private respuesta : boolean	
Documentación	Indica si se debe dar respuesta del mensaje.

Tipo	boolean
private parte : boolean	
Documentación	Dice si el mensaje es parte de otro.
Tipo	boolean
private tamannoParte : int	
Documentación	Tamaño de la información que se envía.
Tipo	int
private idPlugin : string	
Documentación	Especifica el ID del plugin que ejecuta dicha tarea.
Tipo	string

ManejadorPlugin

Documentación	Es la clase que se encarga de manejar la lista de los plugins disponibles.

Atributos

private listaPlugins : vector <Plugin>	
Documentación	Lista de los plugins disponibles.
Tipo	vector <Plugin>

Métodos

public existePlugin (id : string) : int		
Parámetros	**id**	
	Documentación	ID del plugin que se va a buscar.
	Tipo	string
Documentación	Busca el plugin cuyo ID es pasado por parámetro y retorna la posición en la que este se encuentra en la lista. Guarda un registro si el plugin solicitado no se encuentra disponible.	
public getPlugin (posicion : int) : Plugin		
Parámetros	**posicion**	
	Documentación	Posición del plugin en la lista.
	Tipo	int
Documentación	Devuleve un plugin dado la posición en la que este se encuentra en la lista. Guarda un registro si el plugin solicitado no se encuentra disponible.	

public getPlugin (id : string) : Plugin		
Parámetros	id	
	Documentación	ID del plugin.
	Tipo	string
Documentación	Devuelve un plugin dado el ID del mismo. Guarda un registro si el plugin solicitado no se encuentra disponible.	

Plugin

Documentación	Es la clase encargada de crear el objeto que ejecutará las órdenes que dé el usuario.

Atributos

private id : string	
Documentación	ID del plugin
Tipo	string
private nombre : string	
Documentación	Nombre del plugin.
Tipo	string
private nombreClase : string	
Documentación	Nombre de la clase que implementa el plugin.
Tipo	string
private direccion : string	
Documentación	Dirección donde se encuentra disponible el plugin.
Tipo	string

Métodos

public cargar () : boolean	
Documentación	Método que carga el plugin. Retorna "true" si es ejecutado. Para cargar el plugin se recomienda utilizar el método loadLibrary() de la clase Poco::ClassLoader. Guarda un registro del evento realizado.
public cargarConRespuesta () : Data	
Documentación	Método que carga el plugin y construye una respuesta. Para cargar el plugin se recomienda utilizar el método loadLibrary() de la clase Poco::ClassLoader. Guarda un registro del evento realizado.

public descargar ()		
Documentación	Método que termina la ejecución del plugin. Guarda un registro del evento realizado.	
public notificar (notification : Poco::Notification)		
Parámetros	**notificacion**	
	Documentación	Notificación que se va a lanzar.
	Tipo	Poco::Notification
Documentación	Método que se encarga de lanzar una notificación.	

Guardar un registro del evento realizado

Existen algunos métodos en los que se ha descrito que es necesario realizar la acción de "Guardar un registro del evento realizado". Esta consiste en escribir en un fichero los eventos realizados en el sistema que no tengan un impacto importante en otras acciones y por esta razón, no es necesario notificarlo a un observador.

Para realizar esta acción, se utiliza del framework SistClon, la clase LogMsg. Primeramente se tienen que inicializar los parámetros de configuración del registro (esto se realiza en la clase *Principal*), entre los que se encuentran:

- Dirección donde se almacenarán.
- Formato en el que se almacenará.
- Capacidad máxima para la rotación del registro.
- Indicar true o false si los registros deben comprimirse.

Cuando sea necesario guardar un registro de algún evento del sistema, se implementa como se indica a continuación:

SistClon::Log::LogMsg::getInstance()->logMsg(Título, Texto, Prioridad);

Entre las prioridades más utilizadas se encuentran:

Poco::Message::PRIO_INFORMATION, PRIO_ERROR y PRIO_WARNING.

Las demás prioridades disponibles se pueden consultar en la clase Poco::Message, de la libpoco.

Implementación del patrón Observador

En la descripción de las clases *Emisor* y *Receptor* se propuso que sean del tipo Poco::Task, para que puedan ser manejadas por la clase Poco::TaskManager, la cual puede inicializarlas, cancelarlas y ejecutarlas en varios hilos al mismo tiempo. Esta clase tiene un Poco::NotificationCenter incorporado, el cual funciona como un despachador de notificaciones y tiene como función notificar a todos los observadores de las notificaciones que cumplan determinada circunstancia.

Por esto, para implementar el patrón Observador se realizan las siguientes acciones:

1- Se implementa la clase *ObservadorDeComunicacion*, la cual se comportará como clase observadora de los eventos de ejecución de las tareas y de las conexiones entre los socket. En esta clase se implementa un método por cada tipo de notificación que se vaya a observar y en el cual se implementará una acción a realizar cuando sea lanzada alguna notificación.

2- En la clase *Principal*, se añaden al Poco::NotificationCenter los observadores que tendrá, especificándole el nombre de la clase observadora, la notificación que estará observando y el método de esta clase que se ejecutará cuando se publique una notificación.

En resumen, cuando algún objeto publica una notificación con el método postNotification(), el Poco::NotificationCenter invoca al método registrado en cada observador coincidente.

Las notificaciones son del tipo Poco::Notification. La libpoco tiene implementados varios tipos de notificaciones y es muy útil utilizarlas pues no tienen que ser publicadas por ningún objeto, sino que, cuando ocurren estas, el objeto por si solo las publica en el Poco::NotificationCenter. Entre las notificaciones que se recomienda utilizar se encuentran:

- TaskStartedNotification, TaskCancelledNotification, TaskFinishedNotification, entre otras, que se encargan de las notificaciones que realiza un objeto tipo Task.
- Poco::Net::SocketNotification, se encarga de las notificaciones que realizan los sockets.

También se pueden implementar tantas notificaciones como se estimen convenientes, siempre heredando de la clase Poco::Notification.

3- Además, se implementa la clase *ObservadorDePlugin*, la cual se comportará como clase observadora de las acciones de los plugins. En esta clase se implementa un método por cada tipo de notificación que se vaya a observar y en el cual se implementará una acción a realizar cuando sea publicada alguna notificación. Los métodos a implementar no se propondrán pues no se tiene el conocimiento de todos los plugins con los que contará el sistema, ni las características de cada uno de ellos. Por esta razón, cuando un programador del proyecto desarrolle un plugin, debe también desarrollar el o los métodos que necesita el plugin en la clase observadora.

A continuación se muestran las descripciones de las clases que implementan el patrón Observador:

ObservadorDeComunicacion

Documentación	Clase que se comportará como observadora de los eventos de ejecución de las tareas y de las conexiones entre los sockets.

Métodos

public tareaIniciada (notificacion : Poco::TaskStartedNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::TaskStartedNotification
Documentación	Método que ejecuta una acción cuando se inicia una tarea, es decir, cuando un objeto de tipo Task se inicia.	
public tareaCancelada (notificacion : Poco::TaskCancelledNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::TaskCancelledNotification
Documentación	Método que ejecuta una acción cuando se inicia una tarea, es decir, cuando un objeto de tipo Task se cancela.	
public tareaTerminada (notificacion: Poco::TaskFinishedNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::TaskFinishedNotification
Documentación	Método que ejecuta una acción cuando se inicia una tarea, es decir, cuando un objeto de tipo Task finaliza.	
public tareaFallida (notificacion: Poco::TaskFailedNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::TaskFailedNotification
Documentación	Método que ejecuta una acción cuando se inicia una tarea, es decir, cuando un objeto de tipo Task falla.	
public socketError (notificacion: Poco::Net::ErrorNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::Net::ErrorNotification
Documentación	Método que ejecuta una acción cuando recibe una notificación de que el socket ha señalado algún error.	
public socketTiempo (notificacion: Poco::Net::TimeoutNotification)		
Parámetros	**Notificacion**	
	Documentación	Notificación que recibe el método.
	Tipo	Poco::Net::TimeoutNotification

Documentación	Método que ejecuta una acción cuando recibe una notificación de que el socket no ha realizado ningún evento en un tiempo determinado.

ObservadorDePlugin

Documentación	Clase que se comportará como observadora de las acciones de los plugins. En esta clase se implementa un método por cada tipo de notificación que se vaya a observar y en el cual se desarrollará una acción a realizar cuando sea lanzada alguna notificación. Los métodos de esta clase no se propondrán pues no se tiene el conocimiento de todos los plugins con los que contará el sistema, ni las características de cada uno de ellos.

3.4. Diagramas de Secuencia

Entre los artefactos que se generan en el modelo de diseño se encuentran los diagramas de interacción, que se clasifican a su vez en diagramas de colaboración y de secuencia. Estos son dos de los cinco tipos de diagramas UML que se utilizan para modelar los aspectos dinámicos de los sistemas. Un diagrama de interacción muestra una interacción, que consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes. Muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. El diagrama de secuencia contiene detalles de la implementación del escenario, incluyendo los objetos y clases que se usan para implementarlo, y los mensajes intercambiados entre los objetos. Es muy útil utilizar este diagrama para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones. No están pensados para mostrar lógicas de procedimientos complejos pues el detalle del diagrama depende de la fase en la que se encuentre el desarrollo del sistema y de lo que se quiere lograr con la realización del mismo.

En la metodología SXP, el modelo de diseño se realiza en la fase de Planificación-Definición. Entre los artefactos que se deben generar en esta fase no se contempla el diagrama de secuencia, pero aún así, se decide utilizarlo para brindar al programador una mejor explicación de la interacción entre los objetos del sistema de dos de las cuatro historias de usuario definidas en el epígrafe 2.4.

La figura que se muestra a continuación corresponde al diagrama de secuencia de la historia de usuario “Enviar datos a un ordenador”.

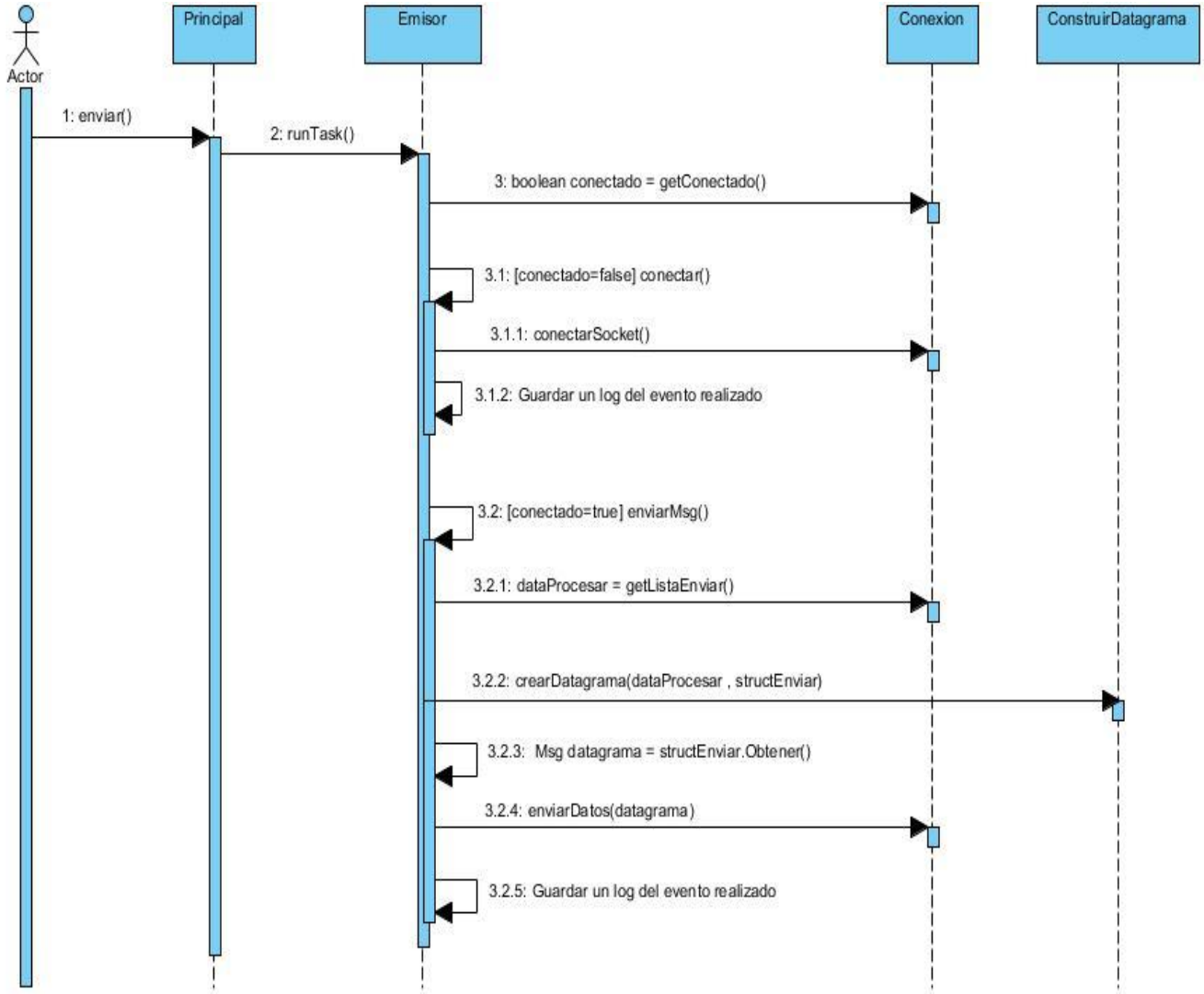


Figura 3.2: Diagrama de Secuencia de la historia de usuario “Enviar datos a un ordenador”.

La figura que se muestra a continuación corresponde al diagrama de secuencia de la historia de usuario “Recibir datos de un ordenador”.

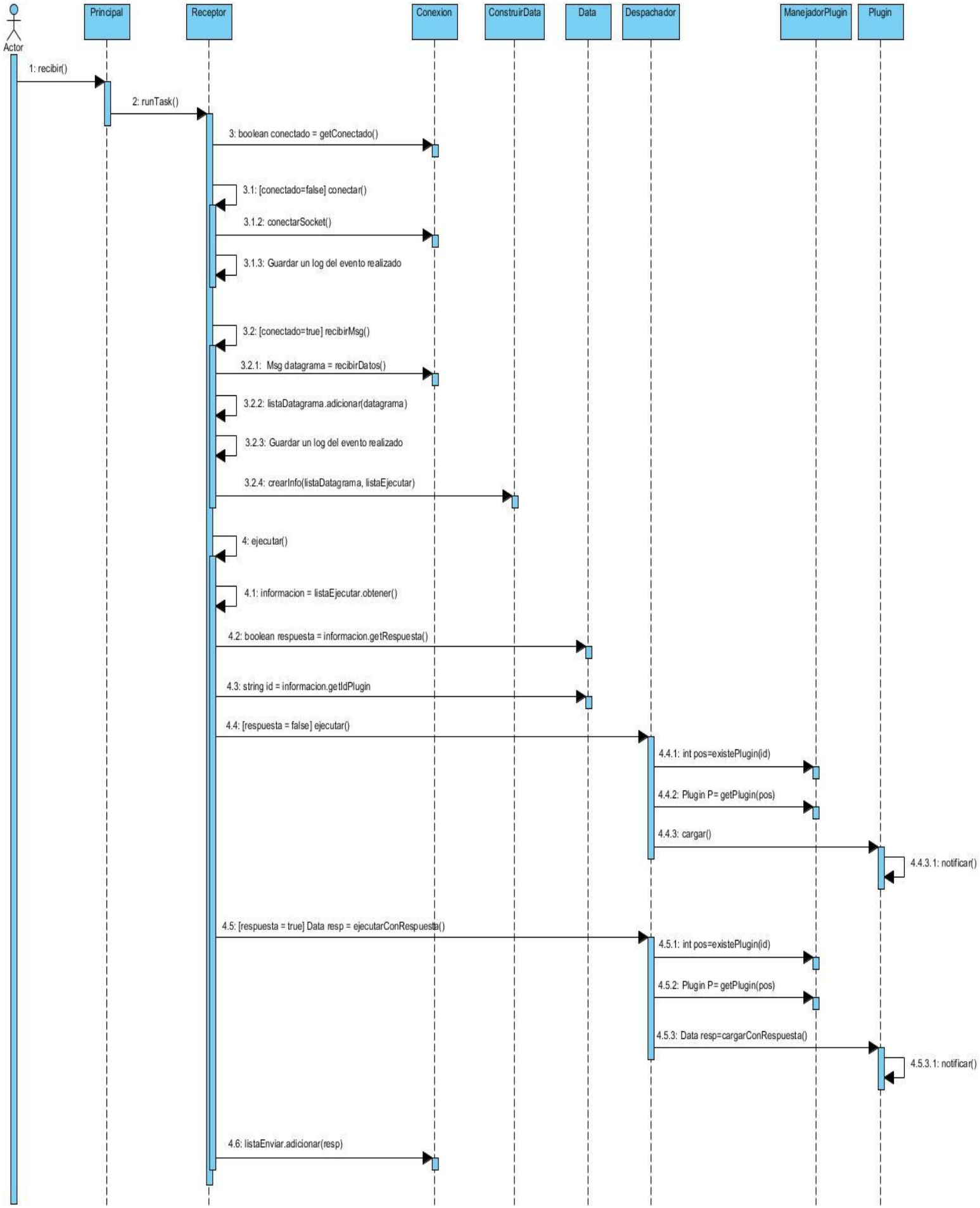


Figura 3.3: Diagrama de Secuencia de la historia de usuario “Recibir datos de un ordenador”.

3.5. Patrones de diseño

Un patrón de diseño es un conjunto de reglas que describen cómo afrontar tareas y solucionar problemas que surgen durante el desarrollo de software. Las ventajas del uso de patrones son que brindan soluciones concretas, técnicas y simples en situaciones muy comunes.

Según su finalidad se pueden considerar varios grupos de patrones, entre ellos:

- **Patrones de creación.** Se encargan de crear objetos de diferentes clases, de manera que no tengan que ser instanciados directamente por el programador. Esto proporciona a los programas una mayor flexibilidad para decidir qué objetos usar.
- **Patrones estructurales.** Permiten crear estructuras de objetos para realizar tareas complejas.
- **Patrones de comportamiento.** Definen la comunicación entre los objetos del sistema y el flujo de información entre ellos, es decir, reparten las responsabilidades de las distintas clases u objetos.
- **Patrones para aplicaciones concurrentes.** Definen patrones de diseño útiles para la programación concurrente.

Patrones utilizados en el diseño del mecanismo de comunicación de Xerberos

Observador (Observer):

Es un patrón de comportamiento. La idea principal detrás del patrón observador es que existe una entidad con estados cambiantes y una o más entidades observándola. Define una dependencia de uno a muchos entre objetos, de forma que cuando el objeto observado cambia de estado, se notifica y actualizan automáticamente todos los objetos que están observándolo. [33]

En el mecanismo de comunicación, el patrón Observador se utiliza para observar todos los eventos que ocurren en el sistema y tienen un impacto significativo para algún otro componente, de forma que, cada vez que ocurre un evento que se desea monitorear, se guarda un registro que lo describe y se ejecuta una acción en particular.

Aceptor/Conector (Acceptor/Connector):

Es un patrón para aplicaciones concurrentes, el cual, en un sistema distribuido, desacopla el establecimiento de la conexión y la inicialización de un servicio, de la transformación que se realiza una vez que el servicio se inicia. Este desacoplamiento se logra con tres componentes: aceptadores, conectores y controladores de servicio. [34]

El conector, de forma activa, establece una conexión con un servidor remoto (generalmente administrados por un aceptor) y se inicializa un controlador de servicio para administrar la conexión. El aceptor espera pasivamente por las solicitudes de conexión (por lo general de un conector remoto) y establece una conexión a la llegada de una solicitud de conexión. Este patrón es muy utilizado en una aplicación cuando se necesita establecer una conexión entre un par de servicios, por ejemplo,

ordenadores en una red, como es el caso de Xerberos. Por esta razón, el mecanismo de comunicación que se propone contiene dos clases llamadas *Aceptar* y *Conectar*, de forma que se instancia una de ellas en dependencia del nodo en que se implemente el mecanismo de comunicación.

Fachada (Facade):

Es un patrón de estructura. Tiene como propósito proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La "fachada" satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas, este proporciona al subsistema independencia y portabilidad. Es muy aconsejable emplearlo cuando se pretende proporcionar una interfaz simple para un subsistema complejo. [35]

Este patrón se utiliza en el diseño del mecanismo de comunicación con la implementación de la clase *Principal*, la cual se comporta como la fachada del mecanismo ante todo el que vaya a hacer uso de las clases del mismo.

Método Plantilla (Template Method):

Es un patrón de comportamiento. El objetivo del mismo es definir el esqueleto de un algoritmo para una operación, dejando para sus subclases la capacidad de redefinir el funcionamiento de los pasos de este algoritmo, siempre y cuando la estructura del mismo permanezca intacta. Este patrón se puede utilizar cuando se quiere "sacar factor común" del comportamiento compartido por varias clases para localizarlo en una única clase, que evita la duplicación de código. A esta técnica se la conoce comúnmente como "refactorizar para generalizar". [36]

Este patrón se emplea en el diseño de la clase *Conexion* la cual implementa las operaciones comunes para los dos tipos de conexiones que son las clases *Aceptar* y *Conectar*, las cuales heredan de esta y contienen las operaciones específicas para cada una.

Solitario (Singlenton):

Es un patrón de creación, cuyo propósito principal es garantizar que una clase solo tiene una única instancia, proporcionando un punto de acceso global a la misma. Se usa cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes. [37]

En el mecanismo de comunicación se emplea este patrón en la clase *Conexion*, la cual contiene un atributo de tipo estático (static) llamado "instancia". Este atributo hace referencia a la instancia de la clase, de manera que, cuando otra clase vaya a hacer uso de la misma, no la instancia de nuevo, sino que hace una llamada al método getInstancia(), obtiene la instancia de la clase y realiza las operaciones necesarias sobre ella.

3.6. Demostración del funcionamiento de la propuesta

Con el objetivo de demostrar la utilización del marco de trabajo que proporciona el mecanismo de comunicación propuesto, se da a conocer un ejemplo que manifiesta, utilizando una funcionalidad real, cómo se integra este a Xerberos. A continuación se explica cómo funcionaría el sistema cuando el usuario decide particionar un disco duro de una estación cliente.

El subsistema de comunicación de la estación de administración utiliza el marco de trabajo para establecer conexión con el servidor y enviar órdenes, en este caso envía la orden de particionar un disco duro a una estación en particular. A su vez, el subsistema de comunicación del servidor central utiliza el marco de trabajo para aceptar conexiones desde las estaciones de administración y los clientes que estén conectados y para recibir y enviar órdenes a ambos. En este caso, recibe desde el usuario la orden de particionar un disco duro de un cliente específico, la cual es manejada por el despachador que ejecuta el plugin correspondiente. Este plugin primeramente comprueba que el cliente esté conectado y luego le envía la orden.

Por otro lado, el subsistema de comunicación de la estación cliente también utiliza el marco de trabajo para establecer conexión con el servidor, además de enviar y recibir órdenes de este, por lo que recibe la orden de particionar el disco duro de la computadora, orden que es manejada por el despachador, el cual ejecuta el plugin que se comunica con el servicio encargado de realizar esta funcionalidad. La figura siguiente muestra de forma gráfica lo anteriormente descrito:

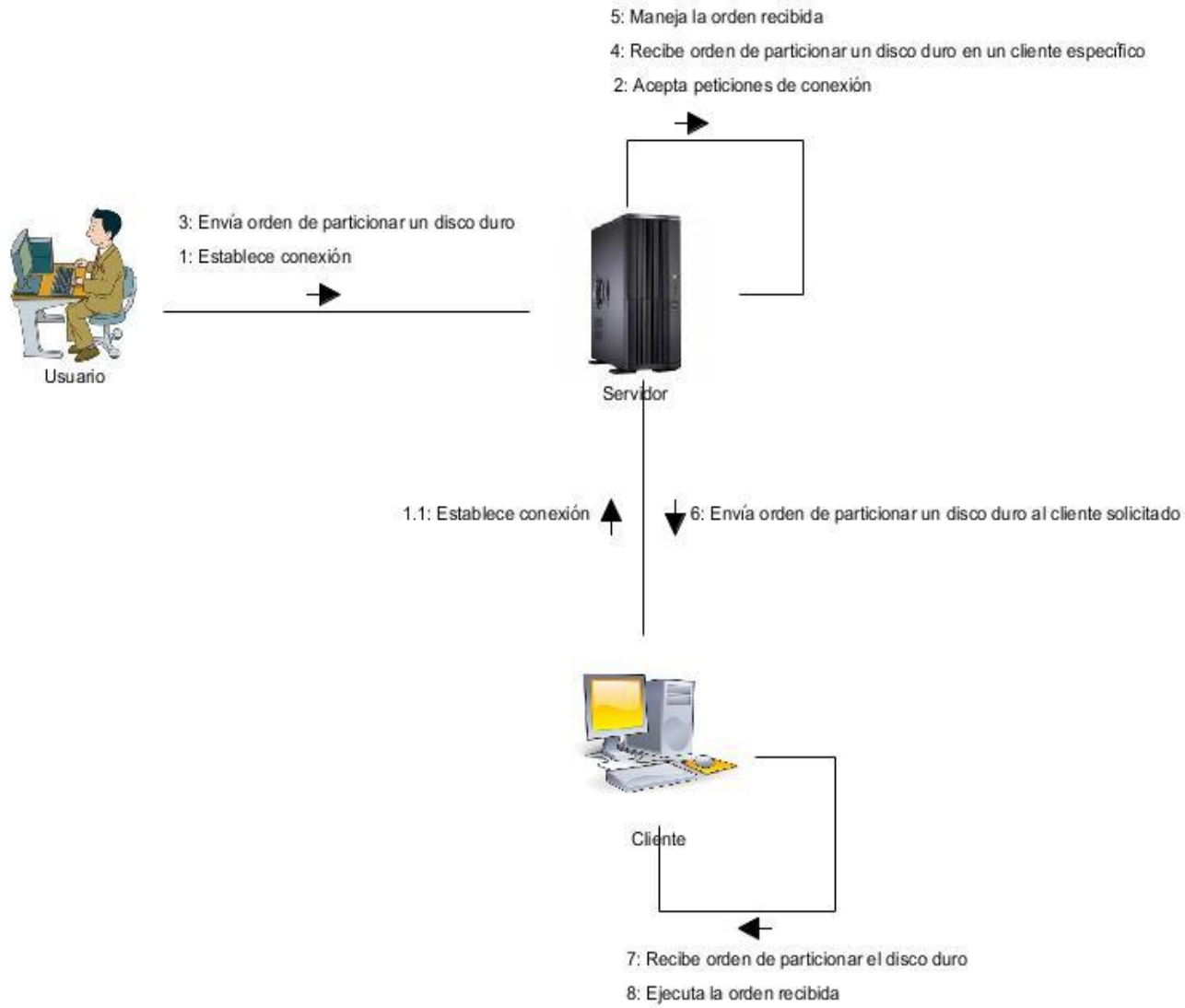


Figura 3.4: Ejemplificación del funcionamiento de Xerberos.

CONCLUSIONES

De manera general en la presente investigación se diseñó un mecanismo de comunicación para Xerberos, permitiendo la gestión de la comunicación en cada uno de los nodos que lo conforman. Entre otros aspectos significativos que concluyen el desarrollo de la presente investigación podemos destacar que:

- El estudio realizado sobre los protocolos de comunicación entre procesos y ordenadores existentes, permitió determinar que el empleo de sockets con los protocolos TCP/IP y SSL garantizará en Xerberos que la gestión de la comunicación se realice de forma eficaz y segura.
- La metodología y las herramientas utilizadas permitieron generar los artefactos capaces de brindar un mejor entendimiento de la propuesta del mecanismo de comunicación.
- El diseño del mecanismo de comunicación propuesto, proporcionará un marco de trabajo (framework) que será utilizado en la implementación del subsistema de comunicación de Xerberos en cada uno de los nodos por los que está compuesto el sistema.

RECOMENDACIONES

Para futuras investigaciones y proyectos que guarden relación con la presente investigación se recomienda:

- Desarrollar un marco de trabajo para el desarrollo de los distintos plugins de Xerberos, con el objetivo de garantizar la correcta integración de estos con el mecanismo de comunicación.

- Extender la propuesta realizada a aquellos sistemas con arquitectura de comunicación similar a la de Xerberos, en el marco de los desarrollos del centro CESOL.

GLOSARIO DE TÉRMINOS

B

Buffer:

Espacio de memoria, en el que se almacenan datos temporalmente.

Byte:

Unidad básica de almacenamiento de información conformada por una secuencia contigua de bits, regularmente de ocho (un bit equivale a un dígito en numeración binaria, que puede ser 0 o 1).

C

Certificado digital:

Es un documento digital mediante el cual un tercero confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad (por ejemplo: nombre, dirección y otros aspectos de identificación) y una clave pública.

D

Datagrama:

Es una de las formas de encaminar los paquetes en una red de conmutación de paquetes. La estructura de un datagrama es: cabecera y datos. La cabecera contiene la dirección de destino, la cual es utilizada por los encaminadores para dirigir los datagramas al destino.

Demonio:

Es un tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Este tipo de programas se ejecutan de forma continua (infinita) y aunque se intente cerrar o matar el proceso, este continuará en ejecución o se reiniciará automáticamente.

E

Escalabilidad:

Es la capacidad de mejorar recursos para ofrecer una mejora (idealmente) lineal en la capacidad de servicio. Es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para extender el margen de operaciones sin perder calidad.

Estaciones clientes:

En Xerberos, son las estaciones a las cuales se les puede brindar soporte y mantenimiento de forma

remota, permitiendo gestionar las particiones, las configuraciones, instalación y desinstalación de software en cada estación.

Estaciones de administración:
En Xerberos, son las estaciones desde las cuales el técnico brinda soporte y mantenimiento a las estaciones clientes, comúnmente llamadas usuarios.

F

FIFO:
First in, first out (en español "primero en entrar, primero en salir"), es un concepto utilizado en teoría de colas.

Framework o marco de Trabajo:
Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H

Hilos:
Un hilo de ejecución, en sistemas operativos, es una característica que permite a una aplicación realizar varias tareas concurrentemente.

I

Interconexión:
Acción y efecto de interconectar. Conexión recíproca entre varios dispositivos.

Interoperabilidad:
Habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

L

Librería .so:
Las librerías son una forma sencilla y versátil de modularizar y reutilizar código. Las .so son librerías que pueden ser cargadas automáticamente por el cargador (ld.so), como resultado de haber sido indicadas durante la fase de compilación. Pero también pueden ser cargadas manualmente utilizando

rutinas especializadas de la librería C del sistema, de manera que un programa puede cargar librerías por sí mismo y enlazarlas a tiempo de ejecución.

Log:

Es un registro oficial de eventos durante un rango de tiempo en particular.

M

Mecanismo:

Estructura interna que hace funcionar algo. Modo de funcionamiento, desarrollo.

N

Nodos:

Es un punto de intersección, conexión o unión de varios elementos que confluyen en el mismo lugar. En redes de computadoras cada una de las máquinas es un nodo, y si la red es Internet, cada servidor constituye también un nodo.

P

Parser:

En español Parseador, es un programa o módulo encargado del parseo de un texto, es decir, transforma una entrada de texto en una estructura de datos.

Plug-in:

Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande. En Xerberos, un plug-in no es más que una librería dinámica con extensión .so, la cual contiene el código necesario para establecer la comunicación por dbus con uno de los servicios del sistema, de manera que, cuando se desee ejecutar alguna funcionalidad, se carga el plugin correspondiente y se hace uso de los métodos o funcionalidades que brinda el mismo, sin tener en cuenta las características de la aplicación desde la que se accede.

Proceso:

Es una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual y un conjunto de recursos del sistema asociados. Puede entenderse como un programa en ejecución.

Protocolo:

En informática, un protocolo es un conjunto de reglas de comunicación que permiten el flujo de

información entre equipos que manejan lenguajes distintos.

S

Servicio:

En Xerberos, un servicio es una aplicación que provee al sistema de una o varias funcionalidades.

Servidores auxiliares:

Se le llama así a los servidores secundarios de Xerberos, entre los que se encuentran: el servidor de base de datos y el servidor de imágenes.

Servidor central:

Es el servidor principal de Xerberos por el que pasa toda la comunicación del sistema. En él ocurren los procesos centrales del software.

Subsistemas:

Es un sistema que es parte de otro sistema mayor (suprasistema o supersistema). Es un conjunto de elementos ordenados o funciones relacionados para cumplir con un propósito o fin determinado y cuyas partes deben reunir ciertas condiciones de tal manera que se complementen formando un sistema.

X

XML:

Es un metalenguaje extensible de etiquetas. Se le denomina metalenguaje pues no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

REFERENCIAS BIBLIOGRÁFICAS

[1] PÉREZ ROLDÁN, Dayron. *Sistema de Clonación y Administración Centralizada de Imágenes de Sistemas Operativos.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2009, página 1.

[2] HERNÁNDEZ LEÓN, Rolando Alfredo and COELLO GONZÁLEZ, Sayda. *El proceso de investigación científica*. Ciudad de la Habana: Editorial Universitaria, 2011. ISBN 978-959-16-1307-3, página 58.

[3] PÉREZ ROLDÁN, Dayron and TORRES SÁNCHEZ, José Ernesto. *Breve descripción del sistema Xerberos*. Expediente de proyecto de Nova-Xerberos. Universidad de las Ciencias Informáticas, página 2. [Consultado: Octubre 2011]. Disponible en: <https://repositorio.geitel.prod.uci.cu/svn/xerberos/Xerberos/stable/descripcion de xerberos.odt>

[4] Anon. *Comunicación entre procesos.* [Consultado: Octubre 2011]. Disponible en: <http://es.wikipedia.org/wiki/Comunicación_entre_procesos>

[5] RUIZ CATALÁN, Jacinto*. Las redes. Transmisión de datos (primera parte)*. 27/08/2008, capítulo 2. [Consultado: Octubre 2011]. Disponible en: <http://www.mailxmail.com/curso-redes-transmicion-datos-1/comunicacion-ordenadores-protocolos-arquitectura>

[6] Anon, *Protocolos de comunicaciones industriales*. Sitio de la Asociación de la Industria Eléctrica en Chile. Agosto 2006. [Consultado: Octubre 2011]. Disponible en: <http://www.aie.cl/files/file/comites/ca/articulos/agosto-06.pdf>

[7] GARCÍA DE JALÓN, Javier and AGUINAGA, Iker and MORA, Alberto. *Aprenda LINUX como si estuviera en primero*. Universidad de Navarra, San Sebastián, Enero 2000, página 57.

[8] CALVO GORDILLO, Isidro. *Colas de Mensajes*. Laboratorio de sistemas informáticos en tiempo real. 2003-2004. [Consultado: Octubre 2011]. Disponible en: <http://www.disa.bi.ehu.es/spanish/asignaturas/17225/practicas/Tareas_comunicacion_sincronizacion_II.pdf>

[9] MARQUEZ GARCIA, Francisco. *UNIX. Programación Avanzada.* 3ª edición, 2004. ISBN 978-84-7897-603-4. Editorial: RA-MA EDITORIAL.

[10] MARSHALL, Dave. Remote Procedure Calls (RPC). Mayo 01, 1999. [Consultado: Octubre 2011]. Disponible en: <http://www.cs.cf.ac.uk/Dave/C/node33.html>

[11] Anon. *RPC.* [Consultado: Octubre 2011]. Disponible en: <http://es.wikipedia.org/wiki/RPC>

[12] WINER, Dave. XML-RPC Home Page. *What is XML-RPC?*. [Consultado: Octubre 2011]. Disponible en: <http://www.xmlrpc.com/>

[13] GONZÁLEZ C, Benjamin. *SOAP (Simple Object Access Protocol).* Julio 07, 2004. [Consultado: Octubre 2011]. Disponible en: <http://www.desarrolloweb.com/articulos/1557.php>

[14] WINER, Dave. *Heads up: A key difference between SOAP and XML-RPC*. Marzo 31, 2001. [Consultado: Octubre 2011]. Disponible en: <http://www.xmlrpc.com/stories/storyReader$1514>

[15] KDE TechBase Home Page. *Introduction To D-BUS (es)*. Junio 29, 2011. [Consultado 17 de Octubre 2011]. Disponible en: <http://techbase.kde.org/Development/Tutorials/D-Bus/Introduction_%28es%29>

[16] ALVAREZ, Miguel Angel, MONTEIRO, Juliana, JUSZKIEWICZ, Leo, WONG, William and Alvarez, Sara. *Tutorial de FTP*. [Consultado 17 Octubre 2011]. Disponible en: <httphttp://www.wiener.edu.pe/manuales2/5to-ciclo/REDES-2/manual-tutorial-ftp.pdf>

[17] PÉREZ ROLDÁN, Dayron. *Sistema de Clonación y Administración Centralizada de Imágenes de Sistemas Operativos.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2009, página 108.

[18] Microsoft TechNet Home Page. *Protocolo de datagramas de usuario (UDP)*. [Consultado 17 Octubre 2011]. Disponible en: <http://technet.microsoft.com/es-es/library/cc785220%28v=ws.10%29.aspx>

[19] W3C Home Page. *HTTP Activity Statement.* Consultado 17 Octubre 2011]. Disponible en: <http://www.w3.org/Protocols/Activity.html>

[20] JIMÉNEZ FERNÁNDEZ, Mikel. Gestión centralizada equipos. Irontec: Internet y Sistemas sobre GNU/Linux

[21] Visual Paradigm Home Page. [Consultado 17 Octubre 2011]. Disponible en: <http://www.visual-paradigm.com/>

[22] Tigris Home Page. [Consultado: Octubre 2011]. Disponible en: <http://rapidsvn.tigris.org/>

[23] Poco Home Page. [Consultado: Octubre 2011]. Disponible en: <http://pocoproject.org/index.html>

[24] TORRES SÁNCHEZ, José E. and BAGAROTTI MARÍN, Marcos. Módulo para el control y reporte del uso de dispositivos externos y aplicaciones. Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2011, página 30.

[25] TORRES SÁNCHEZ, José E. and BAGAROTTI MARÍN, Marcos. Módulo para el control y reporte del uso de dispositivos externos y aplicaciones. Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2011, página 36.

[26] PRESSMAN, Roger S. Ingeniería de software. Un enfoque práctico. Sexta Edición. Página 77.

[27] PEÑALVER, Gladys, Meneses, Abel and García, S. SXP, metodología ágil para el desarrollo de software. Universidad de las Ciencias Informáticas, 2010.

[28] MARTINEZ, Ivet Carolina. Modelo Conceptual/ Modelo de dominio. Universidad Simón Bolívar. [Consultado: Enero 2012]. Disponible en: <http://ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf>

[29] JACOBSON, Ivar. G. B 2004. *El proceso unificado de desarrollo de software.* Habana: Felix Varela.

[30] PEÑALVER ROMERO, Gladys Marsi. *Metodología ágil para proyectos de software libre.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, Junio 2008, página 57.

[31] PRESSMAN, Roger S. *Ingeniería del software. Un enfoque práctico*. Quinta Edición. Página 238.

[32] BAHIT, Eugenia. *El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC.* [Consultado: Febrero 2012]. Disponible en: <http://eugeniabahit.blogspot.com>

[33] Unidad Docente de Ingeniería del Software. Patrones del Gang of Four. Universidad Politécnica de Madrid, Facultad de informática. Página 38.

[34] MARTIN, Robert, BUSCHMANN, Frank and RIEHLE, Dirk. *Pattern Languages of Program Design 3*. Primera edición, Octubre 17, 1997. ISBN-10: 0201310112. ISBN-13: 978-0201310115.

[35] Unidad Docente de Ingeniería del Software. Patrones del Gang of Four. Universidad Politécnica de Madrid, Facultad de informática. Página 25.

[36] Unidad Docente de Ingeniería del Software. Patrones del Gang of Four. Universidad Politécnica de Madrid, Facultad de informática. Página 35.

[37] Unidad Docente de Ingeniería del Software. Patrones del Gang of Four. Universidad Politécnica de Madrid, Facultad de informática. Página 47.

BIBLIOGRAFÍA

HERNÁNDEZ LEÓN, Rolando Alfredo and COELLO GONZÁLEZ, Sayda. *El proceso de investigación científica*. Ciudad de la Habana: Editorial Universitaria, 2011. ISBN 978-959-16-1307-3.

Alvarez de Zayas, Dr. Cs. Carlos. *Metodología de la investigación científica.* Universidad de oriente. Santiago de Cuba, 1995.

PÉREZ ROLDÁN, Dayron and TORRES SÁNCHEZ, José Ernesto. *Breve descripción del sistema Xerberos*. Expediente de proyecto de Nova-Xerberos. Universidad de las Ciencias Informáticas. [Consultado: Octubre 2011]. Disponible en: <https://repositorio.geitel.prod.uci.cu/svn/xerberos/Xerberos/stable/descripcion de xerberos.odt>

VAZQUEZ PAREDES, Jorge Mijail and SÁNCHEZ MARTÍN. Danelys. *SistClon: Sistema de Clonación y Distribución de Imágenes de Sistemas Operativos.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2009.

PÉREZ ROLDÁN, Dayron. *Sistema de Clonación y Administración Centralizada de Imágenes de Sistemas Operativos.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2009.

MARTÍNEZ PÉREZ, Lázaro Damián. *Módulo para la Gestión y el Particionamiento Remoto de Discos Duros del Subsistema de Clonación de Xerberos.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2011.

TORRES SÁNCHEZ, José E. and BAGAROTTI MARÍN, Marcos. *Módulo para el control y reporte del uso de dispositivos externos y aplicaciones.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2011.

TANENBAUM, Andrew S. and WOODHULL, Albert S. *Sistemas Operativos, Diseño e implementación.* Segunda Edición, México, 1997. ISBN 970-17-0165-8.

RUIZ CATALÁN, Jacinto*. Las redes. Transmisión de datos (primera parte)*. 27/08/2008, capítulo 2. [Consultado: Octubre 2011]. Disponible en: http://www.mailxmail.com/curso-redes-transmicion-datos-1/comunicacion-ordenadores-protocolos-arquitectura

GARCÍA DE JALÓN, Javier and AGUINAGA, Iker and MORA, Alberto. *Aprenda LINUX como si estuviera en primero*. Universidad de Navarra, San Sebastián, Enero 2000, página 57.

Anon, *Protocolos de comunicaciones industriales*. Sitio de la Asociación de la Industria Eléctrica en

Chile. Agosto 2006. [Consultado: Octubre 2011]. Disponible en: <http://www.aie.cl/files/file/comites/ca/articulos/agosto-06.pdf>

TEJEDA, Hector. *Comunicación entre procesos (IPC Interprocess Communication), PIPES.* [Consultado: Octubre 2011]. Disponible en: <http://www.fismat.umich.mx/mn1/manual/node23.html>

Anon. *Comunicación entre procesos.* [Consultado: Octubre 2011]. Disponible en: <http://es.wikipedia.org/wiki/Comunicación_entre_procesos>

Anon. *¿Cómo se realiza la comunicación entre computadoras?.* [Consultado: Octubre 2011]. Disponible en: <http://www.cavsi.com/preguntasrespuestas/como-se-realiza-la-comunicacion-entre-computadoras/>

CALVO GORDILLO, Isidro. *Colas de Mensajes*. Laboratorio de sistemas informáticos en tiempo real. 2003-2004. [Consultado: Octubre 2011]. Disponible en: <http://www.disa.bi.ehu.es/spanish/asignaturas/17225/practicas/Tareas_comunicacion_sincronizacion_II.pdf>

Anon. *Tuberías (Pipes).* [Consultado: Octubre 2011]. Disponible en: <http://supercomputo.izt.uam.mx/linux/curso.pdf>

Anon. *¿Que es un Socket? - Definición de Socket.* [Consultado: Octubre 2011]. Disponible en: <http://www.masadelante.com/faqs/socket>

TEJEDA, Héctor. *Sockets.* [Consultado: Octubre 2011]. Disponible en: <http://www.fismat.umich.mx/mn1/manual/node24.html>

MARQUEZ GARCIA, Francisco. *UNIX. Programación Avanzada.* 3ª edición, 2004. ISBN 978-84-7897-603-4. Editorial: RA-MA EDITORIAL.

MARSHALL, Dave. Remote Procedure Calls (RPC). Mayo 01, 1999. [Consultado: Octubre 2011]. Disponible en: <http://www.cs.cf.ac.uk/Dave/C/node33.html>

Asran. *Protocolo RPC.* [Consultado: Octubre 2011]. Disponible en: <http://articulos-blackhat4all.blogspot.com/2008/06/protocolo-rpc.html>

Anon. *RPC.* [Consultado: Octubre 2011]. Disponible en: <http://es.wikipedia.org/wiki/RPC>

WINER, Dave. XML-RPC Home Page. *What is XML-RPC?*. [Consultado: Octubre 2011]. Disponible en: <http://www.xmlrpc.com/>

GONZÁLEZ C, Benjamin. *SOAP (Simple Object Access Protocol).* Julio 07, 2004. [Consultado: Octubre 2011]. Disponible en: <http://www.desarrolloweb.com/articulos/1557.php>

WINER, Dave. *Heads up: A key difference between SOAP and XML-RPC*. Marzo 31, 2001. [Consultado: Octubre 2011]. Disponible en: <http://www.xmlrpc.com/stories/storyReader$1514>

RAMIREZ, Edgar. *SOAP*. Blog informático, 14 de octubre de 2008. [Consultado: Octubre 2011]. Disponible en: <http://edgarramirez.wordpress.com/2008/10/14/soa/>

KDE TechBase Home Page. *Introduction To D-BUS (es)*. Junio 29, 2011. [Consultado: Octubre 2011]. Disponible en: <http://techbase.kde.org/Development/Tutorials/D-Bus/Introduction_%28es%29>

PENNINGTON, Havoc; CARLSSON, Anders; LARSSON, Alexander; HERZBERG, Sven; MCVITTIE, Simon and ZEUTHEN, David. *D-Bus Specification*. [Consultado: Octubre 2011]. Disponible en: <http://dbus.freedesktop.org/doc/dbus-specification.html>

Anon. *Protocolo FTP (Protocolo de transferencia de archivos).* [Consultado: Octubre 2011]. Disponible en: <http://es.kioskea.net/contents/internet/ftp.php3>

ALVAREZ, Miguel Angel, MONTEIRO, Juliana, JUSZKIEWICZ, Leo, WONG, William and Alvarez, Sara. *Tutorial de FTP*. [Consultado 17 Octubre 2011]. Disponible en: <httphttp://www.wiener.edu.pe/manuales2/5to-ciclo/REDES-2/manual-tutorial-ftp.pdf>

Anon. *File Transfer Protocol.* [Consultado: Octubre 2011]. Disponible en: <http://es.wikipedia.org/wiki/File_Transfer_Protocol>

Anon. *¿Que es el TCP/IP? - Definición de TCP/IP.* [Consultado: Octubre 2011]. Disponible en: <http://www.masadelante.com/faqs/tcp-ip>

BARAJAS, Saulo. *Curso de protocolos TCP/IP.* [Consultado: Octubre 2011]. Disponible en: <http://www.saulo.net/pub/tcpip/>

Anon. *TCP/IP.* [Consultado: Octubre 2011]. Disponible en: <http://es.kioskea.net/contents/internet/tcpip.php3>

Anon. *El Protocolo UDP.* Redes y Seguridad, 18 de Febrero de 2009. [Consultado: Octubre 2011]. Disponible en: <http://www.redesyseguridad.es/el-protocolo-udp/>

Anon. *Protocolos TCP y UDP en el nivel de transporte.* [Consultado: Octubre 2011]. Disponible en:

<http://es.wikibooks.org/wiki/Redes_inform%C3%A1ticas/Protocolos_TCP_y_UDP_en_el_nivel_de_transporte>

Microsoft TechNet Home Page. *Protocolo de datagramas de usuario (UDP)*. [Consultado 17 Octubre 2011]. Disponible en: <http://technet.microsoft.com/es-es/library/cc785220%28v=ws.10%29.aspx>

W3C Home Page. *HTTP Activity Statement*. [Consultado 17 Octubre 2011]. Disponible en: <http://www.w3.org/Protocols/Activity.html>

Anon. *Definición de HTTP.* [Consultado: Octubre 2011]. Disponible en: <http://www.definicionabc.com/tecnologia/http.php>

Anon. *Definición de HTTP.* Master Magazine. [Consultado: Octubre 2011]. Disponible en: <http://www.mastermagazine.info/termino/5288.php>

Anon. *HTTPS.* [Consultado: Octubre 2011]. Disponible en: <http://www.hooping.net/glossary/https-57.aspx>

Anon. *Definición de HTTPS.* [Consultado: Octubre 2011]. Disponible en: <http://www.definicion.org/https>

ÁLVAREZ MARAÑÓN, Gonzalo. *Secure Socket Layer (SSL).* Web Seguro, 1998. [Consultado: Octubre 2011]. Disponible en: <http://www.iec.csic.es/criptonomicon/ssl.html>

Anon. *¿Qué son los certificados SSL?.* [Consultado: Octubre 2011]. Disponible en: <http://www.certstopshop.com/QuesonlosCertificadosSSL.aspx>

Anon. *Protocolo SSL.* [Consultado: Octubre 2011]. Disponible en: <http://www.pedroximenez.com/ssl.htm>

YLÖNEN, Tatu. *Protocolo SSH (Secure SHell).* [Consultado: Octubre 2011]. Disponible en: <http://ingeniatic.euitt.upm.es/index.php/tecnologias/item/560-protocolo-ssh-secure-shell>

OCHOA DE ASPURU, Jorge García. *El protocolo SSH*. Tesis de doctorado en Ciencias de la Computación. ESIDE. [Consultado: Octubre 2011]. Disponible en: <http://www.naguissa.com/archivos.php?path=11&accion=descarga&IDarchivo=58>

BUSTAMANTE, Paul; AGUINAGA, Iker; AYBAR, Miguel; OLAIZOLA, Luis and LAZCANO, Iñigo. *Aprenda C++ básico como si estuviera en primero.* San Sebastián, Febrero 2004.

JIMÉNEZ FERNÁNDEZ, Mikel. Gestión centralizada equipos. Irontec: Internet y Sistemas sobre GNU/Linux

Visual Paradigm Home Page. [Consultado 17 Octubre 2011]. Disponible en: <http://www.visual-paradigm.com/>

Tigris Home Page. [Consultado: Octubre 2011]. Disponible en: <http://rapidsvn.tigris.org/>

Poco Home Page. [Consultado: Octubre 2011]. Disponible en: <http://pocoproject.org/index.html>

OBILTSCHNIG, Günter. *Poco C++ Libraries*. Versión 1.3.6-all, 2012. Disponible en: <http://pocoproject.org/>

PRESSMAN, Roger S. Ingeniería de software. Un enfoque práctico. Sexta Edición.

PEÑALVER, Gladys, Meneses, Abel and García, S. SXP, metodología ágil para el desarrollo de software. Universidad de las Ciencias Informáticas, 2010.

MARTINEZ, Ivet Carolina. Modelo Conceptual/ Modelo de dominio. Universidad Simón Bolívar. [Consultado: Enero 2012]. Disponible en: <http://ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf>

PEÑALVER ROMERO, Gladys Marsi. *Metodología ágil para proyectos de software libre.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de la Habana, Junio 2008.

PRESSMAN, Roger S. *Ingeniería del software. Un enfoque práctico*. Quinta Edición.

BAHIT, Eugenia. *El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC.* [Consultado: Octubre 2011]. Disponible en: <http://eugeniabahit.blogspot.com>

Unidad Docente de Ingeniería del Software. Patrones del Gang of Four. Universidad Politécnica de Madrid, Facultad de informática.

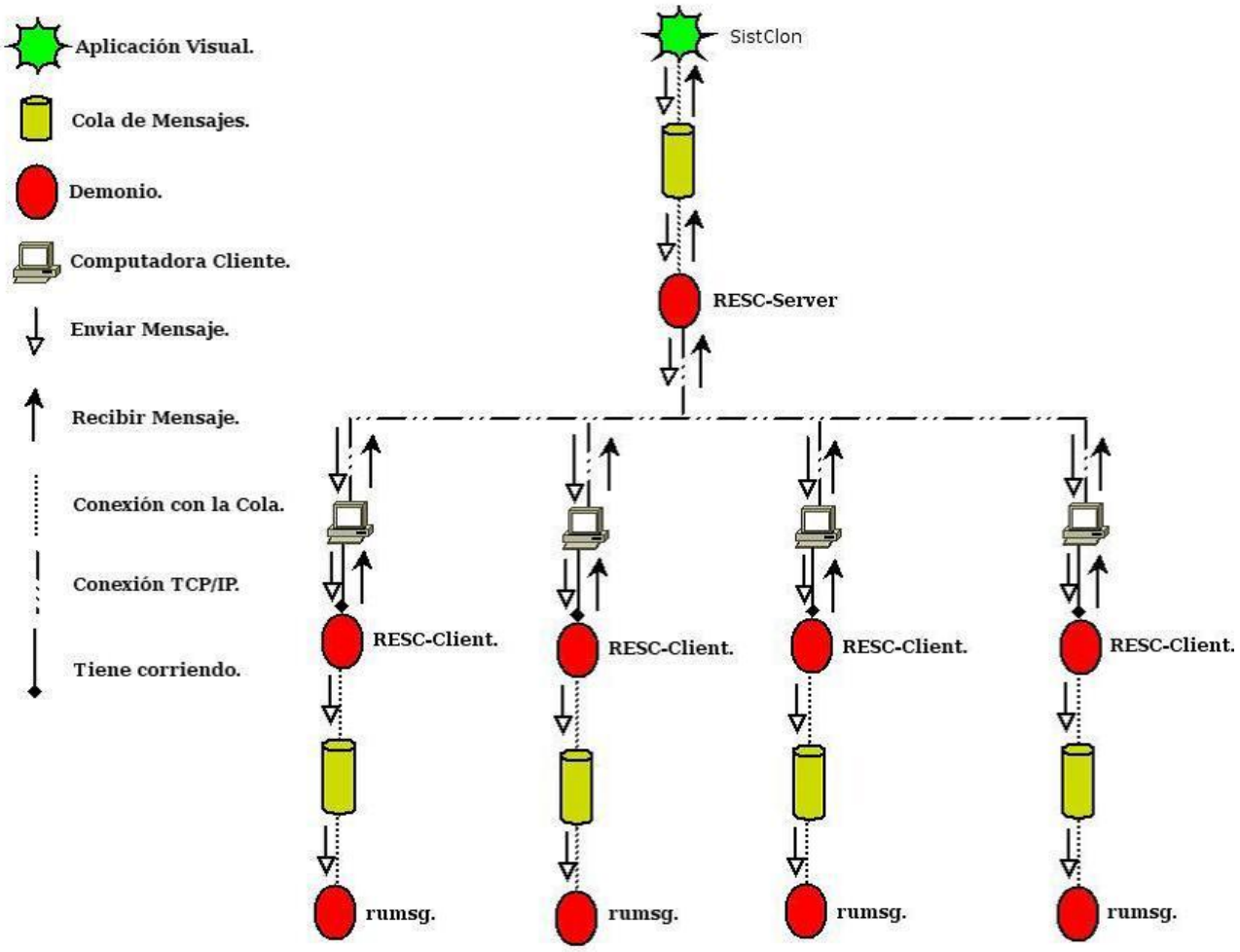
COPLIEN, James O. and SCHMIDT, Douglas C. *Pattern Languages of Program Design*. Primera edición, Mayo 12, 1995. ISBN-10: 0201607344. ISBN-13: 978-0201607345.

MARTIN, Robert; BUSCHMANN, Frank and RIEHLE, Dirk. *Pattern Languages of Program Design 3*. Primera edición, Octubre 17, 1997. ISBN-10: 0201310112. ISBN-13: 978-0201310115.

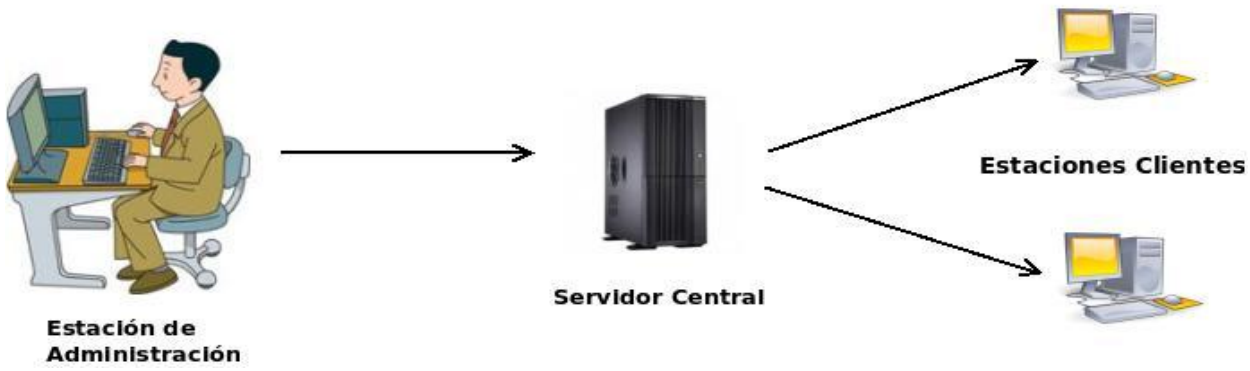
WordReference. *Diccionario Español.* Disponible en: <http://www.wordreference.com/definicion/>

ANEXOS

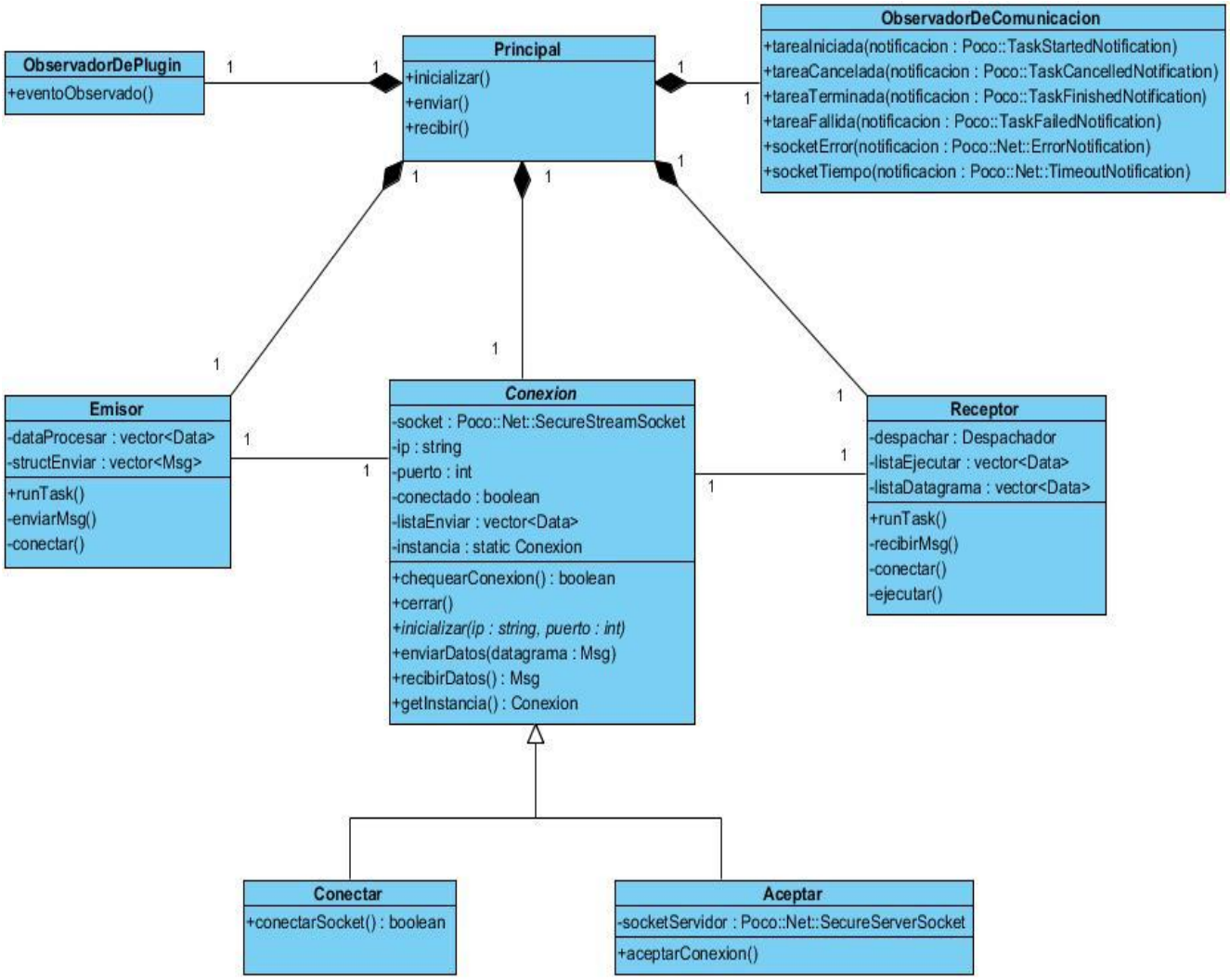
Anexo 1: Modelación de la comunicación en SistClon.



Anexo 2: Componentes físicos (nodos) de Xerberos en los que se implementa el subsistema de comunicación.



Anexo 3: Ampliación 1 del Diagrama de Clases.



Anexo 4: Ampliación 2 del Diagrama de Clases.