Universidad de las Ciencias Informáticas

Facultad 2

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título: "Plataforma para el Análisis Estático de Vulnerabilidades del Código".

MÓDULO: DISTRIBUCIÓN Y EJECUCIÓN DE HERRAMIENTAS.

Autores:

Yaritza Montero Vaillant.

Ernesto Miguel Rodríguez Rodríguez.

Tutores:

Ing. Rogfel Thompson Martínez.

Ing. Adrián Hernández Yeja.

Co-tutor:

Ing. Yosbany Tejas de la Cruz.

La Habana, Junio de 2011.

"Año 53 de la Revolución"

Pensamiento

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

Kruchtel, Philippe

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos al Centro de Telemáticos de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____	_____
Yaritza Montero Vaillant	Ernesto Miguel Rodríguez Rodríguez
Firma del Autor:	**Firma del Autor:**
_____	_____
Ing. Rogfel Thompson Martínez	Ing. Adrián Hernández Yeja
Firma del Tutor:	**Firma del Tutor:**

Agradecimientos

Le agradezco a esas luces tan grandes que son mis padres, que me alumbraron el camino para recorrer sin dificultades. Le doy mil gracias a mi madre Martha por darme siempre amor y apoyo durante toda mi vida, por brindarme todo lo que necesito sin importarle cuanto le cueste, a mi padre Inmer por inculcarme valores morales y amor por la Revolución, por estar presente cuando más lo necesito, en fin le agradezco a ambos por forjarme para que hoy me convirtiese en una profesional, en una ingeniera. Les agradezco por su sacrificio y compañía. Han sido el faro guía en mi vida y el motor impulsor en mis estudios. Gracias mamá y papá, mis dos grandes tesoros más preciados.

Les agradezco a mis dos hermanitas: Yaima y Yainelis que son mis amores, por apoyarme siempre y brindarme su amor, por confiar en mí y tenerme siempre presente. Son un ejemplo para mí de amor y confianza. Le agradezco además a mi hermano Inmer por brindarnos a mis hermanas y a mí su amor y carisma.

Le agradezco a mis abuelos: María y Cristian que durante toda mi vida me han brindado apoyo, amor y dedicación, han sido como mis padres, les agradezco por inculcarme los valores morales que necesita toda persona. Abuelita has sido mi luz inspiradora, ejemplo de esfuerzo y energía.

Les agradezco a mis tíos: Zuzel, Alfredo, Eduardo, Concha y Julio por apoyarme durante toda mi carrera y tenerme presente siempre. Por mostrarme que existo en sus vidas. A mis primitos Rachel y Yasmani. Le agradezco a toda mi familia en sentido general.

Le agradezco a mis suegros Violeta y Miguel por aceptarme como una más de la familia y quererme. A Anet por apoyarme.

Le doy gracias a nuestros tutores por ayudarnos durante todo el desarrollo de la tesis y apoyarnos, sin ellos no hubiese sido posible lograr nuestro objetivo. Agradezco a todos mis profesores desde la primaria hasta la Universidad que me han forjado para hoy convertirme en lo que soy. A mis compañeros de estudio, en especial a Rosana, Anet Hung y Daymí por ser grandes amigas y brindarme su apoyo. Le agradezco a Rainer por ayudarme cuando me hizo falta. A Tito y Pedro por ser buenos compañeros de cuarto.

Le agradezco a mi novio Ernesto por darme todo su amor en estos años, por brindarme su apoyo durante la carrera y lograr juntos este sueño.

En fin gracias a todas las personas que estuvieron presentes en mi vida y que posibilitaron que hoy mi sueño se hiciera realidad. Mil gracias.

Yaritza Montero Vaillant

Gracias a mi familia que me han apoyado en momentos difíciles y han sonreído conmigo en los felices. Gracias a mis padres por enseñarme a llorar y reír. Papá, mamá, palabras tan sencillas de pronunciar pero siempre enaltecen mi orgullo de hablar por la fortuna de ser hijo suyo. Que como un testimonio de cariño y eterno agradecimiento por mi existencia, valores morales y formación profesional, que sin escatimar esfuerzo alguno, han sacrificado gran parte de su vida para formarme y porque nunca podré pagar todos sus desvelos ni aún con las riquezas más grandes del mundo. Por lo que soy y por todo el tiempo que les robé pensando en mi… Gracias mamá, papá.

Le agradezco mucho a una persona muy especial que donde quiera que esté, mi familia y yo siempre estaremos en deuda y mientras quede vivo uno de nosotros estará vivo él. Gracias

Agradezco a todos mis profesores, desde mi enseñanza infantil hasta mis profesores de la carrera. A mis grandes amigos de mi tierra Odalis, Freddy, el gordo, Adrián que sin vernos mucho nunca hemos dejado de querernos. A mis amigos de la carrera que estoy seguro que nunca olvidaré, a la gente con la que conviví y convivo .En fin a todos aquellos que me importan y les importo gracias.

He llegado al final de este camino y en mi han quedado marcadas huellas profundas de éste recorrido. Son Madre tu mirada y tu aliento. Son Padre tu trabajo y esfuerzo. Son Maestros tus palabras y sabios consejos, mi trofeo es también vuestro.

Ernesto Miguel Rodríguez Rodríguez

Dedicatoria

Le dedico mi tesis a dos personas que son especiales en mi vida, son la razón de mí existir, mis dos grandes amigos: a mis padres Marta e Inmer.

Yaritza Montero Vaillant

Le dedico mi tesis a mis padres: Violeta y Miguel por ser ejemplo de sacrificio y dedicación, por ser especiales en mi vida.

Ernesto Miguel Rodríguez Rodríguez

Resumen

En la actualidad la demanda de software ha ido creciendo y con esta, han aumentado los problemas de seguridad en el código. En cuanto a seguridad se refiere, se hace necesaria la detección de errores en fases tempranas del ciclo de desarrollo del software, con el objetivo de detectar vulnerabilidades y asegurar la calidad del producto informático.

La Universidad de las Ciencias Informáticas es una institución productiva encargada de desarrollar aplicaciones en diferentes lenguajes de programación. Uno de sus objetivos principales es obtener productos de calidad. Para lograr esta meta se necesita un sistema que permita la revisión de código a través del análisis estático y así detectar las posibles vulnerabilidades que puedan existir.

El presente trabajo: Plataforma para el Análisis Estático de Vulnerabilidades del Código "Distribución y Ejecución de Herramientas", ofrece una arquitectura flexible, escalable y con rendimiento alto ante muchas peticiones, que permita analizar aplicaciones en varios lenguajes de programación mediante el "análisis estático". Además brinda la posibilidad de incluir otras herramientas de validación de código fuente, posibilitando la corrección del código en aplicaciones de diferentes lenguajes.

Una vez subido el fichero, el sistema debe seleccionar el servidor adecuado para ejecutar la herramienta que realizará el análisis pertinente, actualizar en tiempo real el estado de su análisis, brindar además los resultados de las vulnerabilidades obtenidas y permitir cancelar un determinado análisis en curso. Dicho sistema debe ser capaz de analizar tanto un proyecto como un script.

Palabras clave: análisis estático, arquitectura, calidad, seguridad, vulnerabilidades

Índice

Índice de Figuras

Índice de Tablas

Introducción

El software es cada vez más un componente común de la sociedad moderna y a su vez, se necesita medir de forma efectiva su calidad. La seguridad de las aplicaciones es una parte inherente en el proceso de desarrollo de software. Muchos de los problemas relacionados con la pérdida de calidad y seguridad en el software pueden atribuirse a la creciente complejidad del código.

Tradicionalmente, la única manera de encontrar vulnerabilidades en el código consistía en rastrearlo manualmente para encontrar posibles fuentes de error, convirtiéndolo en un proceso tedioso y propenso a errores. A medida que la demanda de software ha ido creciendo, lograr la seguridad de los mismos se hace cada vez más difícil, por lo que se hace necesario la detección de errores en fases tempranas del ciclo de desarrollo de software. Una de las técnicas que se utiliza con este fin es el análisis estático de código, técnica aplicada sobre el código fuente de la aplicación. Para esto se utilizan herramientas automatizadas que permitan el análisis y detección de vulnerabilidades, siendo esta una práctica exigida en la actualidad.

Hoy día existen en el mundo numerosas herramientas de análisis estático que mejoran este proceso, ya que ofrecen los medios para la detección automatizada de errores. Estas herramientas pueden ser propietarias o libres, las que de una forma u otra detectan los problemas de vulnerabilidad en las aplicaciones. Sin embargo, el hecho de que algunas sean propietarias supone un problema a la hora de obtenerlas debido al precio que exige su adquisición, además, presentan grandes restricciones al adquirir las licencias. Su funcionamiento es limitado, ya sea por el tiempo en que pueden ser usadas o por las pocas funcionalidades que brindan al usuario para su uso. Por su parte las que son libres, presentan como dificultad un incompleto desarrollo de software, ya sea por insuficiencia en la búsqueda de vulnerabilidades producto a la no actualización a versiones actuales o excesiva presencia de falsos positivos[1] o negativos[2] en las aplicaciones que analizan.

La Universidad de las Ciencias Informáticas (UCI), juega un primordial papel en el desarrollo de aplicaciones. Para ello, desde sus inicios comenzó a insertarse con fuerza en la industria del software. Esta institución cuenta con centros productivos, uno de estos es el “Centro de Telemática”, dentro de este

[1] ocurre cuando el analizador detecta vulnerabilidades que el programa o código no contiene.

[2] ocurre cuando el programa o código contiene errores que el analizador no ha detectado tras su análisis.

se encuentra el Laboratorio de Seguridad (LabSI), responsable de realizar las pruebas de seguridad a las diferentes aplicaciones desarrolladas en la UCI y detectar las vulnerabilidades existentes en el código. Para esto se desarrollaron sistemas de análisis estático en solo tres lenguajes de programación: PHP, Python y C++, las cuales cuentan con algunas dificultades de vital importancia, como son:

- Analizan el código en un solo lenguaje y no existe relación entre ellas.
- *Rendimiento bajo ante muchas peticiones*: se necesita optimizar y gestionar las cargas al servidor porque podría colapsar su trabajo ya que todo este proceso lo hace el servidor web y no uno especializado para ello.
- *No son flexibles, ni escalables*: estas aplicaciones no tienen la capacidad de adaptarse a otras circunstancias que pueden surgir a medida del desarrollo tecnológico o las necesidades que se tengan, tal es el caso que no permiten el análisis de código en otros lenguajes de programación, ni se puede agregar nuevos analizadores al sistema.

  Ejemplo: en la universidad se realizan aplicaciones en diferentes lenguajes (como son C#, C++, PHP y Java) siendo necesario realizar pruebas de seguridad al código. Actualmente no existe un sistema que permita encontrar vulnerabilidades en el código de las aplicaciones realizadas en Java o C#.

Teniendo en cuenta lo anteriormente planteado, se formula el siguiente **problema a resolver**: ¿Cómo mejorar los problemas de escalabilidad, flexibilidad y rendimiento ante muchas peticiones de la Plataforma para el Análisis Estático de Vulnerabilidades de Código?

A partir del problema planteado se define como **objeto de estudio**: los sistemas de control de análisis estático de vulnerabilidades de código.

Como **campo de acción:** arquitectura de las plataformas de análisis estático de vulnerabilidades de código.

Debido a la necesidad de darle solución a la problemática planteada se decidió cumplir con el siguiente **objetivo general**: desarrollar la arquitectura de la plataforma para el análisis estático de vulnerabilidades de códigos.

Objetivos específicos:

- Adaptar el middleware ICE a la plataforma de análisis estático de vulnerabilidades del código.
- Brindar flexibilidad en cuanto a la inclusión de nuevos analizadores al sistema.

Para cumplir con el objetivo propuesto se plantean las siguientes **tareas de investigación:**

- Estudio de los principales conceptos acerca del análisis estático de código para obtener los conocimientos necesarios del servicio que se brindará.
- Estudio de las herramientas existentes en el mundo que permiten el análisis estático de vulnerabilidades de código.
- Selección de las herramientas y tecnologías necesarias para el desarrollo del sistema.
- Definición de los patrones de arquitectura.
- Selección del middleware para lo comunicación entre aplicaciones.
- Estudio de los modelos o vistas fundamentales para la descripción de la arquitectura.

Se utilizaron los siguientes **métodos teóricos:**

- **Analítico-sintético:** en esta investigación se emplea este método para analizar las teorías, documentos y todo tipo de información que se tiene sobre el desarrollo del software; permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio y lograr así una mejor comprensión del mismo. También se utiliza para el análisis de los resultados más significativos que se han obtenido actualmente en ese campo.
- **Histórico-lógico:** a través de este método se logró un mayor entendimiento y conocimiento de las diferentes técnicas y herramientas de modelado a utilizar, dando la posibilidad de analizar la trayectoria histórica de los mismos, así como su evolución y desarrollo.
- **Modelación:** este método es de gran importancia para la investigación debido a que brinda la posibilidad de representar las propiedades y funcionalidades que tendrá el sistema a desarrollar.

Como **métodos empíricos** se utilizó:

- **Observación:** con este método se logró recopilar información acerca de los principales conceptos y características de los diferentes analizadores estáticos.

El documento está estructurado en 4 capítulos:

Capítulo 1 "Fundamentación Teórica": se incluye un estado del arte de la investigación, así como las metodologías, técnicas de programación y herramientas utilizadas para el desarrollo de la aplicación.

Capítulo 2 "Características del Sistema": se brinda una visión del sistema, se definen los requisitos funcionales y no funcionales a los que se debe dar cumplimiento y se ofrece además la propuesta del sistema a desarrollar.

Capítulo 3 "Diseño del Sistema": se realiza el diseño y descripción de la arquitectura del sistema, se presentan además los diagramas de clases y el diseño de la base de datos.

Capítulo 4 "Implementación y Pruebas del Sistema": se realizan los diagramas de despliegue y componentes como resultado de la implementación del sistema, además de las pruebas realizadas una vez concluido el desarrollo de la aplicación.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se realizará un análisis de los conceptos relacionados con el tema. Se presentará un estudio de las principales herramientas desarrolladas actualmente en el mundo, las cuales brindan seguridad en cuanto a la detección de vulnerabilidades existentes en códigos de diferentes lenguajes de programación. Además de seleccionar las metodologías, herramientas y lenguajes necesarios para el desarrollo del sistema.

1.2 Análisis estático

"Análisis estático es un tipo de análisis que se emplea sobre el código fuente de la aplicación sin necesidad de ejecutarla. Se puede realizar tanto sobre el código fuente, como sobre el código compilado" (1)

"El análisis estático de código es el proceso de evaluar el software sin ejecutarlo. Para ello recibirá el código fuente del programa, lo procesará intentando averiguar qué es lo que quiere que haga y dará sugerencias con las que se puede mejorar ese código." (2)

"El análisis estático del código se refiere al proceso de evaluación del código fuente sin ejecutarlo, en base a este análisis se obtendrá información que nos permita mejorar la línea base del proyecto, sin alterar la semántica original de la aplicación." (3)

"El análisis se realiza principalmente en base a patrones o reglas, mediante el análisis del flujo de los datos o métricas para detectar problemas de seguridad en el desarrollo." (4)

Después que se conocen algunas definiciones de análisis estático se puede decir que esta técnica es un tipo de análisis que emplea la revisión automatizada sobre el código fuente de una aplicación, sin necesidad de ejecutarla. De esta forma ayuda al desarrollador a descubrir posibles fuentes de error en el software y detectar las vulnerabilidades que aún no se han descubierto. Además, permite restringir el ámbito de investigación especificando donde se encuentra el posible error.

El análisis estático en busca de vulnerabilidades de código se encuentra dentro de las buenas prácticas de software, ya que a través de este se analiza a fondo y se asegura que se detecten de forma rápida, eficiente y automatizada las posibles fuentes de error. Este tipo de análisis cuenta con numerosas

ventajas, convirtiéndolo en una de las armas más poderosas en el arsenal del desarrollador de software, como son:

- Permite revisar grandes aplicaciones en poco tiempo.
- Permite encontrar errores en el desarrollo de las aplicaciones de forma temprana en el ciclo de desarrollo del software, reduciendo así el costo.
- Es exhaustivo ya que revisa todas las posibles entradas/salidas de una aplicación.
- Se encuentran en su localización exacta, los problemas del código.
- Se identifica la causa raíz del problema.
- Se puede verificar rápidamente la corrección.

1.3 Herramientas de código estático

Las herramientas de análisis de código estático son herramientas automatizadas que examinan el código fuente de una aplicación, en busca de vulnerabilidades. Se utilizan durante la fase de desarrollo del software con el fin de identificar las inseguridades existentes en fase temprana, resultando menos costoso a la hora de realizar los cambios de los errores detectados. A lo largo de los años se han desarrollado varias herramientas para realizar el análisis estático de código, con el objetivo de encontrar posibles errores. Su uso ayuda a mejorar la calidad de la implementación y reducir la cantidad de problemas que pueda tener el software.

1.3.1 Herramientas de código estático existentes en el mundo

A continuación se realizará un estudio de algunas herramientas existentes en el mundo que realizan auditoría de seguridad sobre el código fuente de una aplicación. Estas herramientas pueden ser propietarias o libres, las que permiten realizar el análisis con el objetivo de detectar las posibles fuentes de error existentes en códigos de diferentes lenguajes de programación.

1.3.1.1 Herramientas propietarias

Las herramientas propietarias o privativas tienen limitadas las posibilidades de que los usuarios la usen, modifiquen o redistribuyan. Mantienen, además, el código fuente en secreto. En (5) se exponen algunas ventajas de estas herramientas, las cuales se presentan a continuación:

- Mejor acabado en la mayoría de las herramientas, ya que los usuarios deben pagar por el producto adquirido.
- Presentan personal altamente capacitado para el desarrollo de las herramientas.

1.3.1.1.1 Code Secure

Code Secure es un analizador multilenguaje que detecta vulnerabilidades en ASP.NET, VB.NET, C#, Java/J2EE, JSP, EJB, PHP, Classic ASP y VBScript. Forma parte de un conjunto de aplicaciones de seguridad de Armorize Appsec Suite. Esta modela el comportamiento de una aplicación Web y tracea el flujo de datos de cada entrada vulnerable de un fichero. Además, brinda reportes detallados con estadísticas y gráficos vinculados del análisis realizado. (4)

1.3.1.1.2 Fortify

Analizador multilenguaje que identifica la causa raíz de las vulnerabilidades tanto en el código fuente como en el código en ejecución mediante la detección de más de 400 tipos de vulnerabilidades de 17 lenguajes de programación. Algunos de los lenguajes en que analiza son: C / C + +, NET, Java, JSP, ASP.NET, ColdFusion, "Classic" ASP, PHP, Visual Basic 6, VBScript, JavaScript, PL / SQL, T-SQL, Python. Las vulnerabilidades pueden recabarse durante la fase de desarrollo o la fase de control de la calidad de un proyecto, o incluso después de que una aplicación haya entrado en producción, lo que minimiza el riesgo de que no se haya detectado un problema grave. Fortify prioriza los resultados de sus analizadores para ofrecer una lista de problemas, precisa y ordenada por nivel de riesgo y de esta forma los problemas más graves se resuelven primero.

Con Fortify los desarrolladores aprenden prácticas de codificación segura, mientras resuelven las vulnerabilidades. Por cada vulnerabilidad esta herramienta proporciona información referenciada, en la que se describe el problema y la forma de resolverlo en el lenguaje de programación específico del usuario. (6)

1.3.1.1.3 Coverity Static Analysis

Anteriormente se le llamaba “Coverity Prevent”. Este analizador identifica vulnerabilidades de código en diferentes lenguajes como son: C, C++, C# y el código java. Proporciona una comprensión completa de su entorno de compilación, el código fuente y el proceso de desarrollo. Esta herramienta es altamente escalable y posee una baja tasa de falsos positivos. (7)

1.3.1.2 Herramientas libres

Las herramientas libres son aquellas que permiten ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el código de acuerdo a las necesidades del usuario. En (5) se exponen algunas ventajas de estas herramientas, las cuales se citan a continuación:

- Pueden ser usadas por cualquier persona sin costo alguno.
- Se pueden hacer modificaciones al código y adaptarlo de acuerdo a las necesidades.
- Tienden a ser eficientes ya que muchas personas lo mejoran y optimizan.
- Son de libre distribución.
- El acceso al código fuente permite el desarrollo de nuevos productos sin la necesidad de desarrollar todo el proceso partiendo de cero.

1.3.1.2.1 Rough Auditing Tool for Security (RATS)

RATS es una herramienta de seguridad desarrollada por los ingenieros del "Secure Software", que actualmente pertenece a "Fortify Software". Es una herramienta para escanear el código fuente de las aplicaciones escritas en C, C++, PHP y Python con el fin de encontrar errores de seguridad. Utiliza una base de datos para encontrar sentencias del código que pueden ser una vulnerabilidad potencial. El usuario puede decidir qué base de datos utilizar en el escaneo y que nivel de riesgo asumir.

Por cada problema potencial encontrado, RATS provee la descripción del mismo y sugiere una solución al respecto. Además, indica el grado de severidad del problema encontrado para facilitarle al auditor la asignación de prioridades y ejecuta un análisis básico para descartar condiciones que no causan problemas de seguridad. El análisis con RATS no es completo ya que no encuentra todos los errores existentes y puede destacar como errores cosas que no lo son. (1)

1.3.1.2.2 Yet Another Source Code Analyzer (YASCA)

Es una herramienta de análisis multilenguaje creada por Michael Scovetta, la cual está escrita en PHP y con licencia BSD. El funcionamiento de la herramienta es mediante el uso de plugins, recopilando analizadores de códigos libres disponibles no solo de seguridad, también escáneres de sintaxis de código. Algunos de estos analizadores son FindBugs, JLint, PMD y Pixy, en los mismos no se realizan cambios ni innovaciones. Esta herramienta es capaz de escanear código de C++, Java, JavaScript, ASP, PHP, HTML/CSS, ColdFusion, Cobol, entre otros. Puede integrarse fácilmente con otras herramientas, tales como RATS. La filosofía de YASCA se localiza en brindar un conjunto de herramientas a los desarrolladores para que su trabajo sea mejor en cuanto a la seguridad y métricas del software. Está diseñado para ser flexible y fácil de extender. (8)

1.3.2 Plataforma de análisis estático desarrollada en la UCI

En la UCI se realizaron sistemas para el análisis estático de código, en tres lenguajes de programación: PHP, Python y C++. Estos sistemas no se lograron integrar en la plataforma, analizan el código en un solo lenguaje y no existe relación entre ellos.

1.3.2.1 Sistema de análisis estático de vulnerabilidades. Módulo PHP

El sistema de análisis estático de vulnerabilidades "módulo PHP", es una herramienta de validación que permite la revisión de código a los desarrolladores y la comunidad universitaria, pero solo en lenguaje PHP. Este sistema brinda la posibilidad de analizar tanto un proyecto como un script, además muestra los reportes de forma detallada y las estadísticas de los análisis realizados. En la implementación del software se aprovecharon las ventajas de los ambientes en la Web, con el motor de búsqueda de vulnerabilidades del analizador Pixy. (4)

1.3.2.2 Sistema de análisis estático de vulnerabilidades. Módulo Python

El sistema de análisis estático de vulnerabilidades "módulo Python", es un sistema de validación para la revisión de código en este lenguaje. Presenta un conjunto de funcionalidades básicas que le permiten al usuario realizar el análisis de un fichero o proyecto que ha sido subido. Además, permite revisar los reportes generados de las vulnerabilidades detectadas, dando la posibilidad de eliminar el que desee. Brinda un conjunto de estadísticas referentes a las revisiones realizadas, donde se puede observar la calidad de los proyectos analizados y por último, permite cancelar un análisis que se encuentre en curso. Se utilizaron los analizadores Pyntch y Pylint con el propósito de realizar un mejor análisis. (9)

1.3.2.3 Sistema de análisis estático de vulnerabilidades. Módulo C++

El sistema de análisis estático de vulnerabilidades, módulo C++, provee a todos los programadores y usuarios de la UCI una herramienta que facilita la detección de vulnerabilidades, logrando que el código esté guiado hacia una programación segura. Es un sistema basado en la Web, al cual se integra una herramienta de análisis estático de código (Cppcheck), encargada de la detección de errores escritos en lenguaje C++. El sistema brinda la posibilidad de analizar tanto un proyecto como un sencillo script y ofrecer los reportes de las vulnerabilidades detectadas, de manera detalla y en un formato entendible. La aplicación posibilita además, la generación de reportes estadísticos de los resultados de los análisis realizados en sentido general. (10)

Después del estudio y análisis de las herramientas de código estático se decide desarrollar una arquitectura basada en la arquitectura de YASCA, ya que su funcionamiento es mediante el uso de plugins, permitiendo incorporar nuevos analizadores libres al sistema, de esta forma se lograría abarcar gran cantidad de lenguajes de programación así como reglas específicas de estos. Se evitaría además, la necesidad de implementar analizadores, encargados de realizar los análisis para los diferentes lenguajes que existen en el mundo; a diferencia de las demás herramientas que solo analizan códigos en los lenguajes que tienen implementado, es decir, no utilizan analizadores externos a su aplicación.

1.4 Metodologías de desarrollo de software

Las metodologías de desarrollo del software son un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. (11) Su principal objetivo es aumentar la calidad del software que se produce, en todas y cada una de sus fases de desarrollo.

En la actualidad, constituyen uno de los temas más polémicos en el mundo del desarrollo de aplicaciones. Su uso persigue, como objetivo principal, lograr el éxito rotundo de los proyectos de producción de software, para lo cual imponen un proceso disciplinado con el fin de hacerlo más predecible y eficiente. Se debe seleccionar la metodología más adecuada que posibilite obtener los resultados óptimos que se desean, es por esto que se necesita conocer las principales características de las diferentes metodologías y así poder elegir la correcta.

1.4.1 Proceso unificado de desarrollo (RUP)

La metodología a utilizar para el análisis y modelado de los procesos de desarrollo del software es RUP[3], ya que se caracteriza por ser iterativo e incremental, de esta forma, en cada iteración se puede llevar al cliente una versión del producto, con el fin de obtener mejoras para el software, alcanzando así la mejor calidad durante el ciclo de vida del proyecto; está centrado en la arquitectura, permitiendo tener una visión completa del mismo y guiado por casos de usos, lo cual constituye el hilo conductor de todo proceso.

Otra de las facilidades por lo cual se decide utilizar RUP es porque constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientado a objetos, permitiendo adaptarse a las necesidades de cualquier proyecto. Dicha metodología cuenta con una amplia documentación, lo que facilita que el producto contenga manuales para su correcto funcionamiento.

[3] Proceso unificado racional (en inglés Rational Unified Process).

Incluye además artefactos[4] y roles[5] que son diseñados durante las diferentes fases del desarrollo de software.

1.5 Lenguaje de modelado.

En todas las disciplinas de la Ingeniería se hace evidente la importancia del modelado, no solamente sirve para los grandes sistemas, aún en aplicaciones de pequeño tamaño se obtienen beneficios de modelado. Sin embargo es un hecho que entre más grande y complejo es el sistema, más importante es el papel que juega.

1.5.1 Lenguaje unificado de modelado (UML).

UML es un lenguaje gráfico que permite visualizar, especificar, construir y documentar un sistema. Está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo. El proceso unificado RUP utiliza UML como lenguaje de modelado , para preparar todos los esquemas de un sistema de software. (12) A través de UML es posible establecer los requerimientos y estructuras necesarias para un sistema de software, antes del código.

En la presente investigación se decidió utilizar UML como lenguaje de modelado ya que es la clave para generar todo la documentación necesaria, ayudando así a crear con facilidad los manuales de usuario. Además, permite que en futuras versiones del desarrollo de la aplicación los desarrolladores comprendan mejor todo el proceso.

1.6 Plataforma de desarrollo

1.6.1 Herramienta CASE

Las herramientas CASE son diversas son diversas aplicaciones destinadas a aumentar la productividad en el desarrollo del software, reduciendo el coste de las mismas en términos de tiempo y dinero. Evitan la aparición de errores permitiendo obtener una idea clara de que es lo que se va a hacer antes de comenzar a programar. Estas herramientas tienen vital importancia en múltiples aspectos del ciclo de vida del desarrollo del software, como son:

[4] Productos tangibles del proceso, ejemplo: modelo de casos de uso, código fuente, etc.

[5] Papel que desempeña una persona en un determinado momento.

- Mejorar la productividad en el mantenimiento y desarrollo del software.
- Mejorar el tiempo, costo de desarrollo y mantenimiento de los sistemas informáticos.
- Implementación de parte del código con el diseño dado.
- Aumentar la calidad del software.
- Documentación o detección de errores.

En el proceso de desarrollo de software cumplir con la planificación y ofrecer un producto de calidad, son dos retos para el equipo de desarrollo. La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Con el objetivo de apoyar el trabajo de los desarrolladores durante el ciclo de vida y desarrollo del proyecto se hace necesario seleccionar una herramienta CASE que brinde soluciones para resolver los problemas relacionados con el software. Estas herramientas fueron desarrolladas para automatizar los procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados.

1.6.1.1 Visual Paradigm

Visual Paradigm es una herramienta CASE que facilita el modelado de artefactos en el proceso de desarrollo de software mediante el lenguaje de modelado UML, soportando el ciclo de vida completo del desarrollo del software. Ofrece la posibilidad de documentar todo el trabajo sin necesidad de emplear otras herramientas en varios formatos, como PDF u otras. Ayuda a la construcción de aplicaciones de calidad y a un excelente costo. Permite además diseñar los diferentes tipos de diagramas de clases y generar códigos a partir de dichos diagramas. Soporta ingeniería inversa, importa proyectos de Rational Rose, genera informes, edita detalles de casos de uso y genera bases de datos, permitiendo la transformación de diagramas de entidad-relación. (13) Teniendo en cuenta las múltiples características antes mencionadas, esta herramienta es seleccionada para el modelado de la aplicación.

1.6.2 Lenguaje de programación.

Un lenguaje de programación es un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura, así como el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina. Actualmente existen lenguajes de programación de bajo, mediano y alto nivel. En el caso de los lenguajes de bajo nivel, las instrucciones son muy simples y cercanas al funcionamiento de la máquina, como lo son el código máquina y el ensamblador. Los de alto nivel están caracterizados por un alto nivel de abstracción y utilizan un lenguaje más cercano al natural,

ejemplo de ellos son: C++, C#, java, python, entre otros. Los lenguajes de programación de nivel medio, como su nombre lo dice, se localizan en un punto medio entre los dos anteriores. Dentro de estos lenguajes podría estar C. (14)

1.6.2.1 Python

Python es un lenguaje de alto nivel, multiplataforma, portable y sencillo de aprender, ofreciendo así una sintaxis extraordinariamente simple. Es un lenguaje libre y de fuente abierta lo que permite distribuir libremente copias de este software, leer su código fuente, hacerle cambios y usar partes del mismo en nuevos programas libres. (15)

Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, ofreciendo así ventajas como la rapidez de desarrollo. Ofrece además una manera poderosa y simple de emplear programación orientada a objetos. La biblioteca estándar de Python es amplia, conteniendo funcionalidades de gran ayuda y facilitando a la vez al programador la implementación de aplicaciones sin la necesidad de recurrir continuamente a bibliotecas externas. Puede ayudar a hacer varias cosas que involucran: expresiones regulares, generación de documentos, pruebas, procesos, bases de datos, navegadores web, ftp, correo electrónico, XML, HTML, etc. Python dispone además de una extensa colección de bibliotecas libres, disponibles en casi todos los repositorios GNU\Linux. (15)

1.6.3 Entorno de desarrollo integrado.

Un IDE[6] es un entorno de programación creado con un conjunto de herramientas para el programador: editor de código, compilador, depurador y constructor de interfaz gráfica. Estos pueden ser aplicaciones independientes o pueden formar parte de aplicaciones existentes, para un lenguaje de programación específico o multilenguaje. (16) Proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación.

1.6.3.1 Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto, multiplataforma, soportado por los principales sistemas operativos, se utiliza para integrar herramientas de desarrollo, con una arquitectura abierta y basada en plugins, tiene una comunidad de desarrolladores verdaderamente activa. Continuamente se están revisando los plugins anteriores y desarrollando nuevos. Permite además, la

[6] Entorno de Desarrollo Integrado

instalación de plugins destinados a mejorar las funcionalidades del propio IDE y a extenderse en más tecnologías.

1.6.4 Gestor de base de datos

Un sistema gestor de bases de datos (SGBD) es un sistema computarizado que permite almacenar información y ofrece la posibilidad a los usuarios de recuperar y actualizar los datos. Su objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos. (17)

Un SGBD debe permitir:

- *Definir una base de datos*: especificar tipos, estructuras y restricciones de datos.
- *Construir la base de datos*: guardar los datos en algún medio controlado por el mismo SGBD.
- *Manipular la base de datos*: realizar consultas, actualizarla y generar informes.

Algunos ejemplos de SGBD son Oracle, DB2, PostgreSQL, MySQL, MS SQL Server, etc.

1.6.4.1 PostgreSQL

PostgreSQL es un sistema de base de datos multiplataforma, potente, dirigido por una comunidad de desarrollo, es orientado a objetos, libre y está bajo la licencia BSD. Posee interfaces gráficas de usuarios y enlazadores para algunos lenguajes de programación. (18) Permite además una alta concurrencia, debido fundamentalmente a que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Es capaz de ajustarse al número de CPU y a la cantidad de memoria que posee el sistema de forma óptima, lo cual permite aceptar muchas peticiones simultáneas al mismo tiempo.

Una de las principales necesidades del sistema es la utilización de un gestor de base de datos potente, que permita el almacenamiento de gran cantidad de información, siendo esta una de las razones por la que fue seleccionado PostgreSQL. Existen otros gestores con esta característica como es el caso de Oracle, pero fue seleccionado PostgreSQL por ser un software libre, aspecto fundamental.

1.6.5 ORM

Un ORM[7] es un componente de software que permite trabajar con los datos persistidos como si fueran parte de una base de datos orientada a objetos. (19) Es una técnica de programación que permite convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional.

Utilizar un ORM tiene una serie de ventajas, a continuación se citan algunas que se exponen en (20):

- *Reutilización*: es la principal ventaja que aporta un ORM ya que permite llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- *Encapsulación*: la capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
- *Portabilidad*: utiliza una sintaxis propia del ORM siendo capaz de traducir a diferentes tipos de bases de datos.
- *Seguridad*: los ORM suelen implementar mecanismos de seguridad que protegen la aplicación de los ataques más comunes como SQL Injections.
- *Mantenimiento del código*: gracias a la correcta ordenación de la capa de datos, modificar y mantener el código es una tarea sencilla.

1.6.5.1 SQLAlchemy

SQLAlchemy es un ORM lanzado bajo la licencia MIT. Está diseñado para bases de datos robustas.

Posee dos niveles:

1. *Nivel ORM*: trabaja a nivel de clases y objetos (ideal para despreocuparse completamente de la existencia de una base de datos).
2. *Nivel SQLExpression*: permite trabajar a un nivel más bajo, manipulando tablas y usando un lenguaje muy cercano a SQL.

[7] Mapeo Objeto-Relacional

Estas características posibilitan comenzar trabajando a alto nivel con ORM y emplear SQLExpression para consultas delicadas. SQLAlchemy brinda numerosas ventajas, como son:

- Aísla completamente el código de la base de datos que se emplea.
- Permite emplear tipos genéricos de bases de datos (aquellos definidos por los estándares) o tipos específicos de alguna base de datos.
- El sistema de consultas es extremadamente potente.
- El diseño de SQLAlchemy permite obviar la tarea de interactuar con la base de datos, permitiendo llevar a cabo tareas complejas.

1.6.6 Protocolo de comunicación.

1.6.6.1 Protocolo XMPP

XMPP[8] es un protocolo abierto y extensible basado en XML, originalmente ideado para mensajería instantánea. Es usado además en una amplia gama de aplicaciones de mensajería de voz y video. Con el protocolo XMPP queda establecida una plataforma para el intercambio de datos XML que puede ser usada en aplicaciones de mensajería instantánea. (21)

En (22) se exponen algunas características de este protocolo, las cuales se citan a continuación:

- El protocolo es gratuito, abierto, público y de fácil comprensión. Es por ello que cuenta con múltiples implementaciones entre clientes, servidores, componentes de servidores y librerías de código.
- *Extensible*: se pueden construir funcionalidades personalizadas sobre el núcleo del protocolo.
- *Flexible*: las aplicaciones originales de XMPP (de mensajería y presencia) se han extendido y ahora pueden encontrarse en administración de redes, sindicalización de contenidos, herramientas de colaboración, compartimiento de archivos, juegos, monitoreo de sistemas remotos, servicios web, etc.

[8] Protocolo Extensible de mensajería y comunicación de presencia (en inglés Extensible Messaging and Presence Protocol).

1.6.7 ICE

ICE[9] es un middleware[10] orientado a objetos, es decir, proporciona herramientas APIs y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos. Su código fuente puede portarse de manera independiente al entorno de desarrollo. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación, pueden ejecutarse en distintos sistemas operativos y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red. (23)

ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas, proporcionando grandes beneficios, como son:

- Proporciona un manejo síncrono y asíncrono de mensajes.
- Soporta múltiples interfaces.
- Es independiente de la máquina.
- Es independiente de la implementación (el cliente no necesita conocer como el servidor implementa sus objetos).
- Es independiente del sistema operativo.
- Soporta el manejo de hilos.
- Es independiente del protocolo de transporte empleado.
- Permite hacer persistentes las aplicaciones.
- Tiene disponible el código fuente.
- Permite replicación (un mismo servidor se puede encontrar en varias computadoras mediante adaptadores).

[9] Internet Communication Engine.

[10] Software de conectividad que hace posible que aplicaciones distribuidas pueden ejecutarse sobre distintas plataformas heterogéneas.

1.6.7 VSFTPD

El VSFTPD[11] es un servidor FTP para sistemas Unix incluido Linux. Es seguro, rápido y está licenciado por medio de la licencia GPL. Se distingue principalmente por su sencillez en la configuración.

Algunas de las características principales son:

- Configuración de direcciones IP virtuales.
- Usuarios virtuales.
- Operación por medio de Inetd o Standalone.
- Poderosa configuración por usuario.
- Control de ancho de banda.
- Límite de conexiones IP por usuario.
- Soporte de encriptación a través de SSL.

1.7 Conclusiones Parciales

En este capítulo se realizó una descripción panorámica general de algunas herramientas existentes en el mundo, las cuales realizan análisis estático en diferentes lenguajes de programación, de esta forma se detectan las posibles vulnerabilidades que pueda tener un código. Se fundamentan los principales conceptos relacionados con el tema para lograr una mejor comprensión de la problemática. Quedó reflejado además el estudio y análisis de la selección de las herramientas, metodologías y lenguajes necesarios para el desarrollo de la implementación de la plataforma.

[11] Very Secure FTP Daemon

Capítulo 2: Características del Sistema

2.1 Introducción

En el presente capítulo se describen las características del sistema a desarrollar, se expone el resultado del proceso de captura de requisitos funcionales y no funcionales a los que se debe dar cumplimiento. Se identifican los actores que intervienen en el sistema y se modela el diagrama de casos de uso. Finalmente, se describirán todos los casos de uso para una mejor comprensión de las funcionalidades de la plataforma.

2.2 Objeto de estudio

2.2.1 Problema y situación problemática

La UCI es un centro productivo de software de importancia para el país, en el que se desarrollan aplicaciones para detectar los errores que pueda tener un código. En la actualidad existen en este centro algunos sistemas que realizan estas pruebas de seguridad, con el objetivo de detectar las vulnerabilidades. Dichos sistemas no se encuentran relacionados, analizan el código en un solo lenguaje de programación: Python, PHP o C++, lo que dificulta a la hora de analizar el código en otros lenguajes. Además, al existir muchas peticiones se necesitaba optimizar las cargas al servidor porque podría colapsar el trabajo.

2.2.2 Objeto de automatización

La problemática expuesta anteriormente permite confirmar la necesidad de desarrollar una arquitectura capaz de mejorar los problemas de flexibilidad, escalabilidad y rendimiento ante muchas peticiones de los sistemas desarrollados en la UCI, brindando la posibilidad de incluir otras herramientas que permitan el análisis en varios lenguajes de programación. Una vez subido el fichero a analizar, el sistema debe seleccionar la herramienta adecuada para realizar el análisis pertinente, actualizar en tiempo real el estado de su análisis y brindar los resultados de las vulnerabilidades obtenidas.

2.2.3 Información que se maneja

Se maneja los scripts y la información pertinente a los proyectos interesados en la revisión de código fuente, además de los resultados de las vulnerabilidades encontradas a través del análisis realizado mediante las herramientas de revisión de código estático.

2.3 Propuesta del sistema

Debido a la necesidad de solucionar el problema existente, se propone realizar una arquitectura genérica que permita la revisión de código en cualquier lenguaje de programación que posea un analizador de código, con el objetivo de detectar las vulnerabilidades existentes. Dicho sistema debe ser capaz de analizar tanto un proyecto (aplicación completa) como un script, determinar la herramienta y servidor más adecuado para realizar el análisis y ofrecer además los resultados de las vulnerabilidades encontradas tras el análisis pertinente.

2.4 Modelo del negocio

Para modelar el negocio se utilizó el modelo de dominio, ya que no se definen concretamente los procesos del mismo, se enfoca al uso de herramientas y tecnologías informáticas, por lo que se decide mostrar los principales conceptos y objetos que se manejan en el dominio de la aplicación. Este modelo contribuye posteriormente a identificar algunas clases que se utilizarán en el sistema, permitiendo comprender el contexto en que se enmarca.

MODELO DE DOMINIO

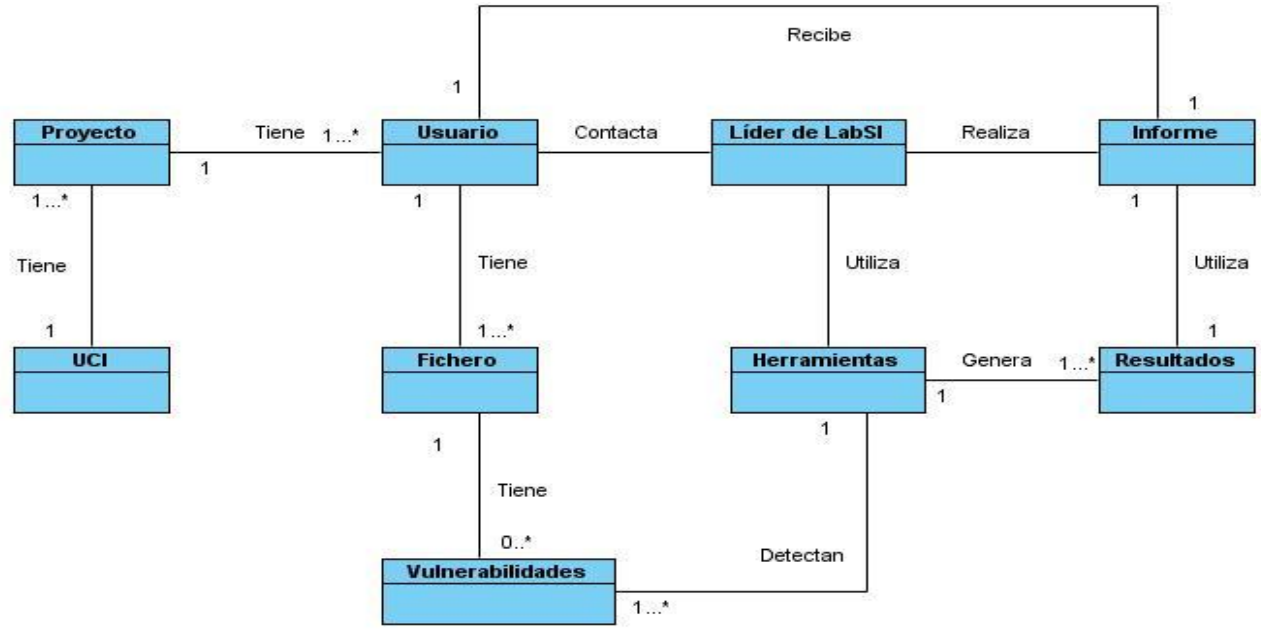

Figura 1: Modelo de dominio

Definiciones

La definición de cada uno de los conceptos del modelo de dominio se muestra a continuación:

UCI: representación de varios proyectos que necesitan analizar un código.

Proyecto: grupo de personas que desarrollan software.

Usuario: persona que necesita un análisis de calidad en cuanto a seguridad de código.

Fichero: representa un archivo que posee un código, el cual será analizado y puede presentar determinadas vulnerabilidades.

Vulnerabilidades: problema de seguridad en el código que se analiza.

Líder de LabSI: especialista del laboratorio de seguridad informática que realiza el análisis del código.

Herramientas: software para la detección de vulnerabilidades.

Informe: informe realizado por la plataforma SAEV luego del análisis de ficheros XML generado por las herramientas.

Resultados: archivo generado por las herramientas con la descripción de las vulnerabilidades analizadas.

2.5 Especificación de los requisitos de software

El proceso de captura de requisitos tiene gran importancia en el proceso de desarrollo del software, ya que a través de estos se identifica lo que el usuario desea y de esta forma se obtiene un producto de calidad.

Los requisitos se pueden clasificar en:

- Funcionales.
- No Funcionales.

2.5.1 Requisitos funcionales

Los requisitos funcionales no son más que capacidades o condiciones que el sistema debe cumplir. (24) Especifican comportamientos particulares de un sistema. A continuación se presenta el listado de los requisitos funcionales en el sistema que se modela:

RF1. Recibir notificación de nueva solicitud de análisis.

Consiste en recibir la notificación de la solicitud de análisis realizada por el front end (aplicación externa) para detectar las vulnerabilidades de un fichero.

RF2. Verificar disponibilidad de herramientas en servidores.

Posibilita buscar si existen herramientas disponibles en los servidores que puedan analizar un determinado fichero.

RF3. Almacenar en la base de datos la solicitud de análisis pendiente.

Requisito que permite que el sistema almacene en la base de datos la solicitud de análisis con estado de pendiente, hasta que exista un servidor que pueda atender la solicitud.

RF4. Obtener datos de los servidores de la base de datos.

Consiste en obtener los datos de CPU y memoria (real y en uso) de los servidores.

RF5. Obtener servidor óptimo.

Determina cuál es el servidor más adecuado para atender la petición de análisis.

RF6. Notificar análisis de código.

Consiste en notificar al servidor seleccionado para atender la solicitud, que existe un código para analizar.

RF7. Obtener datos del fichero a analizar en la base de datos.

Permite que el servidor obtenga la información del fichero que la herramienta tiene que analizar.

RF8. Descargar fichero desde FTP.

Posibilita que el servidor descargue del FTP el fichero que la herramienta va a analizar.

RF9. Solicitar análisis del código a la herramienta.

Requisito que permite que el servidor le solicite a la herramienta analizar el código descargado.

RF10. Actualizar datos del funcionamiento en la base de datos.

Posibilita que los servidores actualicen en la base de datos el porciento de CPU y memoria que están siendo utilizados por las herramientas. Esta actualización se realiza cuando la herramienta comience a analizar el código y una vez que termine.

RF11. Cargar resultado del análisis al FTP.

Permite subir al FTP las vulnerabilidades obtenidas después del análisis de los ficheros.

RF12: Actualizar estado de herramienta.

Las herramientas que permiten el análisis del código se pueden encontrar en dos estados: ocupado (cuando la herramienta se encuentra en ejecución) o desocupado. Este requisito posibilita la actualización en la base de datos del estado de las herramientas.

RF13. Notificar resultado de análisis.

Permite notificar fin de análisis una vez que las herramientas hayan terminado de analizar el código.

RF14. Cancelar análisis en curso.

Posibilita cancelar un análisis que se encuentra en curso.

2.5.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. (24) Representan características del sistema que señalan restricciones haciendo el producto atractivo, rápido o confiable.

A continuación se presentan los requisitos no funcionales del sistema:

Requisitos de software

RNF1. Se necesita la instalación del sistema operativo Linux, Ubuntu versión 10.04 o superior.

RNF2. Se requiere de un servidor jabber para realizar la actualización en tiempo real del estado de la herramienta.

RNF3. El servidor de base de datos es PostgreSQL 8.3 o superior.

RNF4. Se necesita un servidor FTP para guardar los ficheros que se desean analizar y el resultado de las vulnerabilidades encontradas.

Requisitos de hardware

RNF5. Se necesita que los servidores python, base de datos y FTP se encuentren en máquinas de 1GB o más de memoria, con 10 GB de espacio libre en disco duro como mínimo y procesador Intel Dual Core, AMD Athlon X2 o superior.

Requisitos en el diseño y en la implementación

RNF6. El lenguaje de programación a utilizar es python.

RNF7. La herramienta de desarrollo a utilizar es eclipse.

Requisitos de seguridad

RNF8. Los servidores con los que cuenta la plataforma para su correcta ejecución deben estar en una red interna, en la cual el único que tendrá salida al exterior será el front end.

RNF9. La información que viaja por la red a través del middleware ICE es protegida por la funcionalidad Glacier 2, mediante mecanismos de encriptación.

RNF10. La información que viaja por la red para el acceso a datos es protegida por el ORM SQLAlchemy.

Integridad

RNF11. Debe realizarse salvas de seguridad de la base de datos para la recuperación en caso de fallas.

RNF12. La información manejada por el sistema será objeto de protección.

Disponibilidad

RNF13. El sistema debe de estar disponible cuando se envíen las solicitudes de análisis.

Confiabilidad

RNF 14. Los análisis serán completados con la más alta prioridad y a pesar de la caída del sistema.

Requisitos de Rendimiento

RNF15. La aplicación requiere de un procesamiento rápido de la información, debido al análisis que realizan los analizadores. El tiempo de respuesta del sistema está en dependencia de la cantidad de archivos a analizar, este valor puede variar en dependencia de la cantidad de ficheros que se examinen.

Requisitos de soporte

RNF16. El sistema debe contar con un “Manual de Usuario” para evitar problemas en caso de fallas.

Aspectos Legales

RNF17. La aplicación y la documentación pertenecen al centro de Telemática de la facultad 2 de la Universidad de las Ciencias Informáticas.

RNF18. Las licencias están garantizadas ya que las herramientas son libres.

2.6 Definición de los casos de uso

2.6.1 Definición de los actores

Los actores no son más que agentes externo al sistema, los cuales brindan la información necesaria para ejecutar los casos de uso (CU) y a la vez se benefician de estos. A continuación se especifican los actores que interactúan con el módulo “Distribución y Ejecución de Herramientas de la Plataforma de Análisis Estático de Vulnerabilidades del Código”.

Actores	Justificación
Front_End	Sistema encargado de enviar el código hacia el servidor FTP para realizar el análisis estático, así como guardar todos los datos del fichero en la base de datos.
Solicitud_Pendiente	Actor que se origina como parte del evento interno del sistema, “Solicitudes Pendientes”.
Back_End	Sistema encargado de analizar el código de un fichero.

Tabla 1: Definición de actores del sistema

2.6.2 Diagrama de casos de uso (DCU)

El DCU es un modelo que especifica las funcionalidades del sistema en términos de CU y sus relaciones con los actores. A continuación se muestra el diagrama de casos de uso para el desarrollo del sistema.

DIAGRAMA DE CASOS DE USO

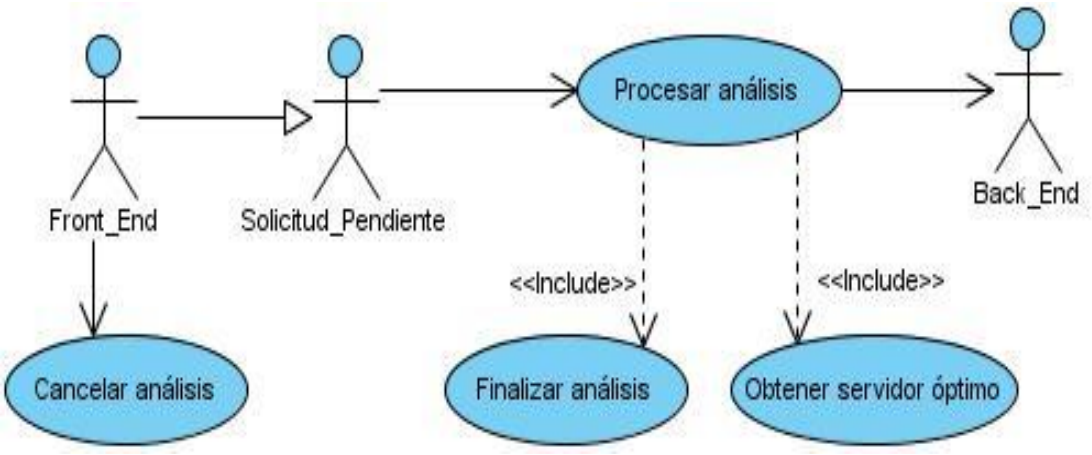

Figura 2: Diagrama de casos de uso

2.6.3 Casos de uso expandido

A través de la expansión de los casos de uso se realiza una descripción detallada de la secuencia de eventos que se ejecutan dentro del caso de uso, en colaboración con sus actores correspondientes. De ésta forma en el sistema se podrán implementar todas las funcionalidades necesarias para su funcionamiento. Para lograr un entendimiento de la descripción de los casos de uso se necesita conocer algunos términos relacionados con esta:

Definiciones:

- *Servidor FTP:* sistema encargado de la gestión de información, permite copiar y descargar el código a analizar, así como el resultado de las vulnerabilidades.
- *Servidor de base de datos (BD)*: sistema gestor encargado de guardar todos los datos necesarios para el desarrollo del sistema, además permite guardar los datos de los ficheros pendientes que no pueden ser analizados.
- *Servidor jabber*: sistema encargado de permitir la comunicación en tiempo real entre los servidores python y el front end para el envío de mensajes.
- *Servidor python*: servidores encargados de mandar a ejecutar los analizadores para que realicen el análisis de un código.

A continuación se presentan las descripciones de los casos de uso del sistema.

Caso de Uso	
CU-1	Procesar análisis
Propósito	Procesar las peticiones de análisis para determinar las herramientas que pueden analizar un código.
Actores	Solicitante (inicia)
Resumen	El caso de uso se inicia cuando el Front_End le envía una solicitud al sistema para analizar el código que se encuentra en el servidor FTP o cuando el sistema verifica que existen solicitudes pendientes por analizar. Finaliza una vez que el sistema le envíe a los servidores python los datos del fichero a analizar y la herramienta que le corresponde.

Referencias	RF1, RF2, RF3, RF6
Precondiciones	Los servidores tienen que estar en funcionamiento.

Acción del Actor	**Respuesta del Sistema**
1. El Front_End envía la solicitud de analizar código con los siguientes datos: • *id_fichero*: indica el identificador del fichero a analizar. • *nombre_herramienta*: nombre de la herramienta que va a analizar el fichero.	2. El sistema recibe la notificación de la nueva solicitud de análisis.
	3. El sistema se conecta a la base de datos.
	4. Guarda en la base de datos la solicitud.
	5. Verifica en la BD si existen herramientas disponibles para analizar el código.
	6. Actualiza en la base de datos el atributo “continuar_analisis” poniéndolo en true.
	7. Invoca al CU “Obtener servidor óptimo”.
	8. Notifica al servidor que existe un código para analizar.
	9. El sistema envía al servidor el identificador del fichero, el ip del servidor y el identificador de la herramienta en la base de datos.
Flujo Alterno 1a “Solicitudes Pendientes”	
	1a.1 El sistema verifica que existen solicitudes pendientes.
	1a.2 Guarda la solicitud enviada por el Front_End en la BD con estado “pendiente”.
	1a. 2 Ir a la acción 5.

Flujo Alterno 3a “Conexión fallida a la base de datos”	
	3a.1 Envía un mensaje de error al Front_End “Error en conexión a la base de datos”, intente enviar nuevamente la solicitud.
	3a.2 El sistema espera que el Front_End envíe nuevamente la solicitud o verifica si existen solicitudes pendientes.
	3a.3 Ir a la acción 1.
Flujo Alterno 5a “No existe herramienta disponible”	
	5a. 1 El sistema almacena en la BD la solicitud con estado de “pendiente”.
	5a. 2 Verifica en la BD si existe otra solicitud para ser analizada.
	5a. 3 Ir a la acción 5.

Tabla 2: Descripción textual del caso de uso 1 “Procesar análisis”

Caso de Uso	
CU-2	Finalizar análisis
Propósito	Determinar las vulnerabilidades de un código.
Actores	Caso de uso “Procesar análisis” (inicia)
Resumen	El caso de uso se inicia cuando el sistema envía los datos del fichero y herramienta al servidor que va analizar el código. Finaliza una vez que los servidores python notifican que el código ha sido analizado.
Referencias	RF7, RF8, RF9, RF10, RF11, RF12, RF13
Acción del Actor	**Respuesta del Sistema**
	1. El servidor se conecta a la base de datos.
	2. Obtiene los datos en la BD del fichero a

	analizar: • *id_fichero*: identificador del fichero a analizar. • *url*: dirección donde se encuentra ubicado el fichero. • *parámetros*: parámetros de la herramienta.
	3. El servidor se conecta al servidor FTP.
	4. Descarga el fichero del FTP con la dirección obtenida y muestra el mensaje “File Upload”.
	5. Solicita al Back_End analizar el código descargado.
	6. Actualiza la información del estado de la herramienta, poniéndola “ocupada”.
7. El Back_End (analizador) comienza a analizar el fichero.	8. El servidor se conecta a la base de datos.
	9. Actualiza sus datos en la base de datos: • *memoria_uso*: memoria en uso del servidor. • *cpu_uso*: porciento en uso del CPU del servidor.
10. El Back_End le informa al servidor que se ha terminado de analizar el código.	11. El servidor se conecta al FTP.
	12. Carga el resultado de las vulnerabilidades hacia el FTP y muestra el mensaje “File Download”.
	13. El servidor se conecta a la base de datos.
	14. Actualiza sus datos en la base de datos: • *memoria_uso*: memoria en uso del servidor. • *cpu_uso*: porciento en uso del CPU del servidor.

	15. El servidor actualiza el estado de la herramienta en la base de datos, poniéndola "desocupado".
	16. Se conecta al servidor jabber.
	17. El servidor le notifica al Front_End a través del servidor jabber que la herramienta ya ha analizado el código.
	18. El servidor notifica al sistema que terminó de analizar el código.
	19. El servidor elimina de la base de datos el fichero que terminó de analizar.
Flujo Alterno 1a "Conexión fallida"	
	1a.1 El servidor muestra un mensaje de error "Error en conexión a acceso a datos".
	1a.2 El sistema espera que se restablezcan las conexiones.
	1a.3 El sistema accede a la base de datos y busca el fichero pendiente.
	1a.4 Ir a la acción 5 del caso de uso base "Procesar Análisis".
Flujo Alterno 3a "Conexión fallida al FTP"	
	3a.1 Muestra un mensaje de error: "No se ha podido conectar al servidor".
	3a.2 El servidor espera que se restablezcan las conexiones.
	3a.3 Ir a la acción 3.
Flujo Alterno 8a "Conexión fallida"	

	8a.1 El servidor muestra un mensaje de error “Error en conexión”, no se pudo terminar de analizar el código.
	8a.2 El sistema espera que se restablezcan las conexiones.
	8a.3 Accede a la base de datos y busca ficheros pendientes.
	8a.4 Ir a la acción 5 del caso de uso base “Procesar Análisis”.
Flujo 11a “Conexión fallida al FTP”	
	11a.1 El servidor muestra un mensaje de error “No se ha podido conectar al servidor”.
	11a.2 El sistema espera que se restablezcan las conexiones.
	11a.3 Accede a la base de datos y busca el fichero pendiente.
	11a.4 Ir a la acción 5 del CU base “Procesar Análisis”.
Flujo Alterno 13a “Conexión fallida”	
	13a.1 Muestra un mensaje de error “Error en conexión”.
	13a.2 El servidor espera que se restablezcan las conexiones.
	13a.3 El sistema accede a la base de datos y busca el fichero pendiente
	13a.3 Ir a la acción 5 del CU base “Procesar Análisis”.

Flujo Alterno 16a "Conexión fallida al servidor jabber"	
	16a.1 Muestra un mensaje de error "No se ha podido conectar al servidor jabber".
	16a.2 El sistema espera que se restablezcan las conexiones.
	16a.3 El sistema accede a la base de datos y busca el fichero pendiente.
	16a.4 Ir a la acción 5 del CU base "Procesar Análisis".

Tabla 3: Descripción textual del caso de uso 2 "Finalizar análisis"

Caso de Uso	
CU-3	Obtener servidor óptimo
Propósito	Conocer cuál es el mejor servidor que puede atender la solicitud de análisis.
Actores	Caso de uso "Procesar análisis" (inicia)
Resumen	El caso de uso se inicia cuando el sistema verifica las herramientas que se encuentran disponibles para realizar el análisis y busca de los servidores que ejecutan esas herramientas, cuál es el óptimo. Finaliza una vez que cambia el estado de la herramienta.
Referencias	RF4, RF5

Acción del Actor	Respuesta del Sistema
	1. El sistema se conecta a la base de datos.
	2. Verifica que servidores poseen las herramientas disponibles para realizar el análisis.
	3. Obtiene los datos de los servidores en la BD: • *memoria_real*: memoria RAM del servidor.

	• *memoria_uso*: memoria en uso del servidor. • *cpu_uso*: porciento en uso del CPU del servidor. • *memoria_herramienta*: memoria RAM que utiliza la herramienta. • *cpu_herramienta*: porciento del CPU que utiliza la herramienta.
	4. El sistema determina la carga de los servidores con los datos obtenidos.
	5. Determina cuál es el servidor óptimo para realizar el análisis.
	6. Ir a la acción 8 del CU base "Procesar Análisis".
Flujo Alterno 1a "Conexión fallida"	
	1a.1 El sistema muestra un mensaje de error "Error en conexión a acceso a datos".
	1a.2 El sistema espera que se restablezcan las conexiones.
	1a.3 Accede a la base de datos y obtiene el fichero pendiente.
	1a.4 Ir a la acción 5 del CU base "Procesar Análisis".

Tabla 4: Descripción textual del caso de uso 3 "Obtener servidor óptimo"

Caso de Uso	
CU-4	Cancelar Análisis.
Propósito	Cancelar un análisis que se encuentra en curso.
Actores	Front_End (inicia)
Resumen	El caso de uso se inicia cuando el Front_End envía al sistema la solicitud de cancelar un análisis que se encuentra en curso.
Referencias	RF14

Acción del Actor	Respuesta del Sistema
1. El Front_End envía la solicitud de cancelar el análisis que se encuentra en curso, dado el identificador del fichero.	2. El sistema recibe la solitud.
	3. Verifica si la solicitud de análisis no se encuentra en estado “pendiente”.
	4. Verifica a partir del identificador del fichero, la herramienta que se encuentra realizando el análisis del código correspondiente a la solicitud recibida.
	5. Actualiza en la base de datos el atributo “continuar_analisis”, poniéndolo en false.
	6. Se cancela la ejecución de la herramienta que está analizando el código en el servidor correspondiente.
	7. El sistema muestra un mensaje al Front_End “Fin de Análisis”.
Flujo Alterno 3a “Estado pendiente”	
	3a.1 El sistema elimina en la base de datos la solicitud de análisis con estado de “pendiente”.

	3a. 2 El sistema envía un mensaje al Front_End "Fin de Análisis".
Flujo Alterno 4a "No existe herramienta analizando el código"	
	4a.1 El sistema elimina del FTP los resultados de las vulnerabilidades encontradas.
	4a. 2 El sistema envía un mensaje al Front_End "Fin de Análisis".

Tabla 5: Descripción textual del caso de uso 4 "Cancelar análisis"

2.7 Conclusiones Parciales

En el presente capítulo se elaboró el modelo del dominio, el cual permitió comprender el funcionamiento del sistema en términos de conceptos y sus relaciones. Se realizó una descripción de los actores del sistema y sus responsabilidades, se identificaron además los requisitos funcionales, siendo de gran importancia para el desarrollo del sistema ya que constituyen la funcionalidad del mismo. Se modeló el diagrama de casos de uso, el cual refleja un esquema de lo que se debe implementar. También se identificaron los requisitos no funcionales y se detallaron todos los casos de uso, siendo de gran valor para la construcción del sistema propuesto. Con la culminación de esta etapa se hace posible la entrada a la fase de diseño del sistema.

Capítulo 3: Diseño del Sistema

3.1 Introducción

En el presente capítulo se realiza el diseño del sistema. Se modelan los diagramas de clases del diseño los cuales permiten determinar con precisión lo que se desea programar. Se diseña la arquitectura a utilizar, así como los patrones empleados. El propósito de este capítulo es especificar una solución que pueda ser fácilmente convertida en código, construyendo así una arquitectura simple y fácilmente extensible.

3.2 Diseño del sistema

A partir de los requisitos definidos, se hace necesario realizar el diseño del sistema, en el que se formularán los modelos para preparar la entrada a las actividades de implementación. El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción, contribuye a una arquitectura estable y sólida, además de crear un plano del modelo de implementación (25). Un buen diseño del sistema permite que el mismo pueda ser implementado sin ambigüedades.

3.2.1 Arquitectura del Sistema

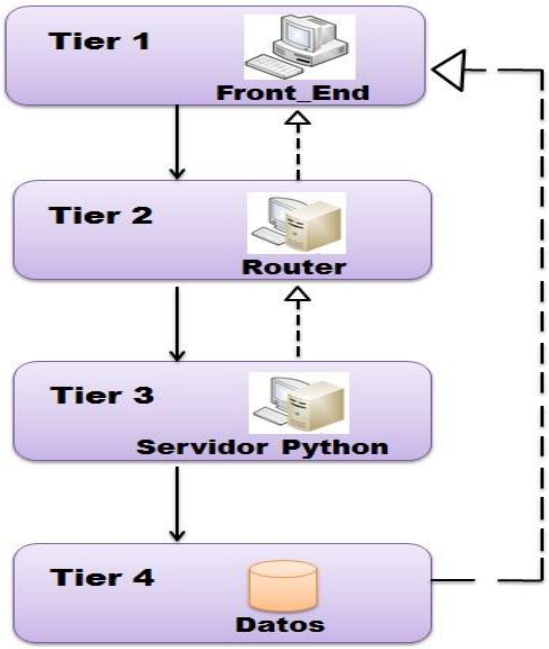

Figura 3: Arquitectura 4-Tier

La arquitectura se desarrolló sobre el patrón 4-Tier, en el cual se distribuyen físicamente los niveles, por cada nivel se encuentra un ordenador donde corre el código y los procesos. La arquitectura presentada tiene cuatro niveles, cada uno tiene una funcionalidad específica, del cual es responsable y están localizadas en diferentes servidores físicos.

- ***Front_End***: aplicación externa encargada de enviar las peticiones de análisis y los ficheros a analizar.
- ***Router****:* aplicación que tiene como objetivo la distribución de solicitudes de análisis que llegan desde el front end hacia los servidores python. Su sistema de distribución se basa en un algoritmo de selección de servidores, donde solo se notifica la ejecución de un análisis al servidor óptimo, de existir este. La comunicación con las aplicaciones front end y servidores python es mediante el uso del middleware ICE y utiliza el ORM SQLAlchemy para acceder a la base de datos. Este servidor debe permanecer en un ordenador donde su función sea única y no existan otras que alteren el rendimiento normal de la plataforma.
- ***Servidor Python***: son las aplicaciones que tienen como objetivo la ejecución del análisis de ficheros, utilizando los analizadores que se destinen para cada uno de estos, los cuales se van anexando a los servidores mediante el uso de plugins. La comunicación con el router es mediante el middleware ICE y con el front end es utilizando el servidor jabber para la actualización en tiempo real del proceso de análisis, ambos para notificar el fin de análisis de los ficheros. Cada servidor debe permanecer en un ordenador donde su función sea única y no existan otras que alteren el rendimiento normal de la plataforma.
- ***Datos***: en este nivel se encuentra el servidor de base de datos, el servidor FTP y el servidor Jabber.

Para representar la distribución lógica de los niveles, es decir, como está estructurado el código, se utilizó la arquitectura 4-Capas. El diseño de aplicaciones en capas es ideal para la creación de sistemas adaptables, donde cada componente puede ser utilizado y reutilizado en nuevas combinaciones para satisfacer los requisitos del negocio.

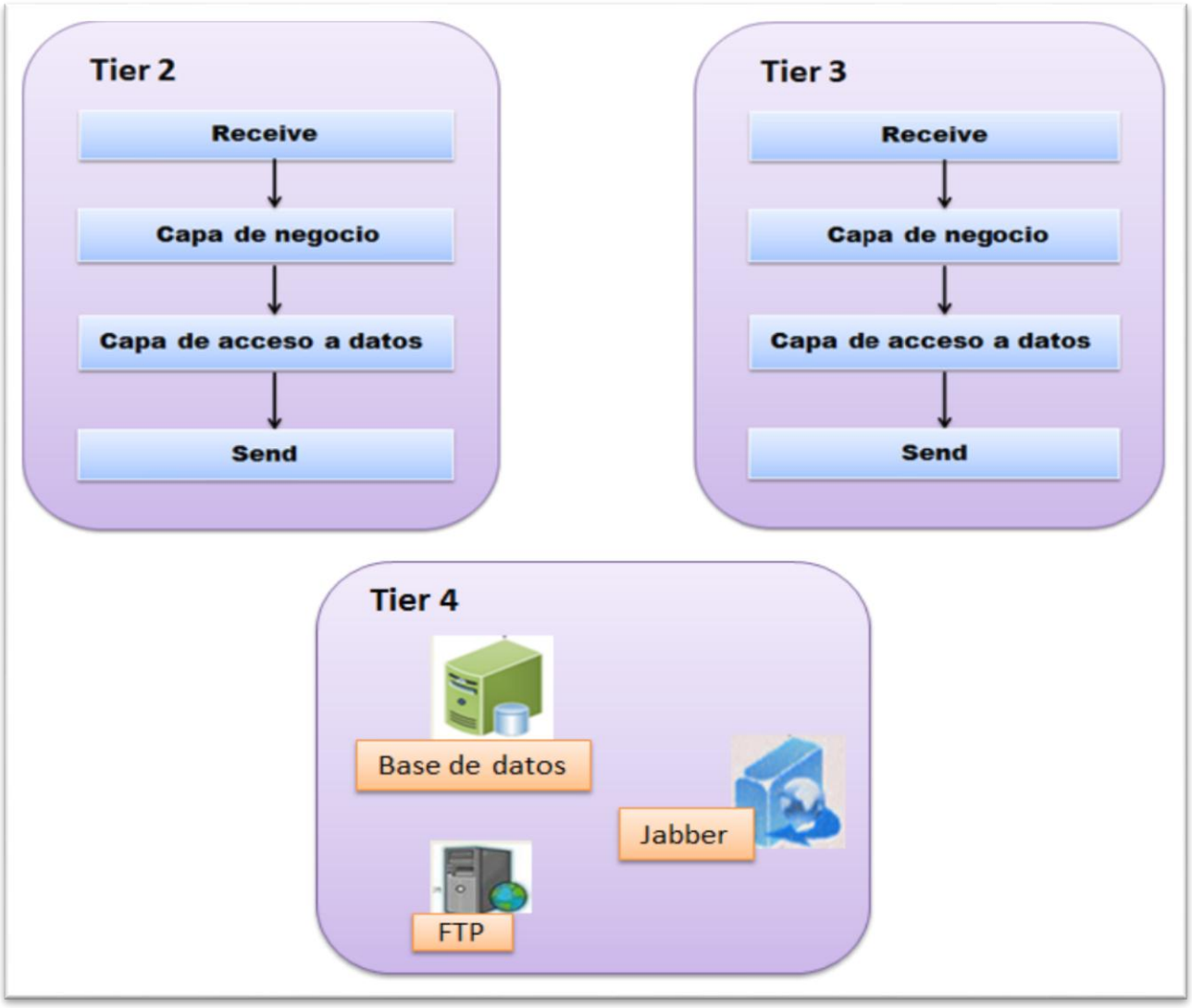

Figura 4: Arquitectura lógica por tier

En los tier 2 y 3 se encuentran cuatro capas lógicas:

- ***Receive***: esta capa se encarga de recibir los datos mediante el uso del objeto remoto que permite el middleware ICE. En el caso del router se reciben las solicitudes de análisis realizadas por el front end y la notificación de fin de análisis que envían los servidores. En el caso de los servidores python reciben la notificación de realizar análisis, enviadas desde el router.

- ***Capa de negocio***: capa donde se encuentran todas las funcionalidades relacionadas con la lógica del negocio. Es donde se establecen todas las reglas que deben cumplirse para el desarrollo del sistema.
- ***Capa de acceso a datos***: capa encargada de acceder a los datos que se encuentran en la base de datos.
- ***Send***: esta capa se encarga de enviar los datos necesarios para el funcionamiento del sistema, mediante el uso de objeto remoto que permite el middleware ICE. En el caso del router envía la notificación de realizar análisis al servidor, mientras que los servidores envían la notificación de fin de análisis al router.

El tier 4 es el nivel donde se presenta la disposición de los servidores FTP, base de datos y jabber. En este nivel se almacenarán los datos con los que trabajará la plataforma y se establecerá la comunicación entre el front end y los servidores python (servidor jabber).

3.2.2.1 Características de la arquitectura de la plataforma

La arquitectura de la plataforma funciona con un grupo de computadoras conectadas a la red, se comunican y coordinan sus acciones mediante el middleware ICE formando un esquema cliente-servidor. Dentro de las características principales están:

- Las aplicaciones pueden utilizar máquinas que se encuentran separadas a cierta distancia.
- La carga de trabajo se distribuye en varias máquinas.
- Si se necesita añadir poder de cómputo a la plataforma, podrían añadirse solo más procesadores al sistema, lo que permite un desarrollo gradual conforme surjan las necesidades.
- El diseño de la plataforma se hace con la idea de posibilitar cambios futuros, que puedan escalar hacia sistemas más grandes.
- El diseño de la plataforma posibilita que si una máquina falla, alguna otra máquina se encargue del trabajo.
- Se puede utilizar diversas métricas de desempeño. El tiempo de respuesta es una, pero también lo son el rendimiento, uso del sistema y cantidad consumida de la capacidad de la red.

3.2.1.2 Flujo de la arquitectura del sistema

El flujo comienza cuando el front end realiza una petición de análisis al router solicitando que se analice el código que se encuentra en el servidor FTP. El router de acuerdo a las herramientas que pueden analizar el código verifica cuales se encuentran disponibles y busca el servidor que mejor pueda atender esta solicitud. Si existe alguna herramienta disponible, el router le notifica al servidor que debe realizar el análisis de un fichero, sino se guarda la solicitud en la base de datos con estado “pendiente”. El servidor busca la dirección del código a analizar en la base de datos y descarga el fichero correspondiente desde el FTP, solicitando así a la herramienta que analice el código. Una vez terminado el análisis el servidor carga el resultado al FTP y le notifica al front end a través del servidor jabber que el código ya ha sido analizado. Inmediatamente el servidor le notifica al router que existe una herramienta desocupada, para el caso de que existan otros ficheros por analizar.

3.2.2 Patrones que se utilizan

3.2.2.1 Patrones de arquitectura

Los patrones de arquitectura definen la estructura general del software y se relacionan con el diseño a gran escala. (25) Los patrones arquitectónicos que se utilizan para el diseño de la propuesta de solución del sistema son:

1. **N- Tiers:**

N- Tiers es un estilo que define el despliegue de las capas de la aplicación, se caracteriza por la distribución de los componentes de servicios y despliegue distribuido. A continuación se muestran los beneficios que aporta:

- Cada nivel es independiente de los demás.
- Se puede actualizar o modificar un nivel sin afectar a la aplicación en su conjunto.
- Es escalable puesto que los niveles están basados en el despliegue de las capas.

2. **N-Capas:**

La programación por capas es un estilo de programación cuyo objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; en este caso se utiliza la capa de negocio y la capa de datos (no se tiene capa de presentación).

- *Capa de negocio*: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso.

  Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. En esta capa se reciben las solicitudes y una vez terminado el análisis se presentan los resultados. La capa de negocio se comunica además con la capa de datos, para solicitar los datos al gestor de base de datos, almacenar o recuperar datos de él.

- *Capa de datos*: es la encargada de acceder a los datos que están en uno o más gestores de bases de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

3.2.2.2 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo del software y otros ámbitos referentes al diseño de interacción o interfaces, se aplican a un elemento específico del diseño. (25)

Factory

Define un interfaz para crear un objeto y permitir a subclases decidir sus instancias. El método de la fábrica deja a una clase la creación de ejemplares o copias a subclases. (26) Este patrón se utilizó para fabricar clases de análisis, las cuales tendrán propiedades específicas para el análisis y ejecución de ficheros y herramientas respectivamente. Además, ayuda a modificar y añadir nuevos analizadores a la plataforma haciendo función de plugins.

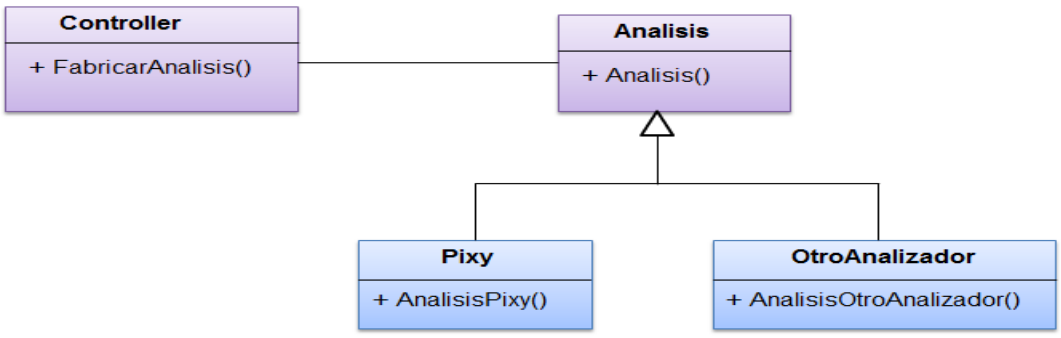

Figura 5: Patrón de diseño factory

3.2.3 Modelo de Diseño

Con el objetivo de realizar una representación previa del sistema a desarrollar se realiza el modelo de diseño, el cual sirve de abstracción a la implementación del sistema y es utilizado además, como entrada fundamental en las actividades de implementación.

3.2.3.1 Diagrama de paquetes

El diagrama de paquetes se usa para estructurar el modelo de diseño, dividiéndolo en partes más pequeñas (25), de esta forma se obtendrá una mejor organización y comprensión del sistema. En este sentido, para lograr mayor claridad en la modelación, se agruparon las clases por paquetes. La distribución existente en el diagrama centra la atención en formar paquetes con las funcionalidades que responden a los mismos procesos de negocio.

Para el módulo “Distribución y Ejecución de Herramientas” se diseñó el diagrama de paquetes que se muestra en la figura 6, el mismo fue dividido en 5 paquetes. En los paquetes “Servidores” y “Enrutador” se encuentran agrupadas las funcionalidades que responden a una misma lógica del negocio y los paquetes “Comunicación”, “Framework” y “Core” son paquetes auxiliares que utilizan los paquetes mencionados anteriormente.

DIAGRAMA DE PAQUETES

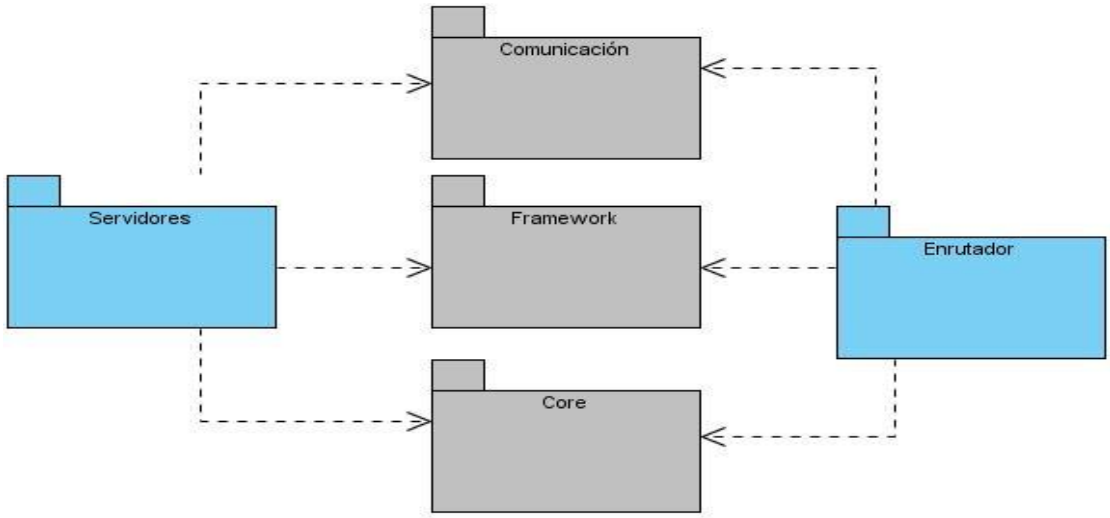

Figura 6: Diagrama de paquetes

3.2.3.2 Diagramas de clases del diseño

A través del flujo de trabajo de diseño uno de los artefactos más importantes a obtener son los diagramas de clases, donde se exponen las clases del diseño que intervienen en las realizaciones de los casos de uso del sistema. En la figura 7 y 8 se mostrarán los diagramas de clases del diseño para cada paquete (Enrutador y Servidores), en los que se modelan las clases correspondientes y sus relaciones.

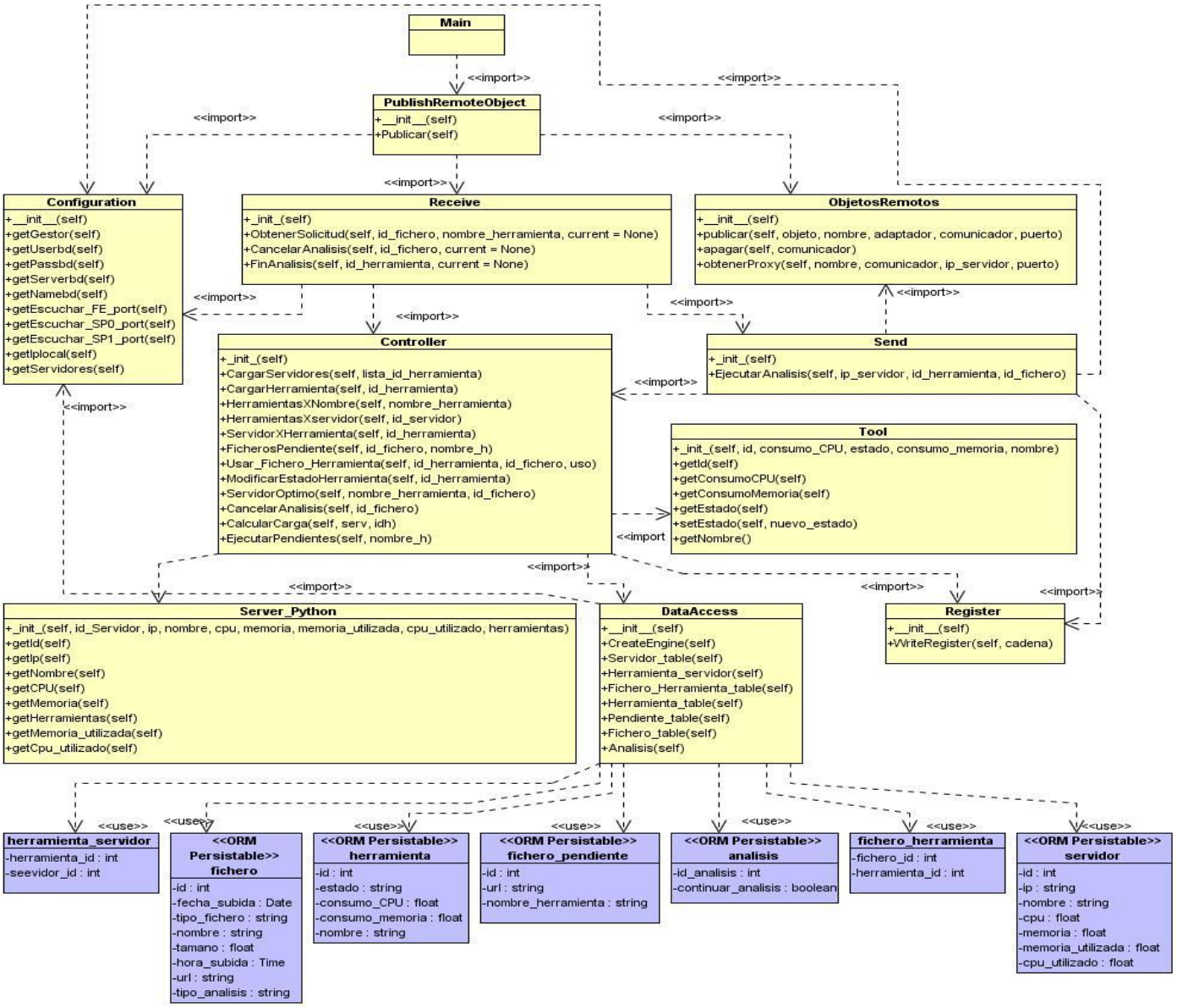

Figura 7: Diagramas de clases del diseño del paquete enrutador

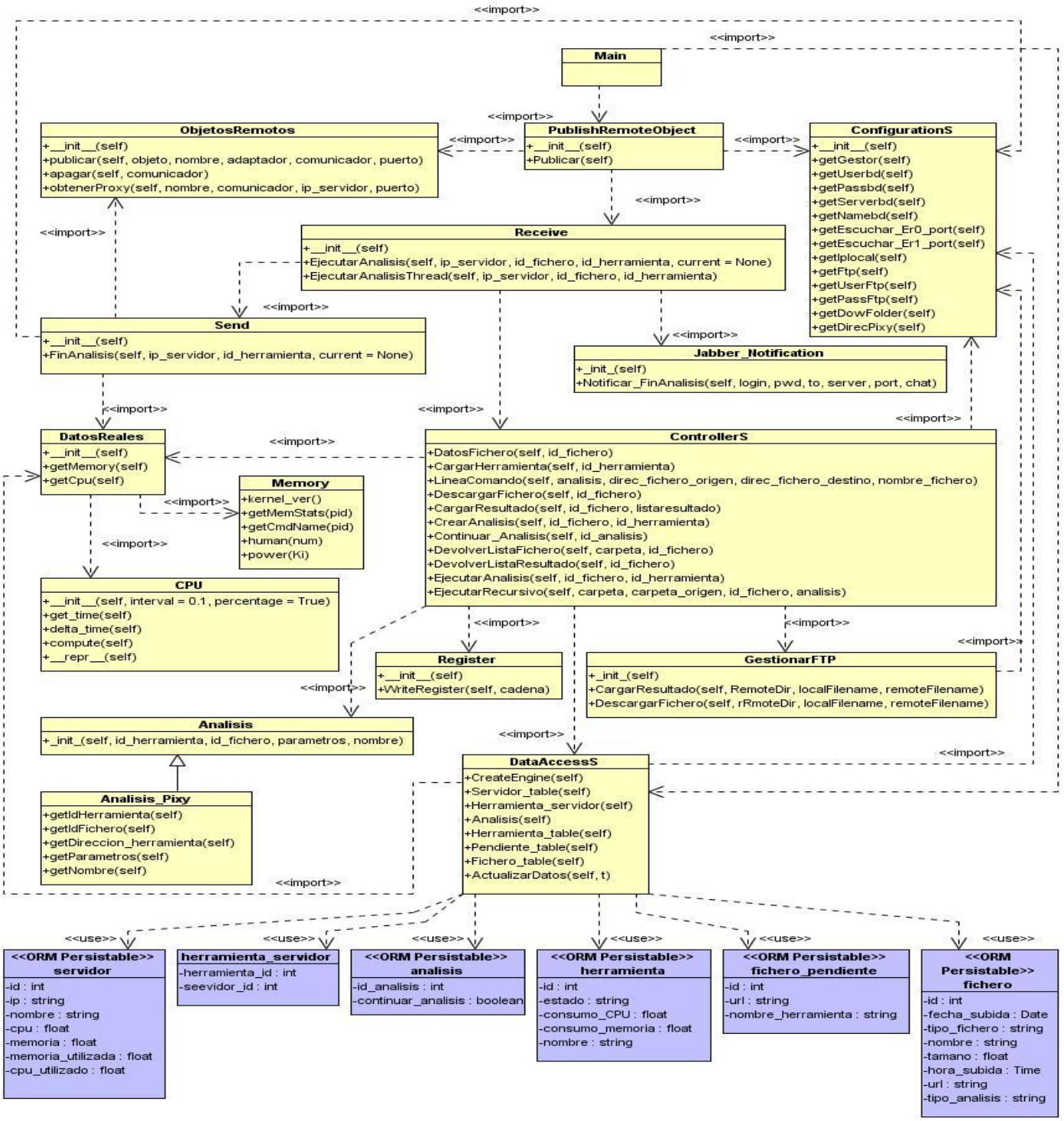

Figura 8: Diagrama de clases del diseño del paquete servidores

3.3 Modelo de datos

El modelo de datos describe las representaciones lógicas y físicas de datos persistentes utilizados por una aplicación. (4)

3.3.1 Modelo lógico de datos (diagrama de clases persistentes)

A través del diagrama de clases persistentes se representan los objetos que se deben almacenar en la base de datos y que son necesarios para que la información pueda persistir. (4)

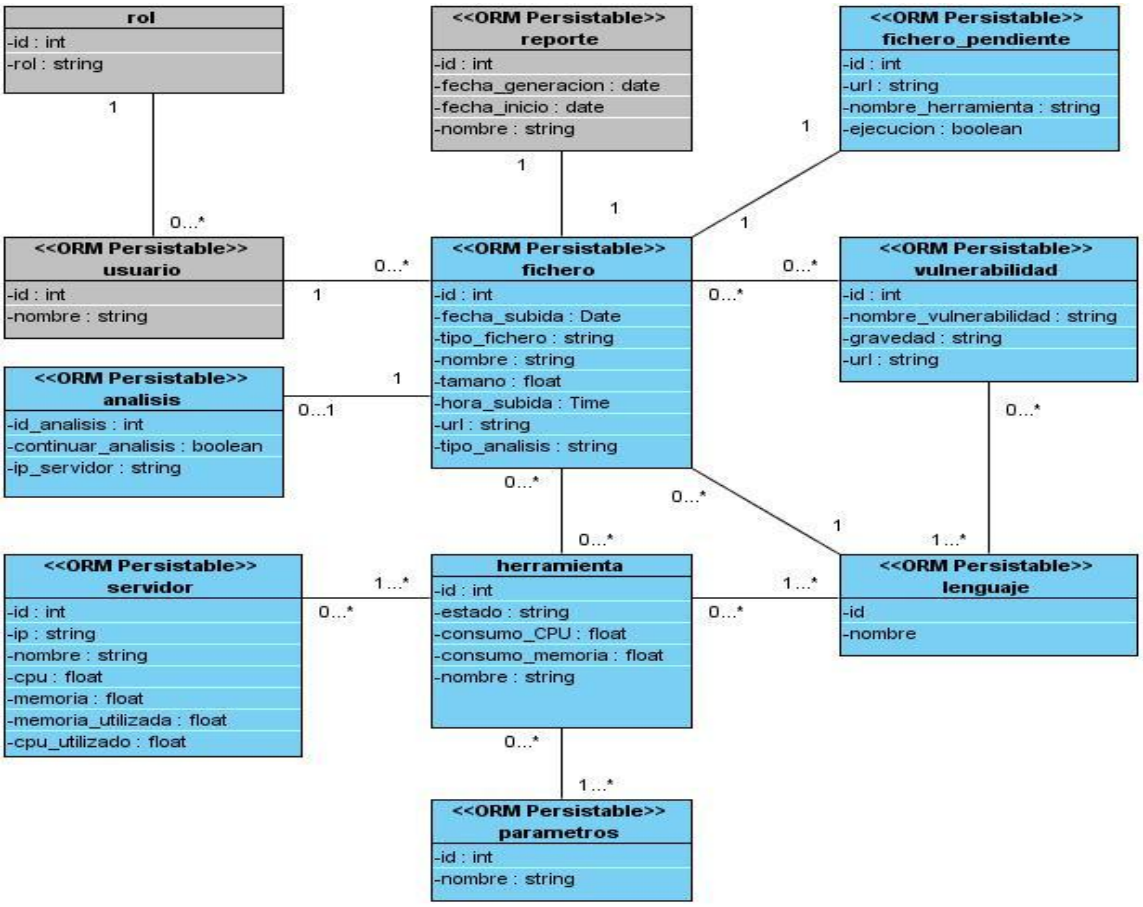

Figura 9: Modelo lógico de datos

3.3.2 Modelo físico de datos (modelo de datos)

El modelo de datos es la representación física o estructura de las tablas de la base de datos. El modelo físico se desarrolló a partir de la base del conjunto de clases persistentes y sus asociaciones en el modelo de diseño. (4)

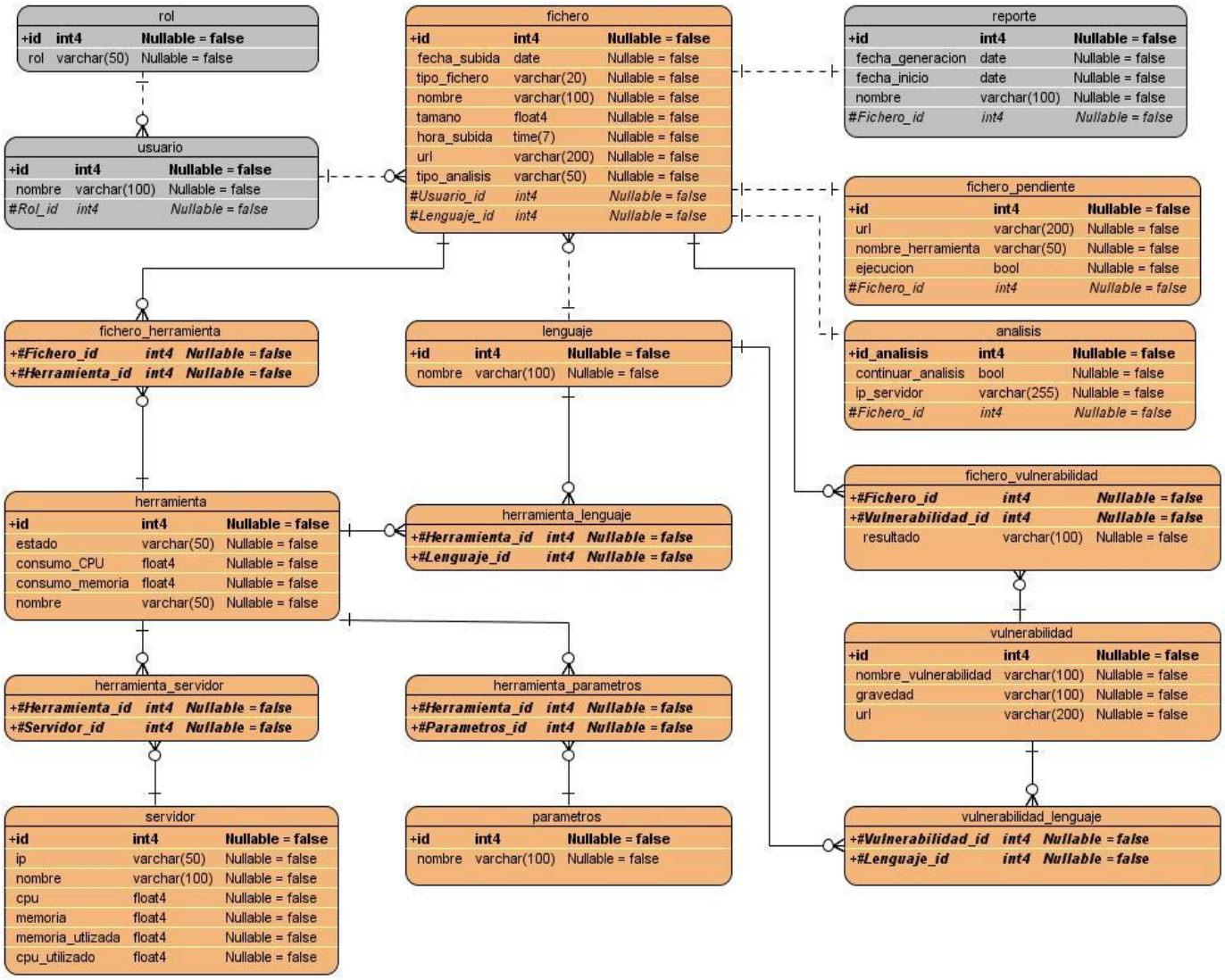

Figura 10: Modelo físico de datos

3.4 Conclusiones parciales

En este capítulo se realizaron los diagramas de clases del diseño, sirviendo como base fundamental para la implementación del sistema. Se utilizó el patrón de diseño Factory el cual se emplea para la creación de plugins, con el fin de anexar los analizadores a la plataforma. Se desarrolló una arquitectura 4-Tiers para representar la distribución física del sistema y 4-capas para la distribución lógica de las tiers, brindando la posibilidad de dividir funcionalidades y lograr una mayor reutilización a la hora de separar el negocio. Además, quedó definida la arquitectura, lo que permite tener una idea completa del software.

Capítulo 4: Implementación y Pruebas del Sistema

4.1 Introducción

El presente capítulo tiene como objetivo desarrollar los artefactos correspondientes a la implementación del sistema, donde se tendrá como base el diseño realizado en el capítulo anterior. Se modelará el diagrama de componentes y diagrama de despliegue, conformando los modelos de implementación; en los que se obtendrá una visión general del desarrollo de la aplicación, así como los resultados del software. Una vez concluido la implementación se realizarán las pruebas al sistema.

4.2 Modelo de implementación

El modelo de implementación describe como los elementos del modelo del diseño se implementan en términos de componentes y como estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue. (27)

4.2.1 Subsistema de implementación

Los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables. Un subsistema es una colección de componentes y otros subsistemas usados para estructurar el modelo de implementación y dividirlos en partes que pueden ser integradas y probadas de forma separada. (27)

Con el objetivo de organizar la vista de realización del sistema, la “Plataforma de Análisis Estático de Vulnerabilidades de Código” cuenta con dos subsistemas de implementación: *subsistema enrutador* y *servidor*. Dentro de cada uno se encontrarán todos sus componentes y relaciones.

4.2.2 Diagrama de componentes

Un diagrama de componentes es un diagrama UML que describe los elementos físicos del sistema y sus relaciones. Los componentes físicos incluyen archivos, librerías, módulos, ejecutables o código fuente. En este diagrama se muestra la organización y las dependencias entre un conjunto de componentes. (28)

4.2.2.1 Diagrama de componentes del subsistema Enrutador

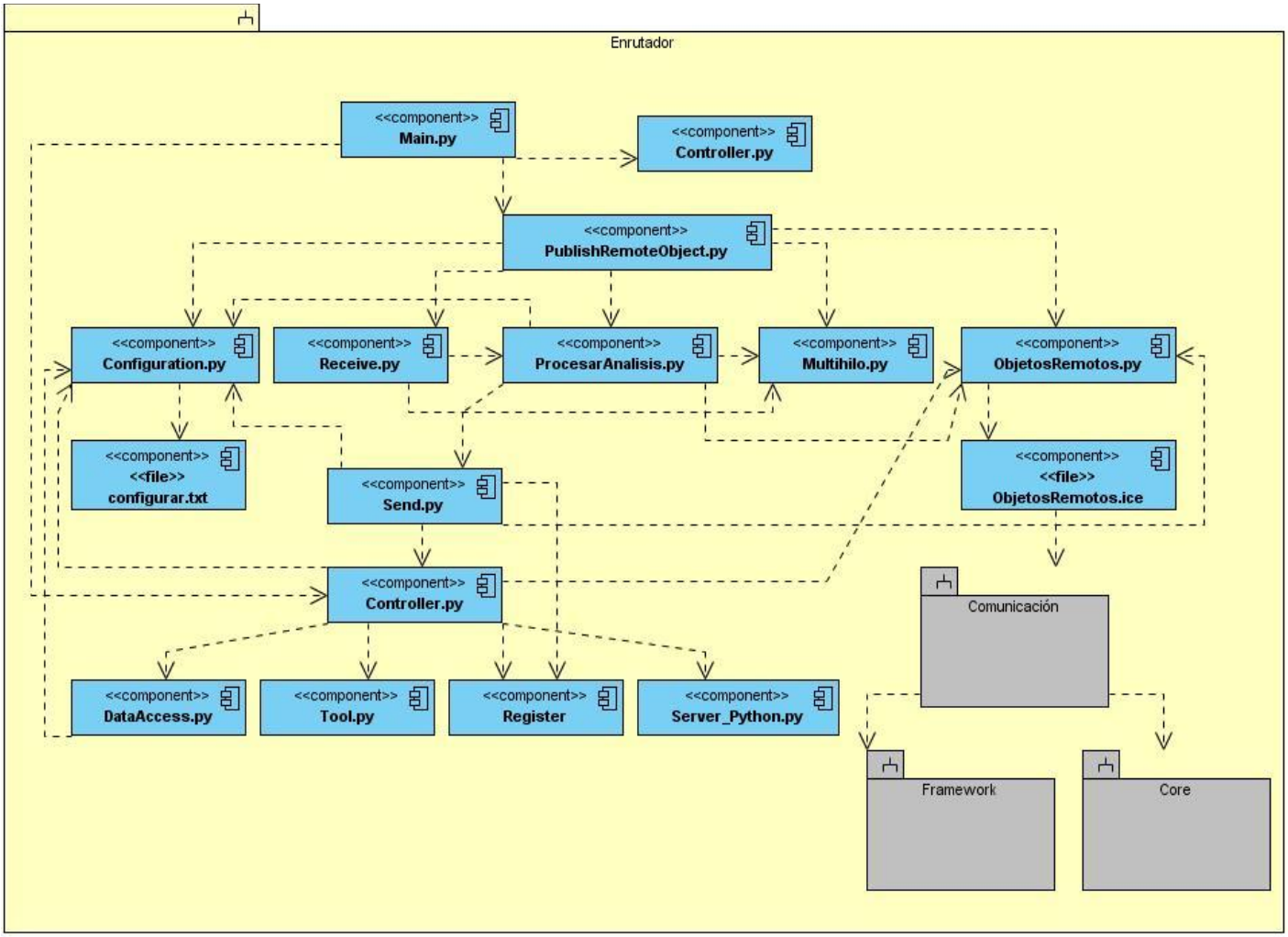

Figura 11: Diagrama de componente del subsistema enrutador

4.2.2.1.1 Descripción de los componentes

Subsistema enrutador: en este subsistema se agrupan todos los componentes encargados de procesar las solicitudes de análisis. Dentro de este se encuentran los siguientes componentes:

Código fuente

- Main.py
- PublishRemoteObject.py
- Configuration.py

- Receive.py
- ObjetosRemotos.py:
- Send.py
- Controller.py
- DataAccess.py
- Tool.py
- Register.py
- Server_Python.py
- ProcesarAnalisis.py
- Multihilo.py

Archivos de configuración

- configurar.txt
- ObjetosRemotos.ice

Subsistemas

Los subsistemas que se mencionan a continuación son subsistemas auxiliares que se utilizan para publicar el objeto remoto.

- Comunicación
- Core
- Framework

4.2.2.2 Diagrama de componentes del subsistema servidor

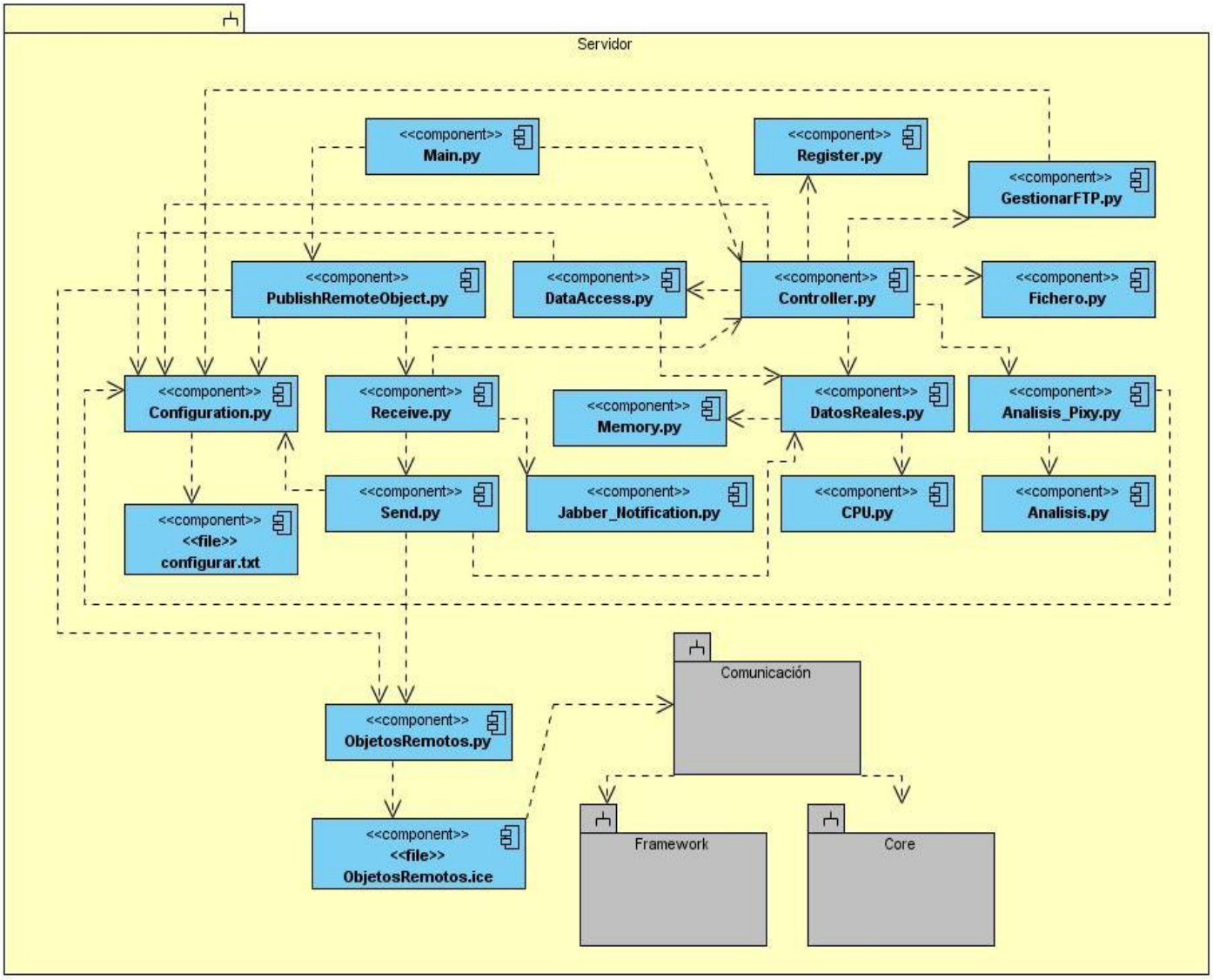

Figura 12: Diagrama de componentes del subsistema servidor

4.2.2.2.1 Descripción de los componentes

Subsistema servidor: en este subsistema se agrupan todos los componentes encargados de la ejecución de las herramientas y del proceso de descarga y subida de los ficheros hacia el FTP. Dentro de este se encuentran los siguientes componentes:

Código fuente

- Main.py
- PublishRemoteObject.py
- DataAccess.py
- Configuration.py
- Receive.py
- ObjetosRemotos.py
- Send.py
- Controller.py
- Register.py
- GestionarFTP.py
- Fichero.py
- Analisis.py
- Analisis_Pixy.py
- DatosReales.py
- Memory.py
- CPU.py
- Jabber_Notification.py

Archivos de configuración

- configurar.txt
- ObjetosRemotos.ice

Subsistemas

Los subsistemas que se mencionan a continuación son subsistemas auxiliares que se utilizan para publicar el objeto remoto.

- Comunicación
- Core
- Framework

4.2.2 Diagrama de despliegue

El modelo de despliegue muestra la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. (29) Por su parte un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento. (28)

Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representan típicamente un procesador o un dispositivo sobre el que se pueden desplegar los componentes. (28) A continuación se presenta el diagrama de despliegue del módulo: "Distribución y Ejecución de Herramientas" de la Plataforma de Análisis Estático de Vulnerabilidades de Código.

DIAGRAMA DE DESPLIEGUE

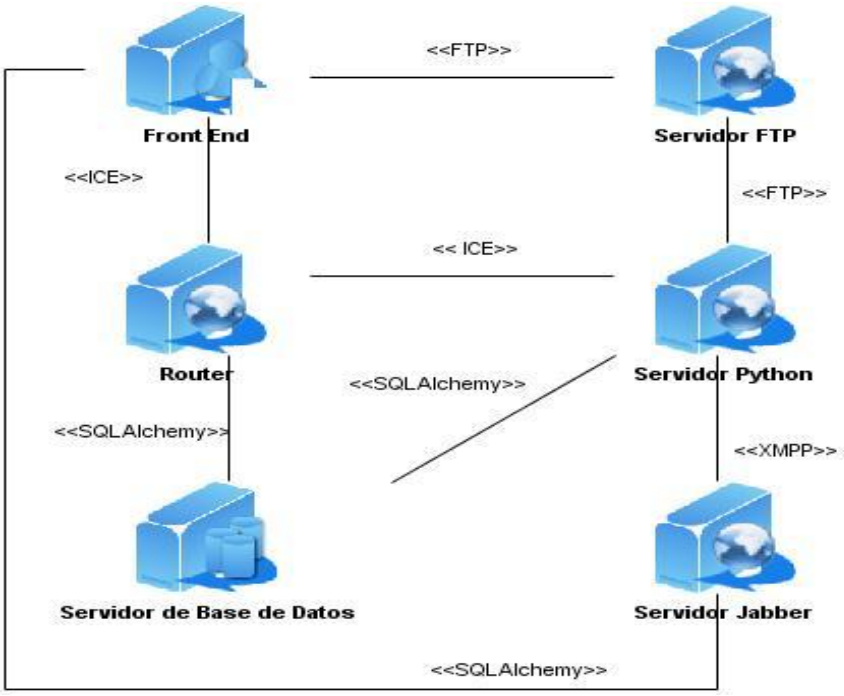

Figura 13: Diagrama de despliegue

4.2.2.1 Descripción de los nodos:

Nombre del nodo	Descripción
Nodo Front End	En este nodo se modela la aplicación externa encargada de enviar peticiones de análisis y los ficheros que se desean analizar, así como sus datos.
Nodo Servidor FTP	En este nodo se modela el servidor FTP VSFTPD, encargado del almacenamiento de ficheros a analizar y los resultados de las vulnerabilidades del código encontradas.
Nodo Servidor de Base de Datos	En este nodo se encuentra la base de datos del sistema, se cuenta con el gestor PostgreSQL para este objetivo.
Nodo Servidor Python	En este nodo se modela el servidor jabber, encargado de permitir el envío de mensajes instantáneos desde los servidores python al front end.
Nodo Router	En este nodo se modela la distribución de los análisis a las distintas herramientas para detectar las vulnerabilidades.

Tabla 6: Descripción de los nodos del diagrama de despliegue

4.3 Pruebas

Una vez generado el código fuente de la aplicación, el software debe ser probado para corregir el máximo de errores que pueda existir antes de la entrega al cliente. Las pruebas del software no son más que la actividad en la cual un sistema o componente es ejecutado bajo condiciones. Es un elemento crítico para la garantía de la calidad de la aplicación y representa una revisión final de las especificaciones del diseño y de la codificación. (30)

Una vez concluido la implementación del sistema se realizarán las pruebas necesarias para verificar y revelar la calidad del producto, identificando así los posibles fallos de la implementación.

4.3.1 Métodos de prueba

4.3.1.1 Prueba de caja negra

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la

entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. (31)

4.3.1.2 Prueba de caja blanca

La prueba de caja blanca se basa en el minucioso examen de los detalles procedimentales del código a evaluar, por lo que se hace necesario conocer la lógica del programa. (32) Se comprueban los caminos lógicos del software, proponiendo casos de prueba que examinen que están correctas todas las condiciones o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar. (31)

4.3.1.2.1 Técnica de caja blanca

Prueba del camino básico: es una técnica de prueba de caja blanca que permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. (31)

Para aplicar esta técnica se siguen los siguientes pasos:

1. Dibujar el grafo asociado a partir del diseño o código fuente.
2. Calcular la complejidad ciclomática del grafo.
3. Determinar un conjunto básico de caminos independientes.
4. Preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (31)

Se decide utilizar las pruebas de caja blanca ya que la “Plataforma de Análisis Estático de Vulnerabilidades de Código. Distribución y Ejecución de Herramientas” no cuenta con interfaz, razón por la cual no se puede aplicar las pruebas de caja negra.

4.3.2 Pruebas realizadas a la aplicación

A continuación se mostrará las pruebas de caja blanca realizadas a las funcionalidades de la aplicación:

Prueba de caja blanca

Caso de prueba #1

```
def ServidorOptimo(self,nombre_herramienta,id_fichero):
    servidores,idh=self.CargarServidores(self.HerramientasXNombre(nombre_herramienta))    1
                  2                3
    if(servidores!=None and idh!=None):
        if(servidores.__len__()==0):        4
            self.FicherosPendiente(id_fichero,nombre_herramienta)          5
            print 'No existen servidores para atender la solicitud 1'
            return False                6
        if(servidores.__len__()==1):        7
            if(self.CalcularCarga(servidores[0], idh[0])!=0):      8
                return servidores[0].getIp(),idh[0]   9
            else:
                self.FicherosPendiente(id_fichero,nombre_herramienta)          10
                print 'No existen servidores para atender la solicitud 2'
                return False            11
        else:
            carga=0
            cont=0          12
            id_h=0
            for i in servidores:            13
                next=self.CalcularCarga(i, idh[cont])      14
                if(next>carga):         15
                    carga=next
                    ip=i.getIp()        16
                    id_h=idh[cont]
                cont=cont+1        17
             18
        if(carga==0):      19
            self.FicherosPendiente(id_fichero,nombre_herramienta)          20
            print 'No existen servidores para atender la solicitud'
            return False        21
        else:
            self.Usar_Fichero_Herramienta(id_h, id_fichero,'insertar')          22
            return ip,id_h      23
    else:             24
        print 'Problemas en el Acceso a Datos'
        return False              25
26
```

Figura 14: Código fuente de la funcionalidad "Servidor óptimo"

1. Grafo asociado

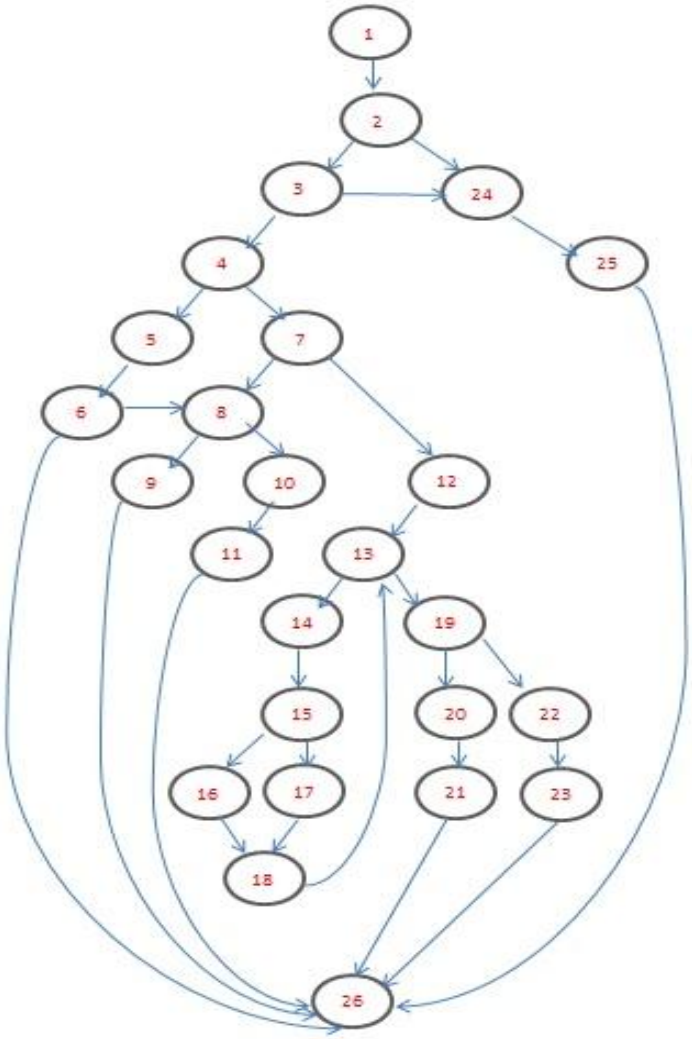

Figura 15: Grafo asociado a la funcionalidad "Servidor óptimo"

2. Complejidad ciclomática

Existen tres formas de calcular la complejidad ciclomática. A continuación se muestran:

V (G) = R V (G) = A – N + 2 V (G) = P + 1

Definiciones de términos:

V (G): complejidad ciclomática

R: número de regiones

A: cantidad de aristas

N: cantidad de nodos

P: número de nodos predicados

V (G)= 9 V (G)= 33 – 26 + 2 = 9 V (G)= 8 + 1 = 9

Por tanto la complejidad ciclomática del grafo es 3.

3. Caminos independientes

Camino 1: 1, 2, 3, 4, 5, 6, 26

Camino 2: 1, 2, 3, 4, 7, 8, 9, 26

Camino 3: 1, 2, 3, 4, 7, 8, 10, 11, 26

Camino 4: 1, 2, 3, 4, 7, 12, 13, 14, 15, 16, 17, 18, 13, 19, 20, 21, 26

Camino 5: 1, 2, 3, 4, 7, 12, 13, 14, 15, 17, 18, 13, 19, 22, 23, 26

Camino 6: 1, 2, 3, 4, 7, 12, 13, 19, 20, 21, 26

Camino 7: 1, 2, 3, 4, 7, 12, 13, 19, 22, 23, 26

Camino 8: 1, 2, 24, 25

Camino 9: 1, 2, 3, 24, 25

4. Casos de prueba

No. de caminos	Casos de prueba	Objetivo	Resultados esperados
1	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=0	Probar cuando no existen servidores para atender la solicitud.	Se crea un fichero pendiente y se retorna false.
2	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=1 carga=[1,∞]	Probar cuando existe solo un servidor y la carga cumple con las condiciones para atender la solicitud.	Se retorna el id del servidor y la herramienta que atenderá la petición.

3	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=1 carga=0	Probar cuando existe un solo servidor y la carga no cumple con las condiciones para atender la solicitud.	Se crea un fichero pendiente y se retorna false.
4	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=[2,∞] carga=0	Cuando las cargas de los servidores no cumplen con las condiciones para atender la solicitud, pasando más de una vez por el ciclo.	Se crea un fichero pendiente y se retorna false.
5	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=[2,∞] carga=[1,∞]	Cuando se encontró un servidor con la mejor carga para atender la solicitud, pasando más de una vez por el ciclo.	Se retorna el id del servidor y la herramienta que atenderá la petición.
6	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=[2,∞] carga=0	Cuando las cargas de los servidores no cumplen con las condiciones para atender la solicitud, sin pasar por el ciclo.	Se crea un fichero pendiente y se retorna false.
7	servidores=[listaservidores] idh=[id herramienta] servidores.__len__()=[2,∞] carga=[1,∞]	Cuando se encontró un servidor con la mejor carga para atender la solicitud, sin pasar por el ciclo.	Se retorna el id del servidor y la herramienta que atenderá la petición.
8	servidores= None	Cuando no existen servidores. Existe problema con la base de datos.	Se retorna false.
9	idh=None	Cuando no existe herramienta. Existe problema con la base de datos.	Se retorna false.

Tabla 7: Caso de prueba de la funcionalidad "Servidor óptimo"

Caso de Prueba #2

```
def DevolverListaResultado(self,id_fichero):
    fichero=self.DatosFichero(id_fichero)
    nombre =fichero.GetNombre()
    nuevo1=nombre.split('.')[0]                    1
    hola =os.listdir('FTP/'+nuevo1+'/')
    nuevalista=[]
    for i in hola:                2
        if(i.find('xml')>0):      3
            nuevalista.append(i)  4
        5
    6
    return nuevalista             7
```

Figura 16: Código fuente de la funcionalidad "Devolver listado de los resultados"

1. Grafo asociado

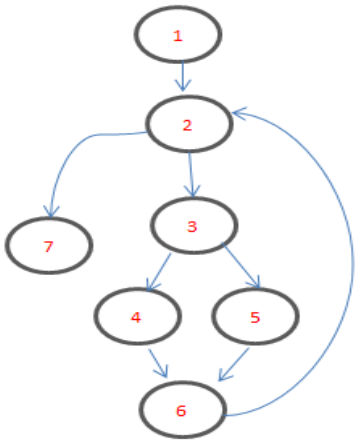

Figura 17: Grafo asociado a la funcionalidad "Devolver listado de resultado"

1. Complejidad ciclomática:

V (G) = 3

La complejidad ciclomática del grafo asociado es 3.

2. Caminos independientes

Camino 1: 1, 2, 3, 4, 6, 7

Camino 2: 1, 2, 3, 5, 6, 7

Camino 3: 1, 2, 7

3. Casos de prueba:

No. de caminos	Casos de prueba	Objetivo	Resultados esperados
1	hola=[lista_archivos] hola.__len__() > 0	Probar que pase por el ciclo y que los nombres de los archivos tengan extensión XML.	Devuelve una lista con todos los archivos XML que estén en la carpeta.
2	hola=[lista_archivos] hola.__len__() > 0	Probar que pase por el ciclo y que los nombres de los archivos no tengan extensión XML.	Devuelve una lista con todos los archivos XML que estén en la carpeta.
3	hola=[lista_archivos] hola.__len__() = 0	Probar que no pase por el ciclo.	Devuelve una lista vacía.

Tabla 8: Caso de prueba de la funcionalidad "Devolver listado de resultados"

4.4 Conclusiones parciales

En el presente capítulo se obtuvo como principal resultado el modelo de implementación de la solución del sistema, a través del diagrama de componentes y de despliegue. Con el diagrama de componentes se muestran los elementos físicos agrupados en términos de subsistemas, permitiendo de esta forma lograr mayor organización en el código. El diagrama de despliegue permitió mostrar la disposición de los distintos nodos que entran en la composición del sistema y el reparto de los programas ejecutables sobre estos nodos.

Se expuso además el método de pruebas definido para verificar las funcionalidades de la solución, donde se realizaron las pruebas necesarias al código de la aplicación para detectar los posibles errores que pudiese tener. De esta forma se le da cumplimiento a todos los objetivos trazados inicialmente.

Conclusiones Generales

Una vez concluido el desarrollo de la “Plataforma para el Análisis Estático de Vulnerabilidades del Código. Distribución y Ejecución de Herramientas” se puede plantear que se cumplieron todos los objetivos trazados al inicio de la investigación, razón por la cual se arriba a las siguientes conclusiones generales:

- El estudio y seguimiento de la propuesta de solución realizada permitió desarrollar una aplicación que cumpliera con todos los requisitos definidos.
- Se logró desarrollar una arquitectura genérica, capaz de seleccionar el servidor adecuado para ejecutar la herramienta de análisis.
- Se logró brindar flexibilidad en cuanto a la inclusión de nuevos analizadores al sistema, lo que permite analizar códigos en diferentes lenguajes de programación.
- La aplicación es capaz de distribuir las cargas en diferentes servidores optimizando así el rendimiento ante las peticiones realizadas.
- El sistema permite, analizar tanto un proyecto como un script y brindar los resultados de las vulnerabilidades existentes.

Recomendaciones

Con el desarrollo de la aplicación y el resultado de la investigación se proponen algunas recomendaciones con el objetivo de mejorar la calidad en los servicios brindados:

- Se recomienda que se realicen las pruebas de estrés con el objetivo de simular grandes cargas de trabajo para observar de qué forma se comporta la aplicación ante situaciones de uso intenso.
- Incluir nuevos analizadores para la revisión de aplicaciones en otros lenguajes de programación.
- Mejorar el algoritmo para obtener el servidor óptimo, para esto es necesario un historial amplio en las ejecuciones de los analizadores teniendo en cuenta las especificaciones de software que utilizan y en las condiciones que se realizan.
- Implantar el sistema en el Laboratorio de Seguridad Informática de la facultad 2 para brindar servicios a la comunidad universitaria.

Referencias Bibliográficas

1. **Filipiak, Joanna María.** *Herramienta de análisis estático de código para encontrar vulnerabilidades de seguridad en las aplicaciones Web.* Universidad Carlos III . Madrid : s.n., 2009. pág. 134, Proyecto Fin de Carrera.

2. **Expósito, Raúl;.** *¿Qué es el Análisis Estático de Código?* Madrid : s.n., 2009. pág. 19.

3. **Mazzarri, Milton.** Pylint: Análisis estático del código fuente en Python. [En línea] marzo de 2010. http://www.slideshare.net/milmazz/pylint.

4. **Hernández Yeja, Adrián y Tejas de la Cru, Yosbany.** *Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP.* Universidad de las Ciencias Informáticas. Cuba : s.n., 2010. pág. 180, Trabajo de Diploma.

5. **Culebro Juárez, Montserrat, Gómez Herrera, Wendy Guadalupe y Torres Sánchez, Susana.** *Software libre vs software propietario. Ventajas y desventajas.* México : s.n., 2006. pág. 170.

6. **Fortify.** Fortify 360. [En línea] https://www.fortify.com/downloads2/public/Fortify_360_Datasheet_Spanish.pdf.

7. Lista de las herramientas para el análisis estático del código. [En línea] 2011. http://www.worldlingo.com/ma/enwiki/es/List_of_tools_for_static_code_analysis.

8. **Yasca.** Yasca v1.0 User Guide. [En línea] 2008. http://yasca.sourceforge.net.

9. **Naranjo Ortiz, Teudis y Sánchez Espinosa, Willian.** *Sistema de Análisis Estático de Vulnerabilidades en Python.* Universidad de las Ciencias Informáticas. Habana : s.n., 2010. pág. 146, Trabajo de Diploma.

10. **López Jaramillo, Yadira Milagros y Alfonso Guerra, Airam.** *Sistema de Análisis Estático de Vulnerabilidades. Módulo C++.* Universidad de las Ciencias Informáticas. Cuba : s.n., 2010. pág. 107, Trabajo de Diploma.

11. **Solórzano Hans, Araujo, Montalván Sócrates, Calsin y Reyes Helen, Requiz.** Metodologias De Desarrollo De Software. [En línea] http://www.buenastareas.com/ensayos/Metodologias-De-Desarrollo-De-Software/281090.html.

12. **Hernández Orallo, Enrique.** *El Lenguaje Unificado de Modelado (UML).* 2010. pág. 6.

13. **Visual Paradigm.** Sitio Oficial de Visual Paradigm. [En línea] 2010. http://www.visual-paradigm.com.

14. **Lira Turriza, Jose Luis.** *Algoritmos y Lenguajes de Programación.* Instituto Tecnológico Superior de Calkiní en el estado de Campeche. pág. 7.

15. **Python Software Foundation.** Sitio Oficial de Python. [En línea] 2011. [Disponible en: http://www.python.org/].

16. **Bell, Douglas y Parr, Mike.** *Entorno de desarrollo integrado página 11.* Tercera Edición. pág. 664. ISBN: 970-26-0144-4.

17. **Date, C.J.** *Introducción a los sistemas de base de datos.* Séptima Edición. Healdsburg, California : s.n., 2010. pág. 960. ISBN: 968-444-419-2.

18. **PosgreSQL-es.** Portal en español de PostgreSQL. [En línea] 2011. [Disponible en: http://www.postgresql-es.org].

19. **Ercoli , Jorge.** Arquitectura de Sistemas Informáticos. [En línea] 31 de octubre de 2007. http://metodologiasdesistemas.blogspot.com/2007/10/que-es-un-orm-object-relational-mapping.html.

20. **Pérez Mata , Manel.** TecnoRetales. [En línea] 3 de julio de 2009. http://www.tecnoretales.com/programacion/que-es-doctrine-orm/.

21. **Robles Elvira, Eduardo.** *Cifrado Punto a Punto en HTML5 con KHTML.* DEpartamento de Matemática Aplicada, Escuela Técnica Superior de Ingeniería Informática. 2010. pág. 108.

22. **Sánchez Rosas, Juan Eladio.** El Mundo es Open Source. *XMPP: Mensajería instantánea para organizaciones.* [En línea] 10 de febrero de 2009. http://blogs.antartec.com/opensource/2009/02/xmpp/.

23. **Vallejo Fernández, David.** *Documentación de ZeroC ICE.* Universidad de Castilla-La Mancha. Escuela Superior de Informática : s.n., 2006.

24. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_06/Conf_7/Materiales_complementarios/Introduccion_a_la_Disciplina_de_Requisitos.pdf.

25. —. Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_1/Conferencia_1/Materiales_basicos/Diseno.pdf.

26. **Veloso Hernández, Pedro.** *Uso de Patrones de Arquitectura.* 2010. pág. 37, ppt.

27. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.

28. *Modelo de Implementación:Diagramas de Componentes y Despliegue.* s.l. : Ingeniería del Software (3º I.T.I.S., I.T.I.G.). pág. 12.

29. **Sparks, Geoffrey, Systems, Sparx y Australia.** *Una Introducción al UML. El Modelo de Componentes.* 2010. pág. 12.

30. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Basicos/Sobre_la_disciplina_de_Prueba.pdf.

31. —. Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_10/Clase_Practica_6/Materiales_Basicos/Material_de_caja_b_y_caja_n.pdf.

32. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *Técnicas de Evaluación de Software.* 2006.

Bibliografía

1. **Filipiak, Joanna María.** *Herramienta de análisis estático de código para encontrar vulnerabilidades de seguridad en las aplicaciones Web.* Universidad Carlos III . Madrid : s.n., 2009. pág. 134, Proyecto Fin de Carrera.

2. **Expósito, Raúl;.** *¿Qué es el Análisis Estático de Código?* Madrid : s.n., 2009. pág. 19.

3. **Mazzarri, Milton.** Pylint: Análisis estático del código fuente en Python. [En línea] marzo de 2010. http://www.slideshare.net/milmazz/pylint.

4. **Hernández Yeja, Adrián y Tejas de la Cru, Yosbany.** *Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP.* Universidad de las Ciencias Informáticas. Cuba : s.n., 2010. pág. 180, Trabajo de Diploma.

5. **Culebro Juárez, Montserrat, Gómez Herrera, Wendy Guadalupe y Torres Sánchez, Susana.** *Software libre vs software propietario. Ventajas y desventajas.* México : s.n., 2006. pág. 170.

6. **Fortify.** Fortify 360. [En línea] https://www.fortify.com/downloads2/public/Fortify_360_Datasheet_Spanish.pdf.

7. Lista de las herramientas para el análisis estático del código. [En línea] 2011. http://www.worldlingo.com/ma/enwiki/es/List_of_tools_for_static_code_analysis.

8. **Yasca.** Yasca v1.0 User Guide. [En línea] 2008. http://yasca.sourceforge.net.

9. **Naranjo Ortiz, Teudis y Sánchez Espinosa, Willian.** *Sistema de Análisis Estático de Vulnerabilidades en Python.* Universidad de las Ciencias Informáticas. Habana : s.n., 2010. pág. 146, Trabajo de Diploma.

10. **López Jaramillo, Yadira Milagros y Alfonso Guerra, Airam.** *Sistema de Análisis Estático de Vulnerabilidades. Módulo C++.* Universidad de las Ciencias Informáticas. Cuba : s.n., 2010. pág. 107, Trabajo de Diploma.

11. **Solórzano Hans, Araujo, Montalván Sócrates, Calsin y Reyes Helen, Requiz.** Metodologias De Desarrollo De Software. [En línea] http://www.buenastareas.com/ensayos/Metodologias-De-Desarrollo-De-Software/281090.html.

12. **Hernández Orallo, Enrique.** *El Lenguaje Unificado de Modelado (UML).* 2010. pág. 6.

13. **Visual Paradigm.** Sitio Oficial de Visual Paradigm. [En línea] 2010. http://www.visual-paradigm.com.

14. **Lira Turriza, Jose Luis.** *Algoritmos y Lenguajes de Programación.* Instituto Tecnológico Superior de Calkiní en el estado de Campeche. pág. 7.

15. **Python Software Foundation.** Sitio Oficial de Python. [En línea] 2011. [Disponible en: http://www.python.org/].

16. **Bell, Douglas y Parr, Mike.** *Entorno de desarrollo integrado página 11.* Tercera Edición. pág. 664. ISBN: 970-26-0144-4.

17. **Date, C.J.** *Introducción a los sistemas de base de datos.* Séptima Edición. Healdsburg, California : s.n., 2010. pág. 960. ISBN: 968-444-419-2.

18. **PosgreSQL-es.** Portal en español de PostgreSQL. [En línea] 2011. [Disponible en: http://www.postgresql-es.org].

19. **Ercoli , Jorge.** Arquitectura de Sistemas Informáticos. [En línea] 31 de octubre de 2007. http://metodologiasdesistemas.blogspot.com/2007/10/que-es-un-orm-object-relational-mapping.html.

20. **Pérez Mata , Manel.** TecnoRetales. [En línea] 3 de julio de 2009. http://www.tecnoretales.com/programacion/que-es-doctrine-orm/.

21. **Robles Elvira, Eduardo.** *Cifrado Punto a Punto en HTML5 con KHTML.* DEpartamento de Matemática Aplicada, Escuela Técnica Superior de Ingeniería Informática. 2010. pág. 108.

22. **Sánchez Rosas, Juan Eladio.** El Mundo es Open Source. *XMPP: Mensajería instantánea para organizaciones.* [En línea] 10 de febrero de 2009. http://blogs.antartec.com/opensource/2009/02/xmpp/.

23. **Vallejo Fernández, David.** *Documentación de ZeroC ICE.* Universidad de Castilla-La Mancha. Escuela Superior de Informática : s.n., 2006.

24. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_06/Conf_7/Materiales_complementarios/Introduccion_a_la_Disciplina_de_Requisitos.pdf.

25. —. Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_1/Conferencia_1/Materiales_basicos/Diseno.pdf.

26. **Veloso Hernández, Pedro.** *Uso de Patrones de Arquitectura.* 2010. pág. 37, ppt.

27. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.

28. *Modelo de Implementación:Diagramas de Componentes y Despliegue.* s.l. : Ingeniería del Software (3º I.T.I.S., I.T.I.G.). pág. 12.

29. **Sparks, Geoffrey, Systems, Sparx y Australia.** *Una Introducción al UML. El Modelo de Componentes.* 2010. pág. 12.

30. **UCI.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Basicos/Sobre_la_disciplina_de_Prueba.pdf.

31. —. Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_10/Clase_Practica_6/Materiales_Basicos/Material_de_caja_b_y_caja_n.pdf.

32. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *Técnicas de Evaluación de Software.* 2006.

33. **León, Eduardo.** *Tutorial Visual Paradigm for UML.* 2010. pág. 25.

34. **Ordóñez Leyva, Yoanni.** *Estado del Arte. Vías de comunicación entre aplicaciones en Python.* Universidad de las Ciencias Informáticas. Cuba : s.n., 2010. pág. 8.

35. **de la Torre LLorente, César, y otros, y otros.** *Guia de Arquitectura N-Capas.* s.l. : Microsoft Ibérica S.R.L, 2010. pág. 443. ISBN:978-84-936696-3-8.

36. **Olivares Rojas, Juan Carlos.** *Patrones de Diseño.* México : s.n., 2010.

37. **Neyem, Andrés.** *Análisis y Diseño de Sistemas con UML.* Oficina 418 de Doctorado. 2007. pág. 21.

38. **Rodríguez, Luis.** *Herrmientas para análisis estático de seguridad: estado del arte.* Responsable laboratorio SW de ALS. CISSP. s.l. : III OWASP Spain Chapter Meeting, 2009.

39. **Software, Unidad Docente de Ingeniería del.** *Patrones del "Gang of Four".* Facultad de informática - Universidad Politécnica de Madrid. Madrid : s.n., 2010. pág. 57.

40. **Derake, José M.** *Programación concurrente y distribuida. ICE.* 2010. pág. 20.

41. **Barrios Dueñas, Joel.** *Como configurar vsftpd (Very Secure FTP Daemon).* 2007. pág. 4.

42. **Marzal, Andrés y García, Isabel.** *Introducción a la programación con Python.* Departamento de Lenguajes y Sistemas Inform´aticos, Universitat Jaume I. 2010. pág. 399.

43. **Craig, Larman.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : s.n. 970-17-0261-1.

44. **Hernández León, Rolando Alfredo y Coello González, Sayda.** *El Proceso de Investigación Científica.* Cuidad de La Habana : Editorial Universitaria, 2011. pág. 110. ISBN 978-959-16-1307-3.

45. **Fernández, Luis H.** Software Guisho. [En línea] 5 de mayo de 2009. Abstract Factory Pattern-Patrones.

46. **Ibarra, Armando F.** Rational Unified Process. [En línea] 15 de 02 de 2009. https://www.u-cursos.cl/ingenieria/2004/2/CC62C/1/material_docente/objeto/44990.

47. **González Duque, Raúl;.** *Python para Todos.* España : s.n., 2010. pág. 160.

48. **Anglada, Ramón Alexander.** *Desarrollo de aplicaciones de escritorio para gnu\linux. Python + QT.* La Habana : s.n., 09. pág. 9, Tutorial.

49. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* EE.UU : Pearson Adisson-Wesley.

50. **Letelier, Patricio.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/mod/resource/view.php?id=14106.

51. **Fernández Panadero, M. Carmen.** Entorno Virtual de Aprendizaje. [En línea] 2010. http://eva.uci.cu/mod/resource/view.php?id=14106.

52. **Mato García, Lic. Rosa María.** Diseño de Base de Datos. [En línea] http://eva.uci.cu/file.php/624/3._Bibliografia/Bibliografia_Basica/Libro_de_BD_de_Rosa_Maria.pdf.

53. **Alonso, Fernando, Martínez, Loic y Segovia, Javier.** *Introducción a la Ingeniería de Software.* MAdrid. España : s.n. ISBN: 84-96477-00-2.

54. **Sommerville, Ian.** *Ingenería del Sotware.* [ed.] Miguel Martín- Romo. Madrid.España : s.n., 2007. pág. 712. Vol. Séptima Impresión. ISBN: 84-7829-074-5.