Universidad de las Ciencias Informáticas

Facultad 7

TRABAJO DE DIPLOMA PARA OPTAR POR EL

TÌTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título: Desarrollo de un sistema para realizar el reconocimiento sintáctico del lenguaje Java

Autor: Emilio Suris Frómeta

Tutor:

Ing. Yurién Ricardo Fuentes Guerra

Ciudad de La Habana, Julio de 2010

“Año 52 de la Revolución”

Resumen:

El Grupo de Calidad del Centro de Informática Médica desarrolló una herramienta para realizar pruebas de Caja Blanca, la misma no cuenta con reconocedores sintácticos en su estructura, ocasionando una dificultad para los probadores del proyecto al hacer uso de dicho software. Implementar un sistema que realice el reconocimiento sintáctico del lenguaje Java fue la meta trazada y de esta manera garantizar el completo y correcto funcionamiento de la herramienta creada. Para el desarrollo del sistema a construir se hizo necesario emplear un generador de reconocedores sintácticos permitiendo analizar un archivo o un proyecto a decisión del probador.

Se logró devolver además un grupo de parámetros relacionados con el código analizado como por ejemplo su tamaño, la densidad de comentarios, el total de líneas en blanco, entre otros valores. Se realizó un estudio de las tecnologías más usadas relacionadas con el desarrollo de aplicaciones en este campo, a nivel internacional, nacional y en la Universidad de las Ciencias Informáticas evidenciándose la necesidad de implementar el sistema propuesto.

Se obtuvo como salida final un XML con la representación del código analizado en forma de árbol sintáctico, el cual se podrá utilizar para cualquier análisis y procesamiento de código fuente. De esta manera se apoyará el proceso de ejecución de pruebas de Caja Blanca dentro del Grupo de Calidad del CESIM.

Palabras claves:

Árbol sintáctico, código, herramienta, Java, reconocimiento sintáctico, sistema, XML.

Índice:

Introducción

Hoy en día se evidencia una sociedad comandada por las nuevas tecnologías, tanto es así que la informática desempeña un papel fundamental en todos los ámbitos a nivel mundial. A lo largo de la historia se han referenciado logros y descubrimientos que marcan la evolución del desarrollo tecnológico, uno de los mayores avances han sido los lenguajes de programación.

La informática como rama principal y los lenguajes de programación como subconjunto de dicha rama tienen una estrecha relación. Cuando se toma como referencia la informática hay una cierta asociación con los softwares que en ella se ponen de manifiesto. Lo mismo pasa con los softwares y los lenguajes de programación. En fin existe una proporcionalidad directa entre estos tres elementos.

El desarrollo de la informática como ciencia ha marcado cada rincón del planeta, por tal motivo se le atribuye una gran importancia en cualquier lugar donde se manifieste. Cada tecnología en la actualidad tiene que ver con los sistemas informáticos, el ahorro de la mano de obra, la eficiencia y la simplicidad son características que garantiza el uso de estos sistemas.

A medida que han pasado los años, la Revolución Cubana ha sido partícipe de la ola de avances tecnológicos y científicos que han revolucionado al mundo. El país no se ha quedado atrás con vista a integrarse plenamente a la infraestructura global de la información haciendo uso de las Tecnologías de la Información y las Comunicaciones (TIC).

A pesar de ser un territorio subdesarrollado y potentemente bloqueado no se encuentra ajeno a este progresivo auge, aprecia la importancia que tiene la ciencia y la tecnología para el desarrollo económico de la nación y constantemente adopta medidas para lograr dominarlas e introducirlas en cada sector de la sociedad.

A raíz del presente avance informático, en Cuba surgió la Universidad de las Ciencias Informáticas (UCI), donde se forman ingenieros informáticos que le brindan con sus conocimientos al país una gran ayuda. En la UCI se producen softwares para

exportarlos a otros países como también para el desarrollo interno de la isla, ayudando así a su avance en todos los sentidos.

Dentro de la UCI, el Centro de Informática Médica (CESIM) se caracteriza por ser un centro de excelencia dedicado al desarrollo de productos, sistemas, servicios y soluciones integrales para la salud con creatividad e innovación. Cuenta con profesionales integrales, calificados y de experiencia, siguiendo las buenas prácticas y las normas internacionales; logrando aplicaciones de alta calidad y competividad, para la optimización en el campo de la salud en Cuba y en otras áreas de referencia.

El Grupo de Calidad del CESIM se dedica al aseguramiento de la calidad de los sistemas que se desarrollan en el CESIM, con fines de comercializarlos internacional como nacionalmente. Como se había ejemplificado anteriormente; la estrecha relación entre software y lenguaje de programación, hace evidente que para el logro de la calidad de un determinado software sea necesario garantizar la calidad de los lenguajes de programación que se usa. Tanto PHP, C# como Java son los lenguajes empleados para desarrollar los sistemas en el CESIM.

Para el análisis del código fuente de cada software se realizan las pruebas de Caja Blanca las cuales se encargan de encontrar defectos en los mismos. Corrigiendo esos defectos se obtendrían tecnologías con mayor calidad.

En el año 2009 en el Grupo de Calidad se decidió realizar la confección de una herramienta para hacer pruebas de Caja Blanca (HAC) y de esa forma poder brindarle a cada software una mejor calidad a su código fuente. Esta herramienta no cuenta con la presencia de reconocedores sintácticos para los lenguajes de programación utilizados en el CESIM, lo cual implica una dificultad para el buen desarrollo y éxito total de la HAC creada.

Por la problemática anteriormente expuesta, se determina como problema a resolver: ¿Cómo realizar el reconocimiento sintáctico del lenguaje Java para conformar HAC?, trazando como objeto de estudio el proceso de reconocimiento sintáctico y como campo de acción, el reconocimiento sintáctico del lenguaje Java.

El objetivo principal de este trabajo es: desarrollar un sistema para el reconocimiento sintáctico del lenguaje Java.

Para cumplir con el objetivo trazado se han definido las siguientes tareas de investigación:

- Realizar un estudio de las herramientas similares existentes a nivel internacional, nacional y en la UCI, para llevar a cabo la construcción del sistema.
- Definir los parámetros del tamaño del código fuente que serán cubiertos.
- Definir el generador del reconocedor sintáctico a usar para construir la herramienta.
- Definir la gramática de Java, que se le introducirá al generador del reconocedor sintáctico.
- Definir una estructura para el objeto que representará el código analizado.
- Definir una arquitectura para la herramienta que se corresponda con la arquitectura de la herramienta que lo usará.
- Realizar un levantamiento de requisitos para el sistema.
- Realizar el análisis y diseño del sistema.
- Implementar el sistema y hacer las modificaciones necesarias para que cumpla con sus objetivos.
- Validar la correcta implementación de las funcionalidades mediante pruebas de Caja Negra.

El software ya terminado tendría una gran utilidad en el proyecto de “Calidad”, ya que la aplicación será parte de la herramienta HAC que realizaría las pruebas de Caja Blanca, así de cada programa que se manipule en él, será probado con exactitud su

código fuente y se obtendrán diferentes parámetros del código fuente analizado, relacionados con su tamaño, luego del análisis se devolverá un objeto con una estructura definida para que pueda ser interpretado por aplicaciones que requieran procesar código fuente a un alto nivel.

El presente trabajo está conformado por tres capítulos, a continuación se desglosa el cometido presente en cada uno:

El capítulo 1 **Fundamentación Teórica** incluye un estado del arte sobre los principales reconocedores sintácticos, a nivel internacional, nacional y de la Universidad, de las tendencias, técnicas, tecnologías, metodologías y software usados en la actualidad e incluso las que sirven de apoyo para la solución del problema identificado. Así como las definiciones necesarias para contribuir a un mejor entendimiento del lenguaje técnico empleado. Se establece un marco conceptual en correspondencia con la información que será manipulada por el sistema exponiéndose las principales funcionalidades del mismo, requerimientos, el objeto de automatización, así como el análisis comparativo con otras herramientas similares ya existentes.

Seguidamente el capítulo 2 **Análisis y diseño del sistema,** este capítulo se centra fundamentalmente en todo lo relacionado con la estructuración detallada y modelación de la aplicación. En el tercer y último, **Implementación y prueba del sistema,** se implementan las clases y procesos en término de módulo. Se llevarán a cabo pruebas de Caja Negra y así se validará la solución e implementación de las funcionalidades previstas. Se presenta la propuesta de solución para obtener los códigos fuentes con la calidad requerida.

Capitulo 1: Fundamentación Teórica

En este capítulo tendrá lugar la justificación a todo lo planteado en la introducción desde un punto de vista teórico, se abordarán temas relacionados con el problema científico, como son algunos conceptos que ayuden a su mejor comprensión. Se enfatizará lo respectivo a las soluciones que parcial o totalmente existan y brinden respuesta al problema científico tanto en el mundo, en Cuba como en la UCI.

Se profundizará en cuanto al objeto de estudio ofreciendo una exhaustiva descripción argumentando el por qué representa una zona vital para la realización de pruebas.

Dentro de este capítulo se manifestará también todo lo relacionado al entorno donde coexiste el negocio y la organización que rodean a la situación problémica.

La definición de una metodología de desarrollo con el objetivo de una organización total del trabajo viene siendo otros de los puntos a abordar. Se complementará la necesidad de utilizar un generador de reconocedores sintácticos para llevar a cabo la realización de la herramienta en sentido general, y por último una conclusión de lo abordado en el presente capítulo.

Conceptos asociados al dominio del problema

Para la mejor comprensión y análisis del problema se citarán algunos conceptos primordiales, y así poder lograr una mejor claridad en cuanto a la meta que se definió. Como primer concepto se encuentra: **lenguajes de programación** (LGP), estos son herramientas que nos permiten crear programas y software. Entre ellos se puede citar, el Delphi, C++, C#, Pascal, Java, etc.

Una computadora funciona bajo control de un programa el cual debe estar almacenado en la unidad de memoria; tales como el disco duro. Los LGP de una computadora en particular se definen como códigos de máquinas o lenguajes de máquinas. Estos lenguajes codificados en una computadora específica no podrán ser ejecutados en otra computadora.

Para que estos programas funcionen para diferentes computadoras hay que realizar una versión para cada una de ellas, lo que implica el aumento del costo de desarrollo.

Por otra parte, los LGP en código de máquina son verdaderamente difíciles de entender para una persona, ya que están compuestos de códigos numéricos sin sentido mnemotécnico.

Estos facilitan la tarea de programación, tanto es así que disponen de formas adecuadas que permiten ser leídas y escritas por personas, y a su vez resultan independientes del modelo de computador a utilizar.

Parámetros de codificación: Es una propiedad o características de las métricas, u orden o valor que se le pasa a la función y esta puede variar su comportamiento.

Métrica: "*Medida estadística que se aplica a todos los aspectos de calidad de software, los cuales deben ser medidos desde diferentes puntos de vista."* (1)

Código fuente: es un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos. Es un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de máquina mediante compiladores, ensambladores o intérpretes.

Análisis sintáctico: es el encargado de realizar la verificación de las distintas reglas de formación de un lenguaje, siendo los resultados de dichos análisis representados gráficamente a través de una estructura jerárquica o árbol sintáctico. Por medio de estos árboles se define si una expresión realmente pertenece a un lenguaje (2)

El análisis sintáctico es el punto de partida del Procesamiento del Lenguaje Natural (NLP por sus siglas en inglés) y sus aplicaciones, el análisis sintáctico se emplea como el paso inicial para la detección de primitivas conceptuales de UML.

Sus funciones son:

- Aceptar lo que es válido sintácticamente y rechazar lo que no lo es.

- Hacer explícito el orden jerárquico que tienen los operadores en el lenguaje de que se trate.

- Guiar el proceso de traducción (traducción dirigida por sintaxis).

Las construcciones sintácticas suelen requerir recursión. Las gramáticas de libre contexto (GLC) son una formalización de reglas recursivas que se pueden usar para guiar el análisis sintáctico.

Pruebas de Caja Blanca: También suelen ser llamadas estructurales o de cobertura lógica. En ellas se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico. Realizan un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose de manera concreta las instrucciones, bloques, etc. que han sido ejecutados por los casos de prueba.

Prueba de Caja Negra: se refieren a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, 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. Una prueba de Caja Negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software. (3)

Árbol de sintaxis abstracta (AST), organiza el código de forma estructural y jerárquica organizando y clasificando las construcciones de código según su tipo y composición. Adicionalmente se puede almacenar el texto de cada construcción, muy útil en análisis de estándares de codificación.

Gramática: se denomina así al conjunto de reglas y principios que gobiernan el uso de un lenguaje muy determinado; así, cada lenguaje tiene su propia gramática.

XML: permite definir la gramática de lenguajes específicos. Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para

diferentes necesidades. Se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Con la presentación de la definición de cada uno de los conceptos que se argumentan se llegaría a un mejor entendimiento de algunas de las características que se evidencian en el problema a resolver, que no viene siendo más que analizar sintácticamente el código de un lenguaje de programación dado (Java), permitiendo así la obtención de la solución concreta y reutilizable luego del proceso marcado.

Análisis de las soluciones existentes

En el mundo actual, la informática es un arma fundamental en la producción de software, diversas herramientas son construidas día a día en cada institución informatizada con la finalidad de ayudar y hacer mucho más eficiente el trabajo diario.

A continuación se ejemplifican algunas de las herramientas, las cuales fueron estudiadas al tener cierta similitud con la aplicación a construir:

JTest: Es el primer sistema automático de búsqueda de bugs (errores en el código de programación) para programadores de Java. Esta nueva tecnología desarrollada por la empresa ParaSoft, utiliza la Test Generation Technolgy (cuya patente está pendiente todavía) para analizar programas de Java. Se trata de una herramienta que permite realizar análisis de código, pruebas unitarias automáticas y cobertura de código, así como generación dinámica de pruebas funcionales.

En el ámbito de los análisis dinámicos JTest es capaz de generar automáticamente todas las pruebas unitarias que sean necesarias, teniendo en cuenta los parámetros de cobertura de código e intentando encontrar pruebas que deriven en errores de ejecución. Genera pruebas funcionales teniendo en cuenta las acciones y los datos, incluyendo peticiones HTTP (Protocolo de transferencia de hipertexto) y JDBC

(Interfaz de programación de aplicaciones entre los programas Java independiente de la plataforma y del gestor de bases de datos utilizado).

Se le han agregado varios módulos realizando revisiones de código más productivas y prácticas para las organizaciones, sin embargo se requiere de conocimientos técnicos para un mejor uso, con necesidad del usuario de recibir cursos de aprendizaje.

Resource Standard Metrics (RSM): Es una herramienta de métrica de código fuente y análisis de calidad para ANSI C, ANSI C++, C# y Java (Lenguajes de programación) para usar en todos los sistemas operativos Windows y UNIX. Es rápido y fácil de usar. RSM proporciona métrica estándar y análisis automatizado del código fuente y es válido para cualquier empresa que apunte a cumplir las certificaciones ISO-9001, TickIt y SEL (Organizaciones de normas o certificaciones de calidad). El código fuente que no compile no pasará correctamente por el analizador RSM. La misma es una herramienta basada en consola, esto implica que para ejecutarse debe usar una consola de comandos o cualquier consola UNIX o Xterm que desee. RSM utiliza un archivo de licencia y un archivo de configuración en el arranque.

JProbe Suite: Esta herramienta, permite detectar los 'puntos calientes' de los componentes de una aplicación JAVA, tales como el uso de la memoria, el uso del CPU, los hilos de ejecución y a partir de ellos, bajar al nivel del código fuente que los provoca ofreciendo una serie de consejos o buenas prácticas de codificación para la resolución del problema. Entre otros parámetros, se puede ver los métodos o líneas de código que más tiempo insumen en sus llamadas, los métodos con mayor cantidad de invocaciones, etc.

También permite medir el consumo de CPU por clases, métodos y paquetes, monitoriza el número de instancias de cada clase creadas, el consumo de memoria por clases, además de monitorizar el estado de los hilos de las aplicaciones, así como advertir de potenciales estancamientos y otros problemas relativos a sincronización. La parte de la suite que ahora es gratis es JProbe Profiler, disponible para descarga para Windows y Linux.

La Suite a completo puede descargarse con una licencia temporal para probarla. Los precios incluyen algunos servicios técnicos y para la mayoría de los productos disponibles para descarga, una copia de seguridad en línea y una actualización gratuita a la nueva versión si ésta se publica en un período de 30 días después de la compra. Todas las ventas están sujetas a sus términos y condiciones estándar y a su política de devolución. Los precios oscilan:

JProbe Suite V7.0 para Windows, Linux, HP, AIX y Solaris -- $ 1 950.00 a $ 6 825.00.

En la Ciudad de México se desarrolló un Software didáctico para la construcción de analizadores sintácticos descendentes no recursivos predictivos. El software se denomina RD-NRP cuyo objetivo es ayudar en el proceso de enseñanza-aprendizaje del tema “análisis sintáctico”. Permite la generación de código de una clase denominada SintDescNRP que es usada para definir objetos cuyo fin es analizar sintácticamente un grupo de sentencias. El trabajo termina mostrando la construcción de un analizador sintáctico no recursivo predictivo escrito en C#, que usa el código producido por los sofwares didácticos SP-PS1 y RD-NRP.

Entre los analizadores sintácticos que fueron tomados de prueba se apreció con el estudio de los mismos que ninguno formaría parte de la solución al trabajo de diploma; este último el software didáctico, se tiene la teoría pero no existe un acceso directo a la información en caso que se haya logrado construir. JTest, requeriría de una preparación total para su uso efectivo, y por el corto tiempo y la falta de información al respecto, viene dado el rechazo a esa herramienta. Algo parecido pasa con RSM ya que requiere de una licencia, y un archivo de configuración de arranque, y el JProbe Suite es demasiado costoso y no es viable para el objetivo que se traza el presente trabajo.

Por ende una gran importancia sobresale a través de dicho análisis y es precisamente la construcción del sistema para realizar el reconocimiento sintáctico del lenguaje Java y así se pondría de manifiesto el correcto funcionamiento de la HAC existente.

En el territorio nacional estos tipos de software que son los analizadores sintácticos no se han llevado a la práctica solo se ha quedado en hipótesis lo que hoy atribuye un

sentido común con la presente investigación. Un trabajo de diploma presentado en la universidad de Santiago de Cuba presentó una solución para interpretar un tipo de lenguaje, en dicho caso “el lenguaje cotidiano de Cuba”, no llegando todavía a lo necesario del proceso que se lleva en curso.

En la Universidad de las Ciencias Informáticas tampoco se ha visto un resultado concreto en cuanto a la fabricación del tipo de sistema que se encomienda. Solo sobresalta el ejemplo de tres profesores de la facultad 7, los cuales desarrollaron un Agente Semántico Inteligente para la Salud el cual quedó a medias, su alcance estuvo basado en teorías que nunca se llegaron a concretar.

Ya visto la variedad de analizadores sintácticos principalmente en el mundo exterior se puede llegar a la conclusión de que hay algunos que parcialmente cumplen o tiene cierta similitud a lo que significa la tarea principal del proyecto de software trazado, con inconvenientes ejemplificados. Ninguno sería ideal para darle respuesta a dicha tarea, brindando una exhortación a darle cumplimiento con un desarrollo satisfactorio al proceso que se induce a lo largo del presente documento.

Descripción actual del dominio del problema

La necesidad de hacerle pruebas al código fuente de cada software es una tarea que se ha puesto de manifiesto en el CESIM. Determinado que en él, se utilizan un gran número de ellos para el desempeño del buen desarrollo de sus metas: desarrollar sistemas, servicios y soluciones integrales para la salud, como se había ejemplificado en la introducción de este documento.

Al ser CESIM un centro de gran importancia para el sustento de las TICs en la salud cubana se hace necesario probar los softwares que de él provienen. La forma de hacer garantizable esta tarea es haciéndole pruebas a dichos softwares tanto funcional, como a su código fuente, dando lugar al éxito deseado. Específicamente se llevarán a cabo las pruebas de Caja Blanca, dirigidas al código de cada producto.

Actualmente en Grupo de Calidad del CESIM existe una herramienta dedicada a este tipo de funcionalidad “HAC”. Resaltándose la necesidad de reconocer un código como

objetivo principal, que puede ser tanto Java, PHP como C# como base fundamental, tomando como punto de referencia el código Java (el más usado). Sin lugar a duda este reconocimiento de código sería una parte que faltaría de dicha herramienta.

Un grado de importancia para la economía del país también se evidencia, pues poniendo en práctica buenos softwares al mercado; construidos por el Grupo de Calidad del CESIM, mejor prestigio y calidad tendrá la tecnología de Cuba ante el mundo actual.

Metodologías de desarrollo

Todo proceso de desarrollo de software es riesgoso y difícil de controlar. Si no se sigue una metodología, aparecen al final del proyecto clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. Con el propósito de desarrollar una aplicación que se ajuste a las características deseadas por el usuario y que, además, brinde solución a los problemas planteados por los clientes, se considera la metodología a utilizar como un elemento vital en el ciclo de desarrollo de software.

Existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Existen propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán.

XP (Extreme Programing o Programación extrema)

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código. Promueve los valores de comunicación directa entre las personas; el coraje de exponer sus dudas, miedos, experiencias sin "embellecer" estas de ninguna de las maneras; el intento de mantener el software lo más simple posible y la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino.

Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final del ciclo de la programación. (4)

El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

El ciclo de vida de XP se enfatiza en el carácter iterativo e incremental del desarrollo. Sus iteraciones son relativamente cortas ya que se piensa que entre más rápido se le entreguen resultados de valor al cliente, más retroalimentación se va a obtener, lo cual representará una mejor calidad del producto a largo plazo.

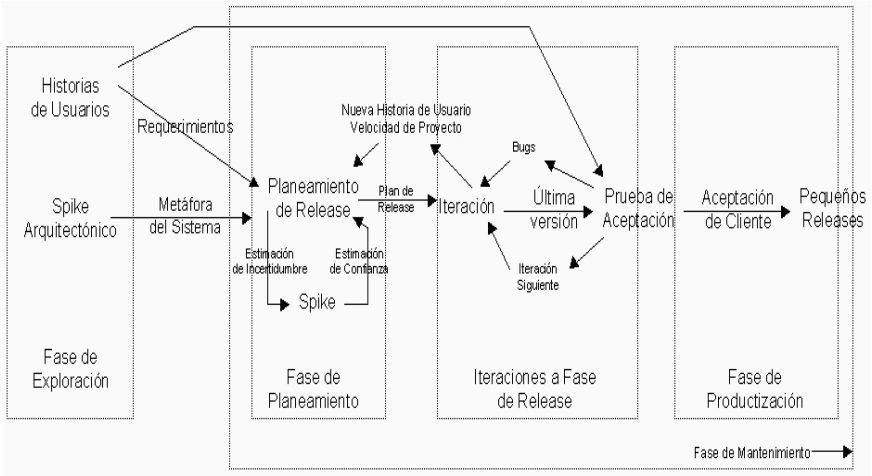

Figura 1: Vista de las fases de XP.

El RUP, es un proceso para el desarrollo de un proyecto de software que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto, con tres características esenciales, está dirigido por casos de uso; que orientan el proyecto a la importancia para el usuario y lo que este quiere, está centrado en la arquitectura; que relaciona la toma de decisiones que indican como tiene que ser construido el sistema y en que orden, y es iterativo y incremental; dividiéndose el proyecto en minis proyectos

donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada.

Fases del RUP dentro de un ciclo:

• Inicio. Se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis de negocio para el producto.

• Elaboración. Se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema.

• Construcción. Se crea el producto.

• Transición. El producto se convierte en la versión beta.

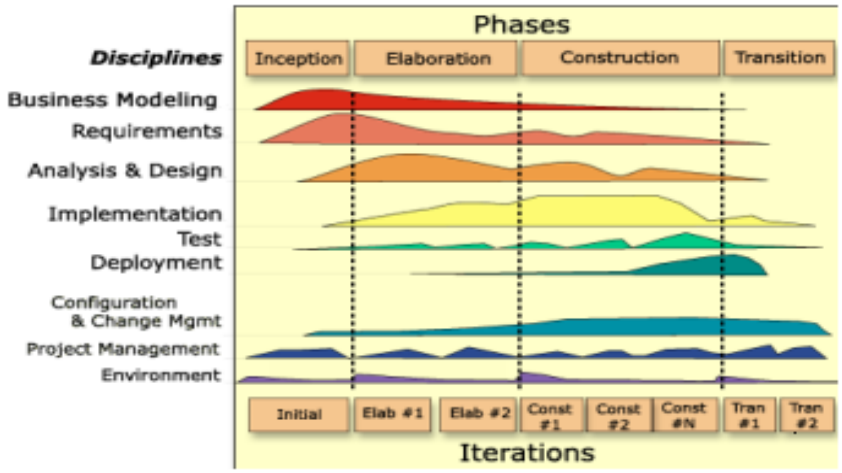

Figura 2: Fases de RUP.

Resultados de las fases en un ciclo:

• Inicio:

– Modelo de casos de uso simplificado.

– Esbozo de la arquitectura mostrando subsistemas más importantes.

– Se identifican y priorizan los riesgos más importantes.

– Se planifica en detalle la fase de elaboración.

– Se estima el proyecto de manera aproximada.

• Elaboración:

– Modelo de casos de usos, del análisis, del diseño, de implementación y de despliegue

– Plan de actividades y estimación de recursos para terminar el proyecto.

• Construcción.

– El producto con todos los casos de uso que la dirección y el cliente han acordado para el desarrollo de esta versión.

• Transición.

– Corrección de defectos.

Beneficios que aporta RUP:

* Permite desarrollar aplicaciones sacando el máximo provecho de las nuevas tecnologías, mejorando la calidad, el rendimiento, la reutilización, la seguridad y el mantenimiento del software mediante una gestión sistemática de los riesgos.

* Permite la producción de software que cumpla con las necesidades de los usuarios, a través de la especificación de los requisitos, con una agenda y costo predecible.

* Enriquece la productividad en equipo y proporciona prácticas óptimas de software a todos sus miembros.

* Permite llevar a cabo el proceso de desarrollo práctico, brindando amplias guías, plantillas y ejemplos para todas las actividades críticas.

* Proporciona guías explicitas para áreas tales como modelado de negocios, arquitectura Web, pruebas y calidad. También se proporciona guías para desarrollar en plataformas IBM WebSphere y Microsoft Web Solution para acelerar el desarrollo de los proyectos.

* Se integra estrechamente con herramientas Rational, permitiendo a los equipos de desarrollo aprovechar todas las ventajas de las características de los productos Rational, el Lenguaje de Modelado Unificado (UML) y otras prácticas óptimas de la industria.

* Unifica todo el equipo de desarrollo de software y mejora la comunicación al brindar a cada miembro del mismo una base de conocimientos, un lenguaje de modelado y un punto de vista de cómo desarrollar software.

* Optimiza la productividad de cada miembro del equipo al poner al alcance la experiencia derivada de miles de proyectos y muchos líderes de la industria.

* No solo garantiza que los proyectos abordados serán ejecutados íntegramente sino que además evita desviaciones importantes respecto a los plazos.

* Permite una definición acertada del sistema en un inicio para hacer innecesarias las reconstrucciones parciales posteriores.

Por lo tanto, en el trabajo arribó a la conclusión de utilizar RUP como metodología ya que brinda ventajas evidentes sobre XP, al darle énfasis a los requisitos y al diseño y basándose todo en las mejores prácticas que se han intentado y se han probado en el campo; como criterio principal de superioridad con respecto a XP. Con esta decisión se espera una mejor calidad del proceso de software a crear, y contar con menos vulnerabilidades en su final construcción.

Definición del Generador del reconocedor sintáctico a utilizar para la construcción del sistema

YACC: es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores

más". Genera un analizador sintáctico (la parte de un compilador que intenta darle sentido a la entrada) basado en una gramática analítica escrita en una notación similar a la BNF. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C.

Fue desarrollado por Stephen C. Johnson en AT&T para el sistema operativo Unix. Después se escribieron programas compatibles, por ejemplo Berkeley Yacc, GNU bison, MKS Yacc y Abraxas Yacc (una versión actualizada de la versión original de AT&T que también es software libre como parte del proyecto de OpenSolaris de Sun). Cada una ofrece mejoras leves y características adicionales sobre el Yacc original, pero el concepto ha seguido siendo igual. Yacc también se ha reescrito para otros lenguajes, incluyendo Ratfor, EFL, ML, Ada, Java, y Limbo.

Puesto que el analizador sintáctico generado por Yacc requiere un analizador léxico, se utiliza a menudo conjuntamente con un generador de analizador léxico, en la mayoría de los casos Lex o Flex, alternativa del software libre. La versión Yacc de AT&T se convirtió en software libre; el código fuente está disponible con las distribuciones estándares del Plan 9 y de OpenSolaris.

Yacc, puede tener tendencias a vulnerabilidades, tales como: desencadenar errores en cascadas y tener la característica de la ambigüedad en algunas reglas en específico, esta última viene dada por la entrada de cadenas de caracteres que se pueden interpretar de dos o más formas, y consecuentemente produce un analizador sintáctico con problemas en la gramática.

ANTLR: mucha gente llama a esta herramienta “compiladores de compiladores”, dado que ayudar a implementar compiladores es su uso más popular.

Sin embargo, tiene otros usos. ANTLR, por ejemplo, podría servir para implementar el intérprete de un fichero de configuración. ANTLR es capaz de generar un analizador léxico, sintáctico o semántico en varios lenguajes (Java, C++ y C# en su versión 2.7.2) a partir de unos ficheros escritos en un lenguaje propio. Dicho lenguaje es básicamente una serie de reglas EBNF y un conjunto de construcciones auxiliares.

Las reglas EBNF no son más que la forma de representar una gramática determinada, estas reglas son una extensión del lenguaje BNF, y este no es más que la descripción de cada elemento gramatical en función de otros más simples, a partir de precisos esquemas definidos, a partir de sus reglas de producción correspondiente.

La versión actual (2.7.7) data de noviembre de 2006. ANTLR es capaz de actuar a tres niveles a la vez (cuatro si se tiene en cuenta la generación de código), el uso de una sola herramienta para todos los niveles tiene varias ventajas. La más importante es la "estandarización": con ANTLR basta con comprender el paradigma de análisis una vez para poder implementar todas las fases de análisis. (5)

Como ventajas adicionales que diferencian a ANTLR de otras herramientas similares sirve de ejemplo: la posibilidad de generar el código de salida en diferentes lenguajes como Java, C, C++, C# o Python, y el hecho de disponer de un entorno de desarrollo propio llamado ANTLRWorks que nos permitirá construir de una forma bastante amigable las gramáticas de entrada a la herramienta, proporcionando representaciones gráficas de las expresiones y árboles generados, e incluyendo un intérprete y depurador propio. Por tal motivo se eligió, él servirá como soporte y herramienta fundamental para la generación del reconocedor; que vendría siendo una parte esencial del sistema a crear.

Plataformas

.NET, es una capa de software que se coloca entre el Sistema Operativo (SO) y el programador y que abstrae los detalles internos del SO. Las características fundamentales de esta plataforma son las siguientes:

- Portabilidad: Debido a la abstracción del programador respecto al SO, una aplicación .NET puede ser ejecutada en cualquier SO de cualquier máquina que disponga de una versión de la plataforma. En estos momentos la plataforma .NET tan solo está disponible para la familia Windows aunque se está desarrollando una versión para Linux de Corel.

- Multilenguaje: Cualquier lenguaje de programación puede adaptarse a la plataforma .NET y ejecutarse en ella.

- Interoperabilidad: La interoperabilidad entre diferentes trozos de código escritos en diferentes lenguajes es total.

Microsoft define la plataforma .NET como un entorno para la construcción, desarrollo y ejecución de servicios web y otras aplicaciones que consiste en tres partes fundamentales: el Common Language Runtime (entorno de ejecución), las Framework Classes (clases de la plataforma) y ASP.NET. Seguidamente se verán en detalle cada una de estas tres partes y sus características. El Common Language Runtime (CLR) es el entorno de ejecución de la plataforma .NET, y constituye su núcleo. El CLR es el entorno en el que se ejecutan nuestras aplicaciones .NET.

Estas aplicaciones pueden escribirse en cualquiera de los múltiples lenguajes que ofrece .NET (Visual C#.Net, Visual Basic.NET) que en lugar de compilarse a código máquina (que es lo más habitual) se compila a un lenguaje intermedio llamado Microsoft Intermediate Language o MSIL (Lenguaje Intermedio de Microsoft). El MSIL es el único lenguaje que el CLR comprende. Esta característica permite, por ejemplo, utilizar un fragmento de código en una aplicación sin depender del lenguaje en el que está escrito.

Las Framework Classes forman otra de las capas que constituyen la plataforma .NET. Esta capa provee al programador de servicios, estructuras y modelos de objetos para datos ADO.Net (siguiente generación de ADO), entrada/salida, seguridad, manejo de documentos XML, ASP.NET es la parte más importante de la capa superior de la plataforma .NET. Para los programadores web ASP.NET es mucho más que una nueva versión de la tecnología ASP ya que supone una nueva idea y forma de programar aplicaciones Web. ASP.NET provee una plataforma más robusta para el desarrollo de aplicaciones, y ofrece mayores beneficios.

A diferencia de ASP, los ASP.NET permiten separar limpiamente la lógica de la aplicación de la interfaz. De esta manera, el programador puede centrarse

exclusivamente en la lógica de la aplicación sin preocuparse de los detalles de la interfaz. ASP.NET además incorpora un nuevo concepto en el desarrollo de tecnologías Internet: los Servicios Web. Estos servicios representan un paso más hacia la descentralización del software en la red y de hecho, son un factor clave para el desarrollo de una web orientada a objetos.

Los servicios Web permiten a los desarrolladores construir aplicaciones combinando recursos locales y remotos para una solución distribuida e integrada. La comunicación a través de la web se hace utilizando el protocolo SOAP, lo cual no supone ningún problema para el desarrollador ya que es la plataforma .NET la que se encarga de tratarlo. (6)

Framework Mono, es la implementación libre del CLI (Common Language Infraestructure) y C#, de acuerdo a las especificaciones enviadas a la ECMA para su estandarización Esta implementación es de código fuente abierto (Open source).

El Mono incluye el CLI, el cual contiene la máquina virtual que se encarga de cargar las clases, el compilador Jit (Just-in-time) y el garbage collector; todo esto escrito desde cero de acuerdo a las especificaciones Ecma-334.

Mono también incluye un compilador de C#, el cual paradójicamente está escrito en C# y al igual que el CLI, este compilador sigue las especificaciones Ecma-335.

Adicionalmente Mono cuenta con un catalogo de librerías compatibles con las librerías del .Net Framework, pero además cuenta con una serie de librerías no existentes en el .Net Framework de Microsoft; como el GTK# que permite crear interfaces gráficas nativas del toolkit GTK+, Mono.LDAP, Mono.Posix, etc.

La motivación de crear Mono se debe a la búsqueda de herramientas que ayudaran a la creación rápida de aplicaciones en el entorno Linux.

Actualmente Mono se puede ejecutar en plataformas x86, PPC, SPARC y S390 en 32 bits; y x86-64 y SPARC en 64 bits; siendo posible crear y ejecutar aplicaciones en los sistemas operativos: Linux, Windows, OSX, BSD y Solaris.

Con esto se busca que un binario compilado en Windows con el .Net Framework pueda ejecutarse en alguna de las plataformas de Mono sin tener que recompilar el binario, y que a su vez pueda hacer uso de las librerías compatibles de Mono, ejemplos: System.Data, System.Xml, etc.

Bueno es posible crear aplicaciones de tipo Web y Webservices con el uso de modulo mod_mono que permite al servidor de Web Apache servir paginas de ASP.NET (aspx) y Servicios Web (asmx).

Es también posible crear aplicaciones que acceden a base de datos como Microsoft SQL, Oracle, Postgresql, etc.

Por el lado de aplicaciones de interfase gráfica, la sugerencia es utilizar GTK#, ya que el toolkit en el que este está basado (GTK+), permite ejecutar aplicaciones gráficas en ambientes Linux, Windows y OSX sin cambios; esta sugerencia toma importancia, debido a que la implementación compatible con Windows Forms en Mono aún no está completa.

Lenguajes de programación

C#:

Lenguaje de programación diseñado por Microsoft en 2001 como parte de su plataforma .NET. Combina el lenguaje de bajo nivel de C y la velocidad de la programación de alto nivel de Visual Basic. Actualmente se está utilizando con gran efectividad el nuevo lenguaje C#, como una recopilación de lo mejor de C++ y Java. Este lenguaje pertenece a la POO, tiene una sintaxis muy parecida a Java y posee la potencia de C++.

Tiene algunas ventajas sobre los restantes lenguajes de alto nivel orientados a objeto. El estándar del lenguaje C# por excelencia está comprendido en la especificación ECMA-334 de la Ecma International que hasta 1994 se llamó European Computer Manufacturers (7). Entre sus principales especificaciones están:

- El ECMA-262 como especificación para el lenguaje de programación ECMAScript.
- El ECMA-334 como especificación para el lenguaje de programación C#.
- El ECMA-335 como especificación para el CIL.

Java:

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos. La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva.

Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objetive C y Cedar/Mesa. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.

El lenguaje Java posee características fundamentales a continuación algunas de ellas:

- Lenguaje de propósito general.
- Lenguaje Orientado a Objetos.
- Sintaxis inspirada en la de C/C++.
- Lenguaje multiplataforma: Los programas Java se ejecutan sin variación (sin recompilar) en cualquier plataforma soportada (Windows, UNIX, Mac)

- Lenguaje interpretado: El intérprete a código máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM). El compilador produce un código intermedio independiente del sistema denominado bytecode.

- Lenguaje gratuito: Creado por SUN Microsystems, que distribuye gratuitamente el producto base, denominado JDK (Java Development Toolkit) o actualmente J2SE (Java 2 Standard Edition). API distribuida con el J2SE muy amplia. Código fuente de la API disponible.

C++:

Este lenguaje no es orientado a objetos puros (en el sentido en que puede serlo Java por ejemplo), además no nació como un ejercicio académico de diseño. Se trata simplemente del sucesor de un lenguaje de programación hecho por programadores (de alto nivel) para programadores, lo que se traduce en un diseño pragmático al que se le han ido añadiendo todos los elementos que la práctica aconsejaba como necesarios, con independencia de su belleza o purismo conceptual.

Estos condicionantes tienen su cara y su cruz; en ocasiones son motivo de ciertos "reproches" por parte de sus detractores, en otras, estas características son precisamente una cualidad. De hecho, en el diseño de la Librería Estándar C++ (5.1) se ha usado ampliamente esta dualidad (ser mezcla de un lenguaje tradicional con elementos de POO), lo que ha permitido un modelo muy avanzado de programación extraordinariamente flexible (programación genérica).

A continuación se presenta una serie de ventajas que posee el lenguaje de programación C# sobre algunos de sus homólogos.

Ventajas frente a C y C++:

* Compila a código intermedio (CIL) independiente del lenguaje en que haya sido escrita la aplicación e independiente de la máquina donde vaya a ejecutarse.

* Recolección de basura automática.

* Eliminación del uso de punteros, en C# no se necesitan.

* Capacidades de reflexión.

* No hay que preocuparse por archivos de cabecera ".h".

* No importa el orden en que hayan sido definidas las clases ni las funciones.

* No hay necesidad de declarar funciones y clases antes de definirlas.

* No existen las dependencias circulares.

* Soporta definición de clases dentro de otras.

* No existen funciones, ni variables globales, todo pertenece a una clase.

* Todos los valores son inicializados antes de ser usados (automáticamente se inicializan al valor estandarizado, o manualmente se pueden inicializar desde constructores estáticos).

* No se pueden utilizar valores no booleanos (enteros, fraccionarios, entre otros) para condicionales. Es mucho más limpio y menos propenso a errores.

* Puede ejecutarse en una sandbox restringida.

Ventajas frente a C++ y Java:

* Concepto formalizado de los métodos get y set, con lo que se consigue código mucho más legible.

* Gestión de eventos (usando delegados) mucho más limpia.

Ventajas frente a Java:

* El rendimiento es, por lo general, mucho mejor.

* CIL (el lenguaje intermedio de .NET) está estandarizado, mientras que los bytecodes de Java no lo están.

* Soporta bastantes más tipos primitivos (value types), incluyendo tipos numéricos sin signo.

* Indizadores que permiten acceder a cualquier objeto como si se tratase de un array.

* Compilación condicional.

* Aplicaciones multi-hilo simplificadas.

* Soporta la sobrecarga de operadores, que aunque pueden complicar el desarrollo son opcionales y algunas veces muy útiles.

* Permite el uso (limitado) de punteros cuando realmente se necesiten, como al acceder a librerías nativas que no se ejecuten sobre la máquina virtual. (8)

Se pudiera decir que el lenguaje C++ es muy potente y estable pero complicado para algunos propósitos. Java es muy similar al C# y su sintaxis es amigable pero eventualmente es menos eficiente en la ejecución de programas de escritorio frente a los desarrollados en C#. Además, este último aporta características novedosas que simplifican grandemente las labores de implementación. Se enriquece con la potencia de C++ y la sencillez y modernidad de Java pero mejorando lo que debe ser mejorado. Se usa en múltiples grupos de proyecto en la facultad 7 de la UCI. Es gratis tanto el C# como la plataforma que lo soporta.

Entorno de desarrollo

Un Entorno de Desarrollo Integrado, Integrated Development Environment (IDE) es un programa que agrupa un conjunto de herramientas que viabilizan la labor de los programadores. Los componentes esenciales que los definen son:

- Editor de texto (código).
- Compilador.
- Intérprete.
- Depurador.

- Herramientas de automatización (completamiento de código y navegación rápida dentro del código).
- Sistema de control de versiones.
- Diseñador de interfaces gráficas.
- Agregación de herramientas externas.
- Soporte para varios lenguajes de programación y plataformas de desarrollo.

No todos poseen las anteriores propiedades ni las implementan al mismo nivel, pero esas son las más usuales, por tal motivo seguidamente se ejemplificaran varios IDE, que generen aplicaciones en lenguaje C# para la plataforma .NET:

Microsoft Visual Studio:

Es un IDE para sistemas Windows desarrollado por Microsoft Corporation. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio 2005 permite a los desarrolladores crear sitios, aplicaciones y servicios web y aplicaciones de escritorio en cualquier entorno que soporte la plataforma. Sobre el mismo se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles (9).

Visual Studio .NET continúa siendo el punto de referencia para la productividad de los programadores. Con un único entorno de programación (IDE) integrado para todos los lenguajes, las organizaciones de programación pueden aprovechar las ventajas de un cuadro de herramientas, un depurador y una ventana de tareas comunes, reduciendo enormemente la curva de aprendizaje del programador y garantizado que siempre puedan elegir el lenguaje más apropiado para sus tareas y conocimientos.

Con la función para completar instrucciones de IntelliSense y la comprobación automática de errores de sintaxis, Visual Studio .NET informa a los programadores cuándo el código es incorrecto y proporciona el dominio inmediato de las jerarquías de clases y las API. Con el Explorador de soluciones, los programadores pueden reutilizar

fácilmente código a través de diferentes proyectos e incluso, generar soluciones multilenguaje que satisfagan con mayor eficacia sus necesidades empresariales. (10)

SharpDevelop:

Es un IDE desarrollado por ICSharpCode Team que soporta el desarrollo de aplicaciones escritas en C#, Visual Basic.NET y BOO. Está desarrollado en C# y es libre. Proporciona todas las características necesarias para un entorno de programación de Windows, entre ellas:

- Autocompletado de código: Esta funcionalidad simplifica el esfuerzo y tiempo de codificación ya que el programador va escribiendo el inicio de las palabras y el editor de texto, ya sea automáticamente o mediante teclas de acceso rápido, despliega un menú contextual con las diferentes palabras que se asocian al lexema que se ha escrito y al contexto en que lo hace.

- Plantillas de proyectos: Estas plantillas son programas ya comenzados con gran parte de su encabezamiento y primeros elementos especificados y que el programador no tendrá que escribir reduciendo tiempo y esfuerzo de codificación.

- Depurador integrado: Esta es una de las ventajas que más se esperan de un buen IDE ya que permite probar el funcionamiento paso a paso de la sección del programa que se desee. Basta con poner los puntos de ruptura en los sitios escogidos y ordenar la depuración dando la orden a cada paso pudiendo observar los valores que van tomando las variables y pasar por alto llamadas a funciones o insertarse en la ejecución de las mismas. Esto es muy útil para probar algoritmos complejos y da la posibilidad de encontrar un error en mucho menos tiempo que analizando el algoritmo mentalmente.

- Diseñador de formularios: El diseñador de formularios permite conformar la interfaz visual de las aplicaciones de forma rápida y real porque la persona que diseña va viendo la apariencia de su diseño a medida que incorpora y ajusta elementos en la ventana. Paralelo a esto, la herramienta va generando el

código fuente correspondiente a cada componente. De esta forma también se ahorra tiempo y esfuerzo.

- Tiene herramientas para ir a definición, encontrar referencias y renombrado que permite desplazarse rápidamente por el código yendo a los puntos deseados con solo dar clic derecho y escoger una de estas opciones. La primera permite desde cualquier invocación a método o variable, ir a su definición. La segunda permite lo contrario, desde el nombre de la función o variable, encontrar todas sus invocaciones hechas en el código. La tercera permite que al cambiar el nombre del identificador de alguno de estos elementos, se realicen los cambios automáticamente a todas las ocurrencias de los mismos.

- Integración con herramientas externas: Muy utilizado a la hora de incorporar aplicaciones principalmente que se dedican a la administración y pruebas de código.

- Pose analizadores para ensamblado.

SharpDevelop es compatible con Visual Studio Express y Visual Studio 2005 por lo que es posible trabajar con proyectos indistintamente en uno u otro ambiente sin cambiar el formato de archivos de proyecto y de código. Su objetivo es brindar una opción para desarrolladores que no tienen la posibilidad de adquirir el Visual Studio.

Soporta además los siguientes lenguajes: HTML, ASP, ASP.NET, VB Script y XML con sintaxis destacada (escritura de las palabras en diferentes colores en dependencia de su tipo dentro del lenguaje) para todos ellos. Es capaz de convertir código C# a Visual Basic .NET y viceversa y de cualquiera de estos dos hacia Boo. Permite desarrollar proyectos para las siguientes plataformas: Microsoft .NET Framework 1.1 y 2.0, Microsoft .NET Compact Framework 1.0 y 2.0 de la familia .NET; Mono 1.1 y 2.0 de la familia Mono.

Se puede llegar a la conclusión que SharpDevelop es una alternativa libre y gratuita para realizar programas para la plataforma .NET. Está totalmente basado en el

ambiente de trabajo de Visual Studio. Posee las características básicas que se necesitan para caracterizarlo como un moderno entorno de desarrollo de aplicaciones de propósitos múltiples.

La herramienta a desarrollar es una aplicación típica de escritorio que no posee conexiones a bases de datos. Su función se basa en procesar grandes cantidades de código fuente de manera eficiente.

Se arribó a la conclusión de que para estos propósitos era factible utilizar SharpDevelop 2.2.1.2648 porque brinda las posibilidades que se requieren para un programa de este tipo. Su descarga desde Internet es gratuita y se instala de forma rápida en la computadora. Es libre y muy parecido al Visual Studio en cuanto al ambiente de trabajo.

De esta forma, una vez analizadas las ventajas y desventajas de todas las posibles combinaciones de lenguaje, plataforma, IDE. Se decidió realizar la implementación mediante el conjunto formado por el lenguaje C#, la plataforma .NET y SharpDevelop como IDE.

Sistema de control de versiones

Una versión de un producto: es el estado en que se encuentra en un momento dado de su desarrollo o modificación. Se llama control de versiones a la gestión de los diversos cambios o configuraciones que se realizan sobre los elementos de algún producto. (11)

Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, controlando el acceso a ficheros que están bajo su cuidado. El código fuente de un programa necesita controlarse en cuanto a su almacenaje, posibilidad de realizar cambios, registro histórico de las acciones realizadas y, tal vez, la generación de informes que traten sobre el estado y los cambios introducidos entre las dos versiones.

Todo es posible hacerse a través del uso de un sistema de control de versiones. Opera generalmente con una copia maestra en un repositorio central y un programa cliente con el que cada usuario sincroniza su copia local. Permitiendo compartir los cambios sobre un mismo conjunto de ficheros y guardar el registro de los cambios realizados por cada usuario, admitiendo así el retorno seguro al estado anterior en caso de necesidad.

Constituye una garantía de seguridad y una gran utilidad, contar en el proyecto con un sistema de control de versiones. Ilustrando en la práctica, se verá muy frecuente la necesidad de un sistema control de versiones. Cuando se está escribiendo un programa o un documento muy largo (como un informe o una tesis) y se quiere hacer una marca en el proyecto al llegar a un punto estable, éste permite guardar una versión estable del trabajo.

La idea de guardar versiones es, que si los nuevos cambios están mal, la marca servirá para volver al punto estable y recomenzar desde ahí. Si los cambios están bien y se alcanza una nueva estabilidad, se sitúa otra marca. Para el control de versiones de esta herramienta se utilizará Subversión (SVN). El Concurrent Versions System (CVS) es el padre de los controladores de versiones, pero el mismo fue reemplazado por SVN debido a que el primero posee varias deficiencias. Por tanto el SVN es el indicado para un desarrollo con alta competitividad y calidad, destacando en el mismo:

- Es software libre bajo una licencia de tipo Apache/BSD.
- A diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.
- Permite selectivamente el bloqueo de archivos.
- Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).

- Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos).
- La creación de ramas y etiquetas es una operación más eficiente. Tiene costo de complejidad constante (O (1)) y no lineal (O(n)) como en CVS.
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- Se sigue la historia de los archivos y directorios a través de copias y renombrados. (12)

Procesamiento de código fuente

A medida que transcurre más el tiempo de vida de las tecnologías y con ellas las ciencias informáticas relacionadas estrechamente; los software que se utilizan para satisfacer las disímiles necesidades del mundo actual son cada vez más complejos. Por ende la programación de estos software mencionados va siendo cada vez de más alto nivel, es decir, del código fuente, entonces surge la necesidad de procesarlo, hacerle pruebas.

Las pruebas que se le hacen al código de un software en sí, se le denominan "**Pruebas de Caja Blanca**", en ellas, se pretende indagar sobre la estructura interna del código, omitiendo detalles referidos a datos de entrada o salida. Su objetivo principal es probar la lógica del programa desde el punto de vista algorítmico.

La prueba de Caja Blanca se basa en el diseño de Casos de Prueba que usa la estructura de control del diseño procedimental para derivarlos. Existen diferentes tipos de pruebas de Caja Blanca:

- De Estructura de Datos Locales
- De Cobertura Lógica
- Cobertura de Caminos
- De bucles

– De condición

Las pruebas de Caja Blanca se llevan a cabo en primer lugar, sobre un módulo concreto, para luego realizar las de Caja Negra sobre varios subsistemas (integración).

La aplicación a construir contaría con tres parámetros de codificación que ayudarían a los probadores del proyecto a evaluar cada fragmento de código en respuesta a la calidad del software.

En los sistemas orientados a objetos, las pruebas de Caja Blanca pueden aplicarse a los métodos de la clase, pero según varias opiniones, ese esfuerzo debería dedicarse a otro tipo de pruebas más especializadas (un argumento podría ser que los métodos de una clase suelen ser menos complejos que los de una función de programación estructurada). Este concepto también es utilizado de manera análoga en la teoría general de sistemas. (13)

Principales funcionalidades del reconocedor sintáctico

El reconocedor sintáctico a confeccionar debe cumplir con diferentes funcionalidades, hay que tener en cuenta para su mejor comprensión que todo parte de una entrada definida de código Java, de ahí a que especificara entonces tres de esas funciones como puntos de referencia para su estudio:

Extraer secuencia de token, una de las clases que conformará el sistema, será la encargada de extraer, de conformar cada token a partir del código fuente introducido a la aplicación. La finalidad de dicha función viene dada ya que una vez completada se obtendrían todas las partes del código en fragmentos y así se facilitaría la posterior funcionalidad que es **la construcción del árbol de estructura de clase (AST).**

Ya con los token que no son más que los elementos más básicos sobre los cuales se desarrolla toda traducción de un programa, se construirá el árbol sintáctico. El cual organiza el código de forma estructural y jerárquica, clasificando las construcciones de código según su tipo y composición. Dando paso entonces a la tercera de las funciones que es **exportar XML que representa el árbol AST.**

Este requerimiento se pone de manifiesto a partir de tener el árbol de sintaxis abstracta formado, tanto de un solo archivo Java como de varios archivos cargados a la vez. Estos últimos se verificarían uno a uno, entonces se exportará un XML correspondiente a cada fichero. Que este no es más que una manera de definir lenguajes para diferentes necesidades. Ya obtenido el XML, se podría reutilizar el resultado, que sería el árbol exportado en XML, y así garantizar un trabajo futuro con dicho estándar de codificación más fácil y seguro.

Al árbol que da como resultado el reconocedor sintáctico, se hace necesario hacerle pruebas, que tienen que ver con el tamaño en sí, del software analizado convertido en árbol, pudiendo entonces determinar tres parámetros de código, que utilizan dicho resultado:

Halstead:

Las métricas de Halstead (14) son un conjunto de medidas primitivas que determinan el tamaño del software asumiendo que el programa está compuesto por un conjunto de elementos que se clasifican en operadores u operandos.

Propone cuatro parámetros básicos:

- n1: Es la cantidad de operadores diferentes en el código. Operadores son todos los elementos que realizan acciones sobre los operandos y los operandos son los elementos que contienen información en el programa.
- N1: Es el número total de ocurrencias de operadores en el programa.
- n2: Es la cantidad de operandos diferentes.
- N2: Es la cantidad de ocurrencias de operandos.

A partir de estos cuatro parámetros básicos, mediante operaciones matemáticas se obtiene otros valores:

- N: Se define como el tamaño o longitud en palabras de un programa que representa la cantidad total de operadores y operandos que conforman el programa. Se define como:

N = N1 + N2.

- n: La suma de operadores y operandos representa el vocabulario de un programa y se define como:
  n = n1 + n2
- L: Es la longitud estimada de un programa se define como:
  L = N1 * ln (n1) + N2 * ln (n2)
- V: Es el volumen de un programa que representa el volumen en bits necesarios para el programa y depende del lenguaje de programación. Se define como:
  V = N * ln(n)
- E: Es el esfuerzo necesario que debe realizar un programador, teniendo en cuenta el número de discriminaciones mentales elementales, para implementar un programa. Se define como:
  E = n1 * N2 * N * ln(n) / (2 * n2)

Líneas de código:

La cantidad de líneas de código es el parámetro más usado para interpretar el tamaño de un programa. No existe una definición exacta de qué se considera realmente una línea de código pudiendo excluir o incluir las líneas de comentario, las líneas declarativas y las líneas en blanco. En este trabajo se toman como líneas de código todas las líneas del programa incluyendo los casos antes mencionados.

- L.Dec: Representa las líneas declarativas que son aquellas que no ejecutan acciones, se usan solo para declarar elementos y cabeceras.
- L.Pro: Representan las líneas procedurales que son aquellas que no son declarativas ni son Líneas en Blanco ni Líneas de Comentarios.
- L.Blac: Representa las líneas en blanco que son aquellas que no tienen ningún elemento.
- L.Com: Representa las líneas de comentarios que son aquellas que solo tienen uno o varios elementos de comentarios.
- C.Lin: Es la suma de todos los parámetros anteriores y representa la cantidad total de líneas de código del programa. Esta forma de determinar las líneas de

código da la posibilidad de hacer modificaciones en el futuro incluyendo o excluyendo alguna de las variantes.

- C.Com: Es la cantidad de ocurrencias de comentarios independientemente de las líneas en que aparezcan. Es un parámetro útil para conocer el grado de comentarios del código.
- D.Com: Es la densidad de comentarios o la proporción de comentarios por líneas de código que da la medida de cuan comentado está el código. (15)

Li – Henry:

Las métricas de Li- Henry (16) interpretan el tamaño de un programa de dos maneras diferentes de las anteriores expuestas y diferentes entre sí.

- Tamaño1: Es la cantidad total de punto-y-coma (“;”) en una clase.
- Tamaño2: es la cantidad de atributos locales y métodos en una clase.
- NML: Es el Número de Métodos Localmente Definidos y Heredados en una clase, es la cantidad de métodos de una clase incluyendo los definidos localmente y los heredados en los casos que las clases sean hijas de otras.

Una vez conocida las funciones del sistema a realizar, se logrará la confección de un trabajo más amigable al llevar a cabo las pruebas de Caja Blanca en el HACC que se está desarrollando, teniendo el XML con el árbol sintáctico, localizar la parte a buscar del código a probar seria instantáneamente resuelta sin la necesidad de rastrear un código en su totalidad. Se logrará saber el valor de cada parámetro de código en cada uno de los casos tratados anteriormente, por tal motivo se tendría, una idea del tamaño del software procesado, y algunos valores que encierran sus características internas.

Dirigido a ejemplificar y argumentar toda la teoría del trabajo de diploma estuvo dedicado el Capítulo 1, enfatizando en la determinación de distintos conceptos de aspectos que tienen que ver con el problema a resolver.

Quedaron ejemplificadas las características de: las principales metodologías de desarrollo de software, lenguajes, entornos de desarrollo integrado, sistema de control de versiones y tecnologías o frameworks de persistencias existentes. Haciendo una caracterización profunda, comparando y seleccionando en cada caso la más adecuada para utilizarla en el proceso de desarrollo del software y lograr de esta forma obtener un producto con los costes de tiempo y calidad requeridos.

Se realizó un estudio del arte de los numerosos reconocedores sintácticos que existían en el mundo, en Cuba y en la universidad demostrando que la construcción del reconocedor sintáctico es una gran necesidad.

Se señaló lo referido al procesamiento de código fuente, brindando una amplia panorámica de lo que significaba analizar un programa como tal. Al igual que se percibió la realidad de las necesidades previstas por el Grupo de Calidad del CESIM a fin de conseguir una eficiencia total en su labor interna.

Se trabajó utilizando las referencias bibliográficas necesarias para garantizar la autenticidad de la información expuesta en el mismo, así el lector podrá auxiliarse de las diferentes fuentes de información consultadas en caso de que requiera mayor información acerca del tema expuesto.

Capítulo 2: Análisis y Diseño del Sistema

En el presente capítulo se concibe la descripción y el diseño de la propuesta que trae este trabajo de diploma, para ello se elaborará el Modelo de Dominio, para mayor comprensión del contexto en que se ubica el sistema. Se definirán conceptos que serán agrupados en dicho modelo, es decir, elementos, clases que intervienen para capturar correctamente los requisitos y poder construir un sistema correcto.

Se especificarán los casos de usos del sistema, y se modelará el diagrama de casos de uso del sistema ejemplificando el flujo de información en la herramienta a construir, como también tendrá lugar la visualización de los diagramas de clases del análisis, y los diagramas de interacción; colaboración y secuencia, de cada caso de uso.

Se verá todo lo referente al diseño de clases en forma de diagramas para mayor comprensión de las funcionalidades de cada clase del proyecto, explicándose así las principales de ellas y poner de manifiesto el papel importante de la futura aplicación.

Modelo de Dominio

El modelo de dominio muestra las clases conceptuales significativas en el dominio del problema, captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará la herramienta. Es considerado un subconjunto del llamado modelo de objetos del negocio.

Durante el desarrollo del software no se pudo contactar procesos bien definidos en el entorno del negocio. Se hizo difícil determinar los elementos más importantes del sistema y sus interconexiones, así como el establecimiento de las reglas de funcionamiento. Sin embargo, se pueden identificar eventos, transacciones y objetos involucrados en ese entorno que no está bien delimitado, por lo que se hizo necesario un modelado del domino perteneciente a la solución, mostrado en la figura 3.

Esto *"ayuda a los usuarios, a utilizar un vocabulario común para poder entender el contexto en que se ubica el sistema. Y para capturar correctamente los requisitos y poder construir un sistema correcto se necesita tener un firme conocimiento del*

funcionamiento del objeto de estudio. Este modelo va a contribuir posteriormente a identificar algunas clases que se utilizarán en el sistema." (17)

Descripción de los conceptos fundamentales

Se denominará **fichero** o archivos informáticos a un conjunto de bits almacenado en un dispositivo periférico. Facilitan una manera de organizar los recursos usados para almacenar permanentemente datos en un sistema informático.

Se considera **parámetros del código** a la propiedad o característica de una métrica, u orden o valor que se le pasa a la función y ésta variará su comportamiento.

Se considera **estructura de código** a la porción de texto escrito generalmente por una persona, se utiliza como base para generar otro código que posteriormente será interpretado o ejecutado por una computadora. Es el código asignado a cada una de las estructuras del mismo, ya sean los ciclos (For, While) o las bifurcaciones (if ó Swich). Normalmente, se refiere a la programación de software. Que este será copiado o cargado desde la plataforma en la que el usuario esté trabajando.

Sentencia de código se denomina a cada línea de código que forma parte de un proyecto de software.

Un **Token** es el elemento más básico sobre el cual se desarrolla toda traducción de un programa, surge en la primera fase, llamada "análisis léxico".

Diagrama del modelo de dominio

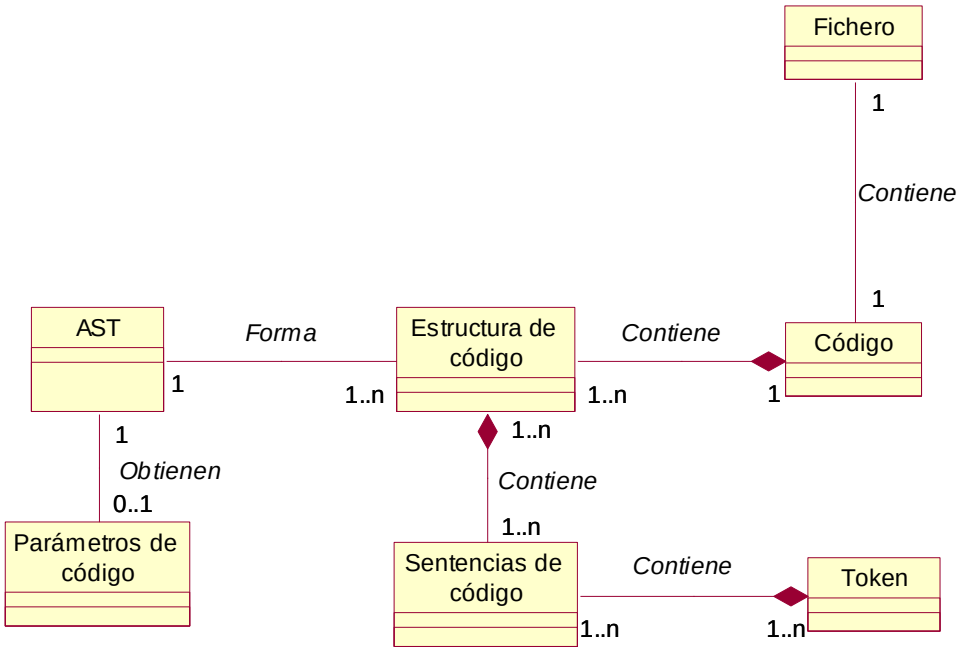

Figura 3: Diagrama del modelo de dominio de la aplicación.

Especificación de los requisitos del software

Los requerimientos del sistema es uno de los aspectos más importantes a considerar cuando se desarrolla un software, pues estos constituyen la base, el fundamento de la solución propuesta y el argumento para desarrollar el modelado del sistema. Por eso la captura de los requerimientos es uno de los procesos críticos de la ingeniería del software.

Requisitos funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, especifican acciones que el sistema debe ser capaz de realizar. Ellos deben de ser comprensibles por los clientes, usuarios y desarrolladores, deben tener una sola interpretación y estar definidos en forma medible y verificable.

Es por ello que una vez conocido los conceptos que rodean al objeto de estudio, se analiza ¿Qué debe hacer el sistema para que se cumplan los objetivos planteados al

inicio del trabajo de diploma?, enumerándose a través de requerimientos funcionales las instrucciones que el sistema deberá ser capaz de efectuar. Dentro de ellos se incluyen las acciones que podrán ser ejecutadas por el usuario, las acciones ocultas que debe realizar el sistema, y las condiciones extremas a determinar por el sistema.

A continuación se muestran los requisitos que debe cumplir el sistema:

RF1 _Formar Árbol Sintáctico.

- RF 1.1_Cargar archivo o directorio
- RF 1.2_Extraer secuencia de token.
- RF 1.3_Clasificar cada construcción de código según su tipo y composición.

RF2 _Determinar parámetros de código.

- RF 2.1 _Obtener valores del parámetro Li_Henrry de forma visual.
- RF 2.2 _Obtener valores del parámetro Líneas de código de forma visual.
- RF 2.3 _Obtener valores del parámetro Halstead de forma visual.

RF3 _ Exportar XML con la representación del árbol sintáctico y los valores de los parámetros.

Casos de uso del sistema

El modelado de Casos de Uso es la técnica más efectiva y a la vez la más simple para modelar los requisitos del sistema desde la perspectiva del usuario. Los Casos de Uso se utilizan para modelar cómo funciona un sistema o negocio, o cómo los usuarios desean que funcione el futuro sistema. No es realmente una aproximación a la orientación a objetos; es una forma de modelar procesos. Es una manera muy buena de dirigirse hacia el análisis de sistemas orientado a objetos. Los casos de uso son generalmente el punto de partida del análisis orientado a objetos.

CU-1	Formar Árbol Sintáctico
Actor	Probador

Descripción	El probador, carga un archivo Java de forma única o a través de un directorio de carpetas, luego el sistema analiza el fichero, extrayendo la secuencia de token correspondiente, y a la vez forma el árbol sintáctico clasificando cada construcción de código según su tipo y composición. Finalmente, se visualiza el árbol construido.
Referencia	RF 1.1, 1.2, 1.3

CU-2	Determinar parámetros de código.
Actor	Probador
Descripción	Una vez que el probador cargue un archivo Java ya sea de forma única o mediante un directorio, y que esté visualizado el árbol sintáctico, el probador da clic derecho sobre el fichero que desee conocer los parámetros de código; ya sea Li_Henrry, Líneas de código (LOC) o Halstead, el sistema visualiza el resultado.
Referencia	RF 2.1, 2.2, 2.3

CU-3	Exportar XML.
Actor	Probador
Descripción	Una vez que se haya visualizado el árbol sintáctico, el probador da clic derecho sobre este y accede a la opción exportar el XML. El sistema te permite además salvarlo en cualquier parte de la computadora y muestra finalmente un mensaje indicando que la operación tuvo éxito.

Referencia	RF 3

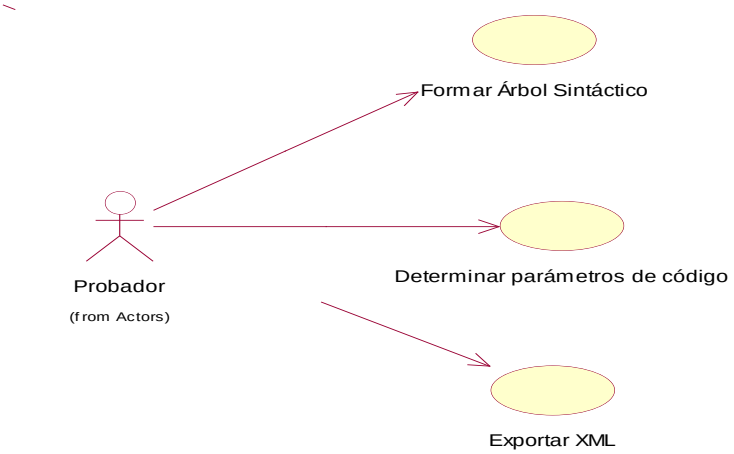

Figura 4: Diagrama de casos de uso del sistema.

Modelo del Análisis

Este modelo es empleado con el fin de representar la estructura global del sistema, sirve como una abstracción del Modelo de Diseño. Es un primer intento por definir los conceptos claves que describen el sistema. Este artefacto es opcional, pero también tiene a su vez la propiedad de ser temporal. Su utilidad radica en que permite un acercamiento visual del sistema. (18)

Diagrama de Clases de Análisis

El Diagrama de Clases del Análisis es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Está compuesto por clases y sus relaciones. Las clases del análisis se centran en los requerimientos funcionales, representan conceptos y relaciones del dominio. Estas poseen atributos y entre ellas pueden existir relaciones de asociación, agregación / composición, generalización/especialización.

RUP propone varias clasificaciones para estas clases, entre las que se destacan:

- **Clase Interfaz (CI):** Modela la interacción entre el sistema y sus actores.
- **Clase Controladora (CC):** Coordina la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.
- **Clase Entidad (CE):** Modela información que posee larga vida y es a menudo persistente.

La herramienta Rational Rose (19), representa los tipos de clases a los que se ha hecho referencia de la siguiente forma:

Figura 5: Representación de los tipos de clases del Análisis.

A continuación se muestran los diagramas de clases del análisis de cada caso de uso:

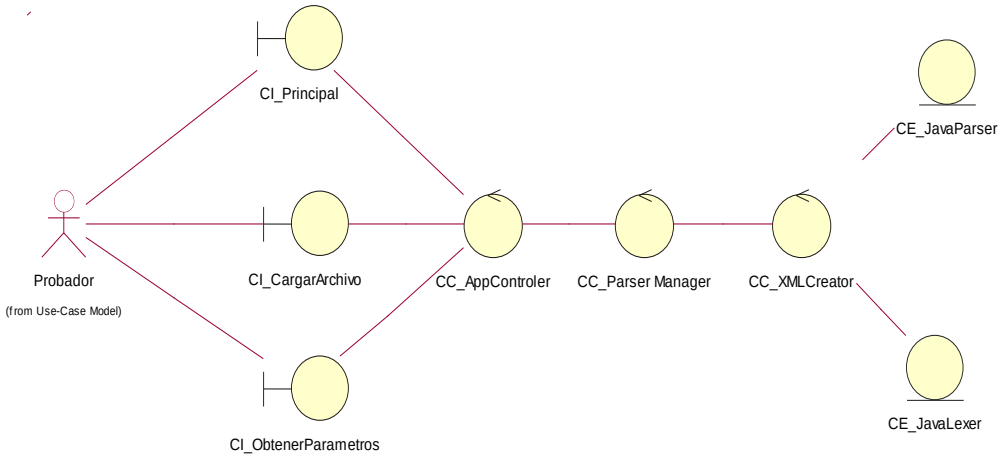

Figura 6: Diagrama de clases de análisis del caso de uso Formar Árbol Sintáctico.

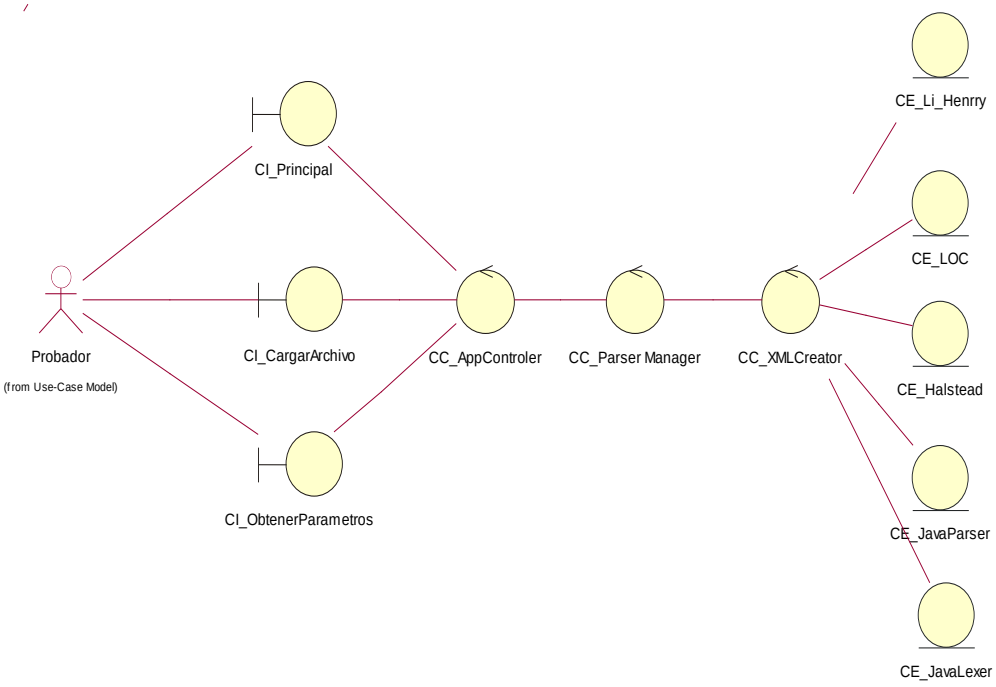

Figura 7: Diagrama de clase de análisis del caso de uso Determinar parámetros de código.

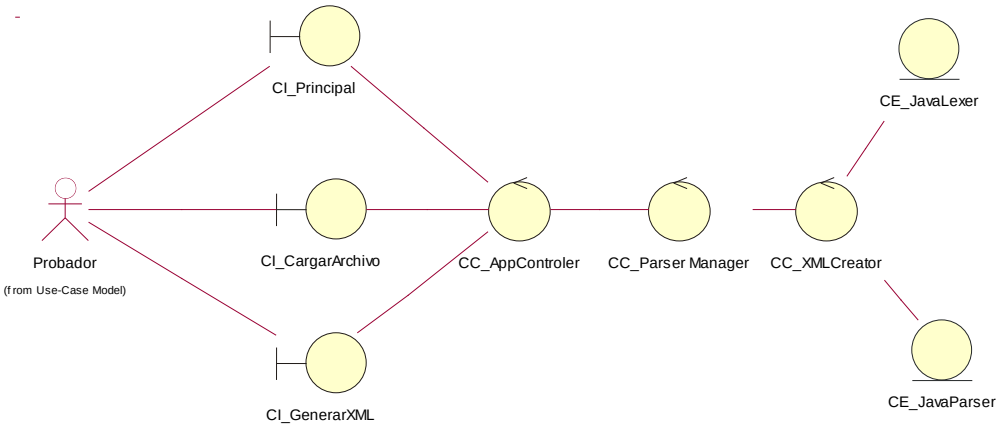

Figura 8: Diagrama de clase de análisis del caso de uso Exportar XML.

Seguidamente se visualizarán los diagramas de interacción (Colaboración y Secuencia) de cada caso de uso:

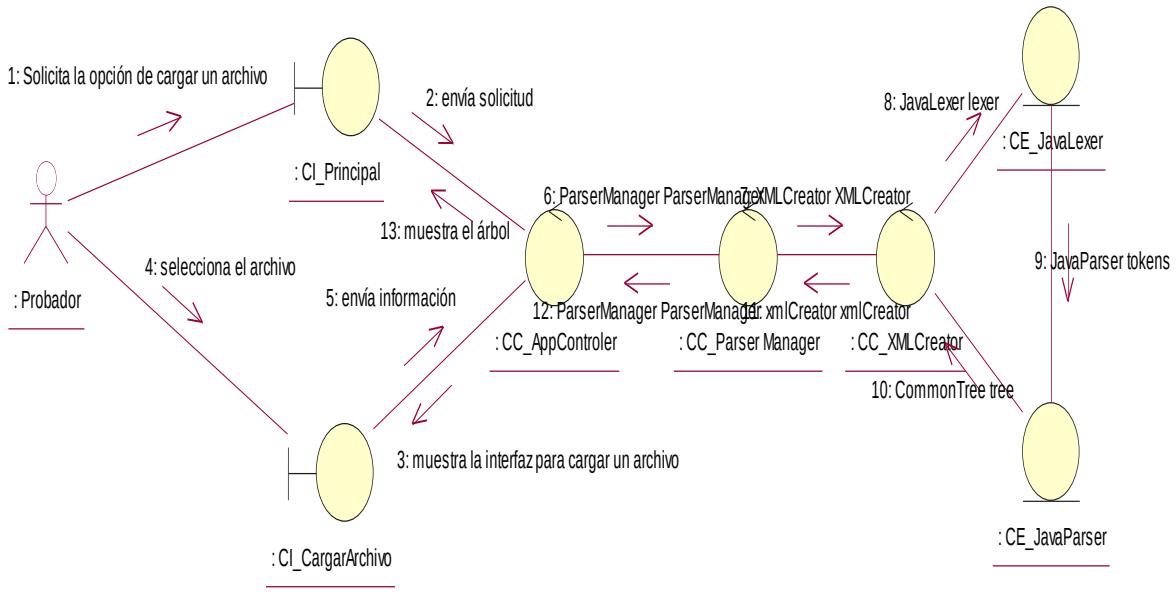

Figura 9: Diagrama de colaboración del caso de uso Formar Árbol Sintáctico.

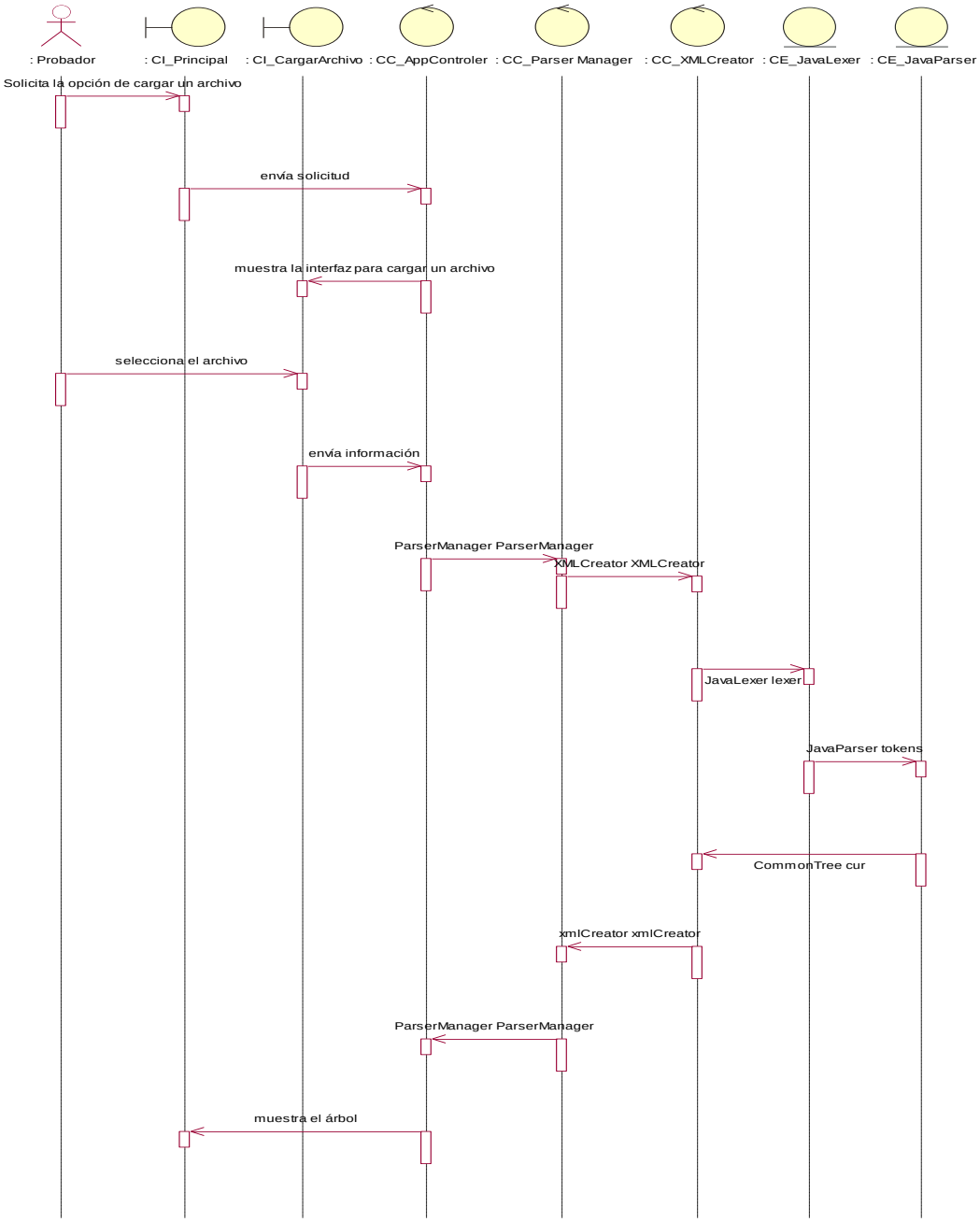

Figura 10: Diagrama de secuencia del caso de uso Formar Árbol Sintáctico.

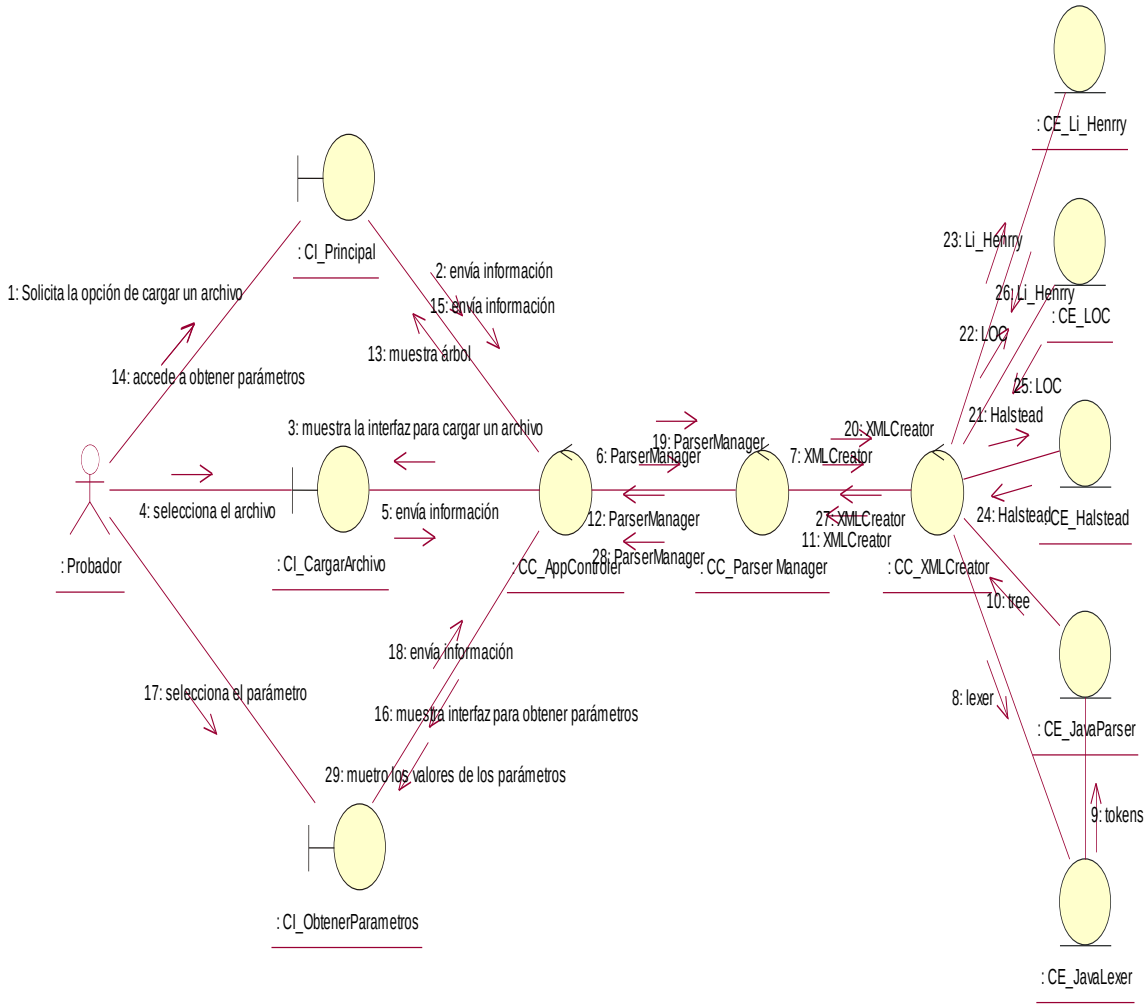

Figura 11: Diagrama de colaboración para el caso de uso Determinar parámetros de código.

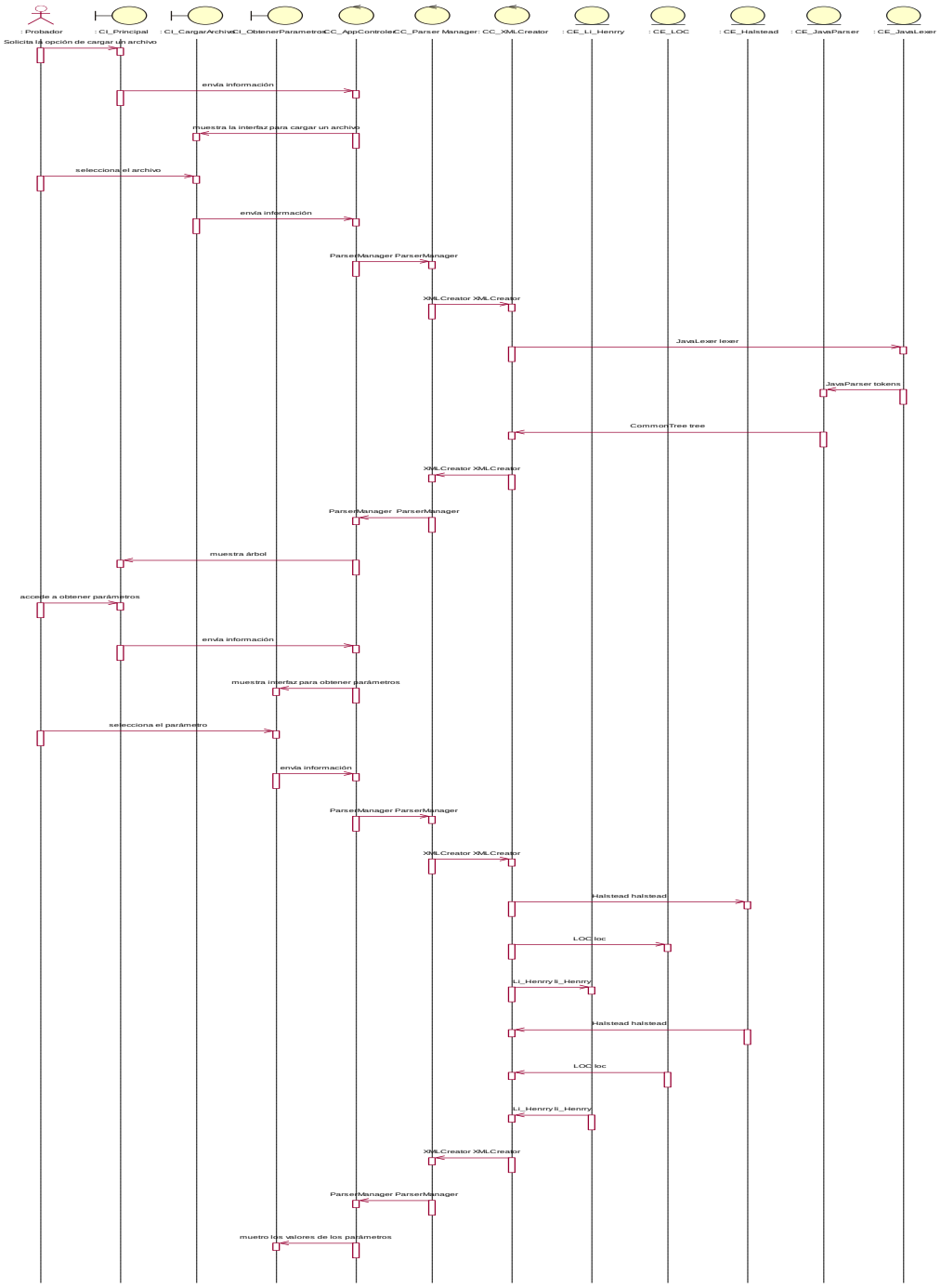

Figura 12: Diagrama de secuencia del caso de uso Determinar parámetros de código.

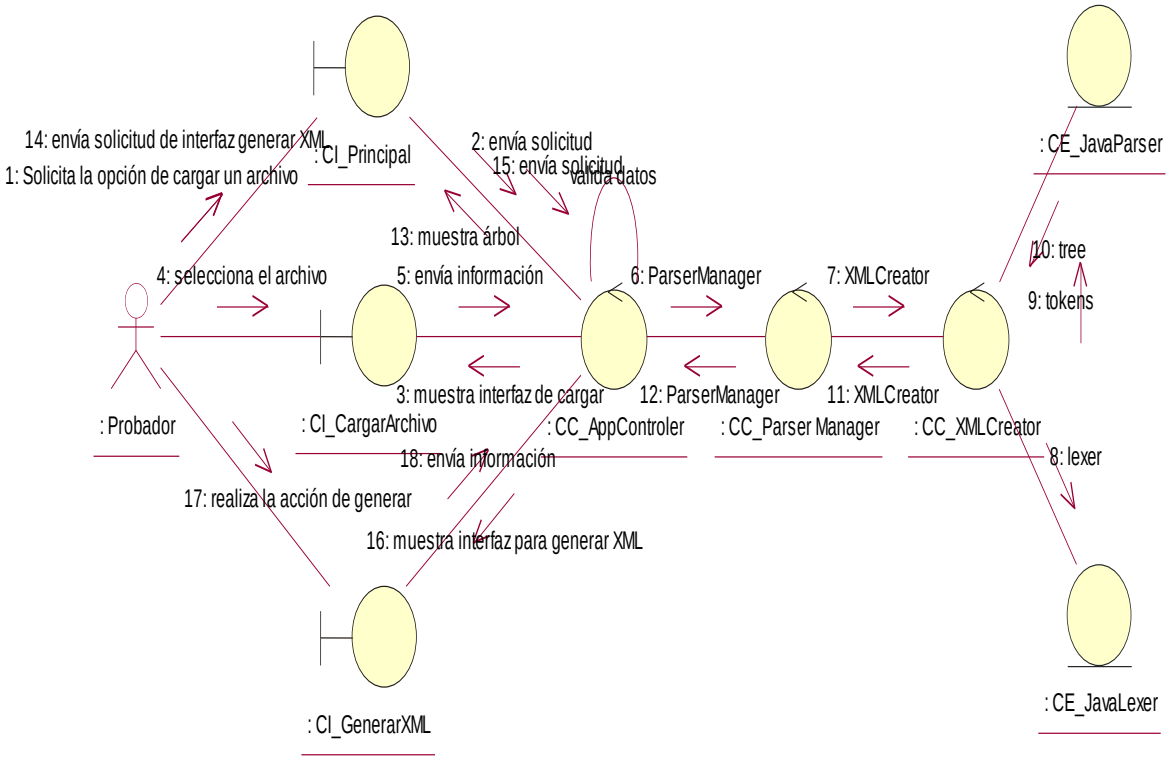

Figura 13: Diagrama de colaboración del caso de uso Exportar XML.

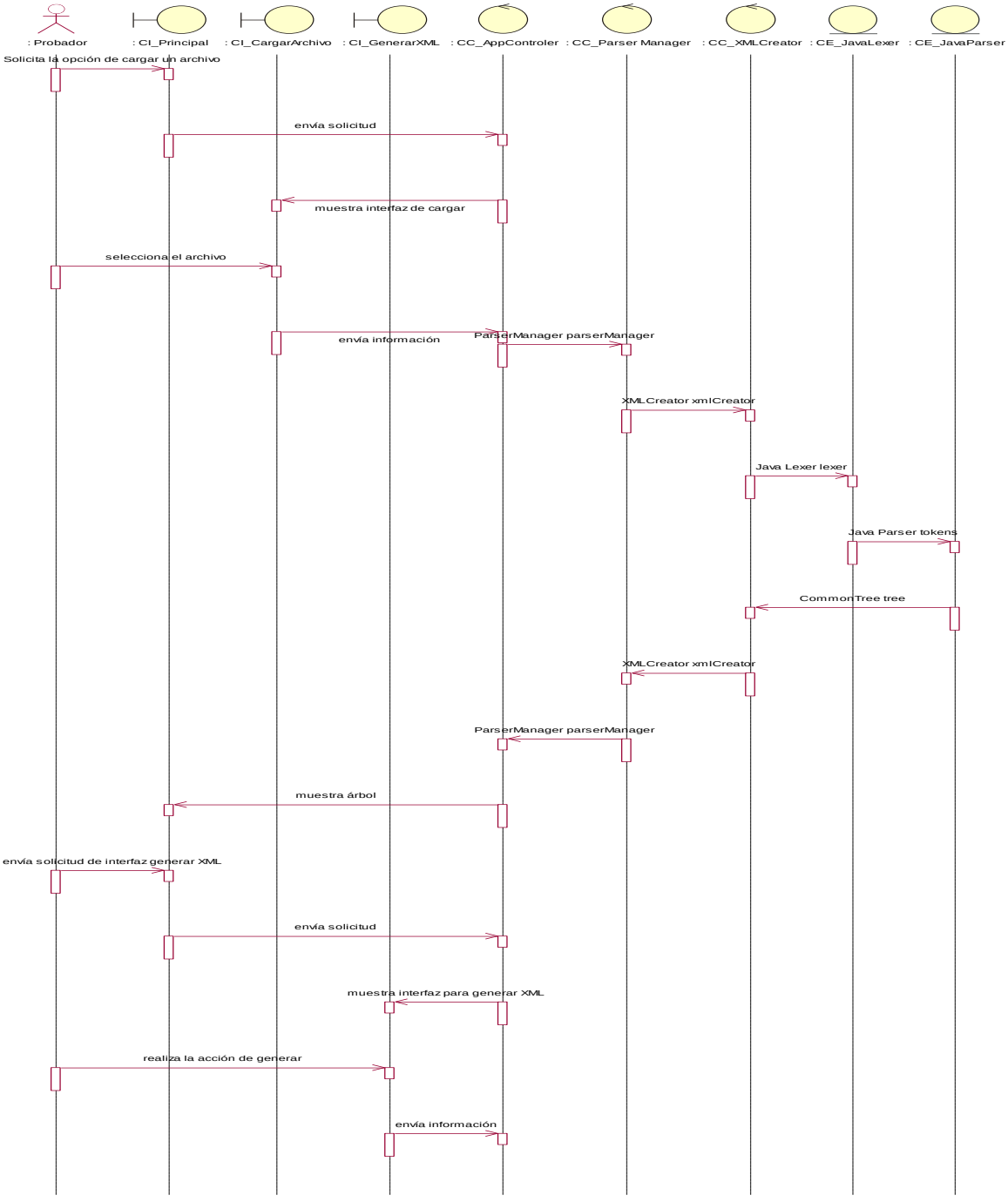

Figura 14: Diagrama de secuencia del caso de uso Exportar XML.

Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso. Se centra en cómo los requisitos funcionales tienen impacto en el sistema a desarrollar. Dentro de sus propósitos están: crear una entrada apropiada y un punto de partida para la implementación, descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo y adquirir una comprensión de los aspectos de las restricciones vinculadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia.

Tiene impacto en el sistema a desarrollar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es utilizado como entrada fundamental de las actividades de implementación.

Diagrama de clases de diseño

En el diagrama de clases de diseño se muestran los atributos y métodos de cada clase y se representa de una forma sencilla la colaboración y las responsabilidades de las distintas clases que forman el sistema.

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados: los diagramas de componentes y los diagramas de despliegue. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida. En el diseño se modeló el sistema y se encontró su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, o sea, el

modelo de análisis, que proporciona una comprensión detallada de los requisitos. Además, impone una estructura del sistema que se deberá conservar lo más fielmente posible, cuando se dé forma al sistema. (20)

Seguidamente se presenta el diagrama de clases de diseño de la aplicación llamada (JavaCodeChecker):

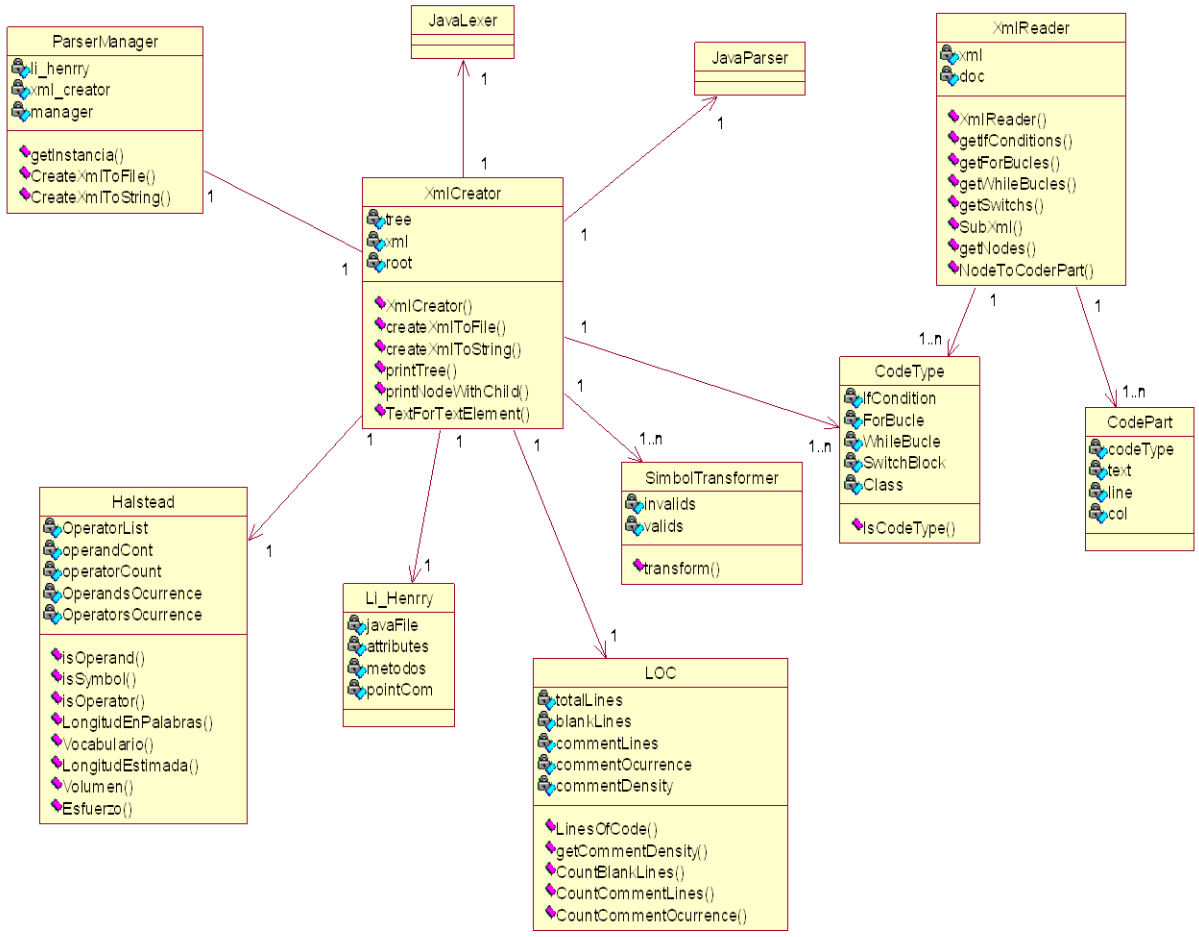

Figura 15: Diagrama de clase de diseño de la aplicación.

Descripción de las clases de la aplicación

La librería cuenta con las clases **JavaLexer** y **JavaParser** que son las encargadas de transformar el código Java en una estructura de árbol, JavaLexer divide en token el

código fuente a probar y JavaParser realiza la funcionalidad de conformar el árbol a partir de los token proporcionados por JavaLexer. Estas dos clases trabajan proporcionalmente, las mismas fueron generadas con la herramienta antlrworks usando una gramática de Java definida. Este árbol es transformado en un XML mediante la clase **XmlCreator** la que permite salvar el XML en un archivo o devolver el XML en forma de cadena mediante los métodos public String createXmlToString() y public void createXmlToFile(string dir).

En la creación del XML existe una **etiqueta** para la mayoría de los elementos del código, todas las etiquetas poseen la línea y la columna del elemento dentro del código, debido a que los símbolos como las "," los ";" y otros no pueden ser nombres de etiquetas porque XML no lo permite, existe entonces la clase **SymbolTransformer** la cual transforma estos símbolos en nombres para las **etiquetas** del XML. Ejemplo: al aparecer un ";" se crea la etiqueta **<pointC value=";" line="5" col="11" />.** La correspondencia entre símbolos y etiqueta es la siguiente:

No válidos = <.> <[> <]> <,> <;> <*> <{> <}> <(> <)>

Válidos = <point>, <openCorch>, <closedCorch>, <coma>, <pointC>, <aster>, <llaveOp>, <llaveClo>, <OpenPar>, <ClosedPar>

Existen también las etiquetas correspondientes a los operadores y los operandos, ejemplo:

<operator value="int" line="5" col="1" />
<operand value="a" line="5" col="5" />

Los atributos y métodos también tienen su representación en el xml, los mismos se encierran dentro de una etiqueta **atributo** o **método** según corresponda y dentro de esta etiqueta se desglosan los elementos que lo conforman.

Existe también el "enum" CodeType para referirse a los tipos de estructuras de códigos (if, while, etc.) evitando así el trabajo con cadenas.

Para el trabajo con este XML se provee la clase **XmlReader** la cual permite dado un XML acceder a las diferentes estructuras representativas del mismo, como las correspondientes a los ciclos “for” y “while” o las bifurcaciones (“if” o “switch”), esta clase devuelve objetos de tipo CodePart los cuales poseen la línea de código, el fragmento de código y la representación XML correspondiente a la estructura que se haya accedido lo que es posible a través de los métodos public String SubXml(String xpath) y public List<CodePart> getNodes(String codeType). Mediante XmlReader se puede acceder además a los valores de los parámetros mediante los métodos que la misma posee los cuales devuelven objetos de los mismos.

Existe además para el trabajo con la librería la clase **ParserManager** que actúa de fachada (Patrón Fachada) la cual evita la interacción con algunas clases de la librería directamente (**XmlCreator, JavaLexer y JavaParser**). Esta clase posee los métodos públicos “CreateXmlToFile(string parseFile, string saveFile)”, “CreateXmlToString(string parseFile”)y public ITree CreateTree(string parseFile) los cuales interactúan con las clases XmlCreator, JavaLexer y JavaParser para realizar sus respectivas operaciones quedando transparente para el usuario el uso de estas clases.

La librería realiza además el cálculo de parámetros correspondientes a las métricas de Li_Henrry, LOC y Halstead.

Se creó un paquete (carpeta) para las clases correspondientes a los parámetros.

-Li_Henrry:

La clase Li_Henrry contiene los atributos correspondientes para almacenar los valores de los parámetros de dicha métrica, los cuales van tomando sus valores a medida que se parsea el código. Los valores almacenados en esta clase son los correspondientes a: “Cantidad de atributos”, “Cantidad de métodos” y “Cantidad de punto y coma”. Interactúa con la clase XmlCreator. Ver (Capítulo1 sección10).

-LOC

Esta clase contiene los algoritmos y atributos necesarios para calcular y almacenar los valores correspondientes a las variantes de análisis de código respecto al tamaño del mismo como: “Cantidad de líneas comentadas”, “Cantidad de comentarios”, “Cantidad de líneas en blanco”, “Cantidad total de líneas” y la “Densidad de comentarios”. Interactúa con la clase XmlCreator. Ver (Capítulo1 sección10).

-Halstead

Esta clase contiene las variables para almacenar los valores correspondientes a la métrica. Como son: la cantidad de operadores, operandos, y con ellos una serie de parámetros como son el Volumen del código, el Esfuerzo, Longitud de las palabras, la Longitud estimada y total de operadores y operandos. Interactúa con la clase XmlCreator. Ver (Capítulo1 sección10).

Estándar de codificación utilizado

Cuando se hace necesario poner en práctica la programación es necesario hacer un código legible y entendible, no sólo para quien lo escribe, sino también para quien lo lee, de esta forma se contribuye a la comunicación de los desarrolladores. A continuación se muestra el estándar de codificación por el cual se regirá la implementación del sistema.

Los estándares, estilos o convenciones de codificación son importantes para los programadores debido a que ofrecen:

Extensibilidad: La facilidad con que se adapta el software a cambios de especificación. Un buen estilo de código fomenta programas que no sólo resuelven el problema, sino que también reflejan claramente la relación problema/solución. Esto tiene como efecto que muchos cambios simples en el problema reflejen de forma obvio los cambios a hacer en el programa.

Verificabilidad: la facilidad con que pueden comprobarse propiedades de un sistema. Si el estilo de código hace obvia la estructura del programa, eso ayuda a verificar que el comportamiento sea el esperado.

Reparabilidad: la posibilidad de corregir errores sin demasiado esfuerzo.

Capacidad de evolución: la capacidad de adaptarse a nuevas necesidades.

Comprensibilidad: la facilidad con que el programa puede ser comprendido.

Los estándares de codificación son reglas específicas a una lengua que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores, lo que evita la ocurrencia de errores. Durante la implementación estos estándares ayudan a la utilización o reutilización de códigos de los compañeros de trabajo del equipo de desarrollo, además otra de las ventajas que tiene es que ayuda a entender el código y a obtener un producto final organizado y con una muy buena calidad. Por lo visto anteriormente se utilizaron estándares de codificación en la solución, esto trae consigo la facilidad de que cuando se le quiera realizar mantenimiento al producto el código sea entendido con claridad.

Tipos de identificadores	Reglas para nombres	Ejemplos
Paquetes	El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel.	núcleo, parámetros.
Clases	Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas.	class JavaParser class JavaLexer class XmlCreator

	Estarán escritos en inglés. Intentar mantener los nombres de las clases simples y descriptivas.	
Métodos	Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula. Estarán escritos en inglés.	getSwitchs() getNodes()
Variables	Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres subguión, "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje. Los nombres de las variables deben ser cortos pero con significado. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales.	CommonTree tree; XmlDocument xml; XmlElement root;

Constantes	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_").	private int commentLines = 0;
		private int commentOcurrence = 0;
		private double commentDensity = 0;

En este capítulo se hizo un estudio de todo lo referente al análisis y diseño del sistema a desarrollar, se pudo observar el modelo de dominio, diagramas de clases del análisis, de interacción por cada caso de uso, evidenciando la arquitectura a seguir para conformar la aplicación JavaCodeChecker. Tuvo lugar también la descripción de cada una de las clases que conforman el sistema, mostrando el diagrama de clases del diseño correspondiente, además de las funcionalidades principales del reconocedor sintáctico, y la estructuración en sí de cada una de las clases expuestas.

Se definieron los casos de uso del sistema, describiendo cada uno de los mismos. Se tuvo en cuenta la presencia de un estándar de codificación a seguir para llevar a cabo la implementación del reconocedor sintáctico, lo que ayudó a tener un código más uniforme, claro y fácil de mantener, quedando ejemplificado el análisis y diseño del sistema; objetivo del presente trabajo.

Capitulo 3: Implementación y Prueba del Sistema

En el presente capítulo se realiza el flujo de trabajo Implementación definido por la metodología de desarrollo RUP. Se ejemplificará una descripción del uso de la aplicación. Así como los principales artefactos relacionados con este flujo de trabajo: los diagramas de despliegue y de componentes. Se especifican las pruebas realizadas al software con el objetivo de verificar que cumpla con los requerimientos y validar el correcto funcionamiento de la aplicación.

Implementación

La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo el trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos como los encontrados con distribuciones beta del sistema. Ya que el modelo de implementación denota la implementación actual del sistema en términos de componentes y subsistemas de implementación, es natural mantener el modelo de implementación a lo largo de todo el ciclo de vida del software.

La implementación es la fase más esperada en un proceso de desarrollo de un producto software, es donde se hacen realidad todas las ideas y artefactos que han sido modelados por el equipo de trabajo responsable de la solución; no constituye una etapa independiente y formalmente delimitada en el proceso.

Descripción del uso de la herramienta

En el Anexo 1 se muestra el XML con el cual se crearon los ejemplos que se mostrarán a continuación:

Para demostrar el uso de la herramienta se creó una aplicación donde dado un código Java, la misma crea un árbol de navegación a través del código donde se muestran detalladamente las estructuras del mismo. A continuación se muestra como queda el árbol de navegación en la aplicación:

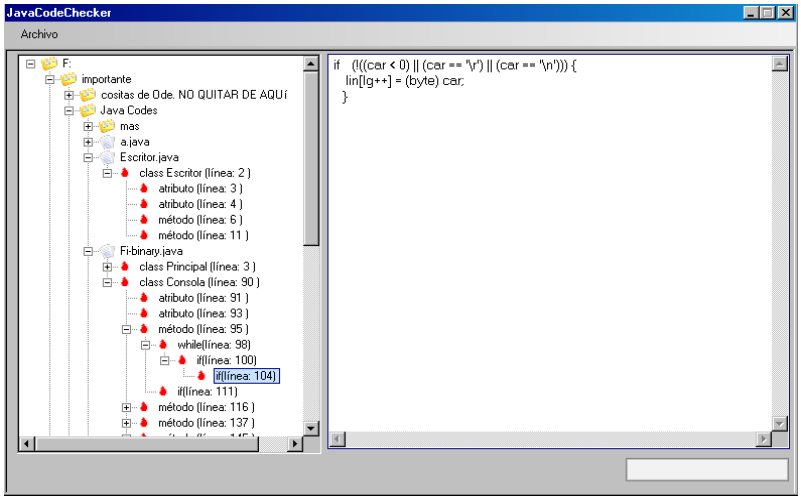

Figura 16: Funcionamiento de la herramienta.

Inicialmente se muestran las clases existentes en el archivo de código, luego las mismas pueden ser desglosadas en atributos y métodos. Al igual pasaría si el usuario escogería la posibilidad existente de entrada de un directorio; viajaría hasta el proyecto que querría analizar y tendría acceso a cada fichero Java perteneciente al mismo, accediendo igualmente a sus clases, luego a los atributos y métodos. Cada método puede ser desglosado en sus estructuras internas, entiéndase por estructura los tipos de código definidos en CodeType.

La aplicación además permite la posibilidad de reconstruir el código correspondiente a cualquier nodo del árbol, esto es posible con la clase XmlReader y sus métodos SubXml y getNodes pues el SubXml te devuelve el XML correspondiente a un nodo en específico y getNodes devuelve una lista de objetos CodePart los cuales además de la información correspondiente al nodo como la línea y columna contienen el código correspondiente al mismo.

La aplicación posibilita la visualización de un grupo de parámetros, estos son Li_Henrry, Líneas de código y Halstead, los mismos se le aplican a cada archivo analizado, dando clic derecho sobre ellos, como también posibilita la generación del XML correspondiente al código de cada uno. El XML posee también los valores de los parámetros. Siendo posible dicha operación teniendo las variantes de cargar de forma

única el archivo o a través de un directorio determinado que el usuario requiera realizar el análisis.

Para el conocimiento del usuario, la aplicación cuenta con una barra de procesos, brindando la posibilidad de visualizar al probador, el progreso de cualquiera de las funcionalidades existentes. Este componente está localizado en la parte derecha inferior de la aplicación. Se considera una característica importante dentro de la herramienta ya que permite mayormente el intercambio usuario-aplicación.

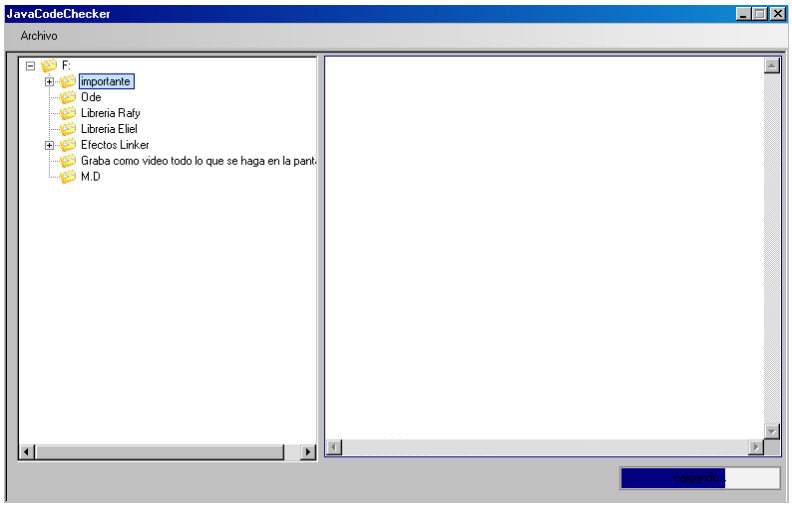

Figura 17: Pantalla de la aplicación donde se muestra la barra de procesos.

Modelo de Despliegue

El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El modelo de despliegue se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño.

Figura 17: PC cliente.

Recursos mínimos de hardware.

• 256 RAM.

• Ordenador Pentium.

• Teclado y Mouse.

• Procesador 600 MHz o superior.

• Espacio mínimo libre en disco duro de 15 MB.

Recursos mínimos de Software

• Sistema Operativo: Windows XP, Windows Vista, Windows 7.

Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean éstos componentes de código fuente, binarios o ejecutables. Normalmente contienen componentes, interfaces y relaciones entre ellos y como todos los diagramas, también puede contener paquetes utilizados para agrupar elementos del modelo.

Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. En cuanto a los componentes, sólo aparecen tipos de componentes, ya que las instancias específicas de cada tipo se encuentran en el diagrama de despliegue. Los diagramas de componentes muestran los componentes software que constituyen una parte reusable, sus interfaces, y sus interrelaciones, en muchos aspectos se puede considerar que un diagrama de componentes es un diagrama de clases a gran escala.

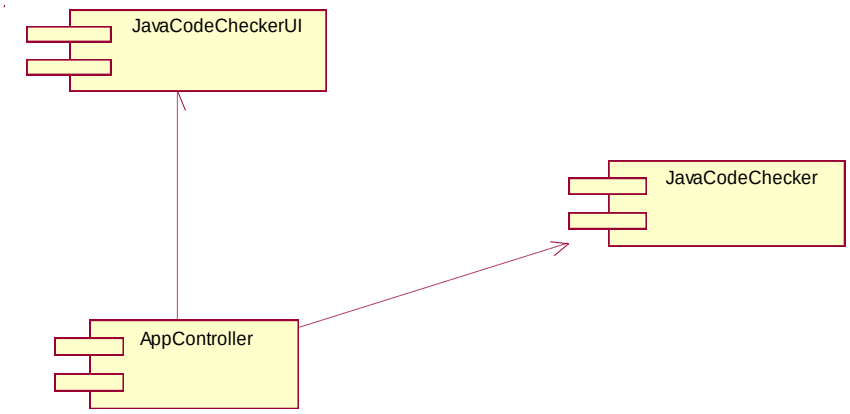

Figura 18: Diagrama de Componentes

Describiendo el flujo de relaciones entre los componentes visualizados en el diagrama; existe una dependencia entre AppController y la librería JavaCodeChecker, tal es así que AppController recibe el código fuente a procesar, luego se la envía a JavaCodeChecker, este componente que es la dll (librería) compone, conforma, crea el árbol AST, y a su vez le devuelve ese resultado hacia donde se originó el envío, luego AppController con la información, se la envía a JavaCodeCheckerUI que es la encargada de visualizar lo obtenido finalmente.

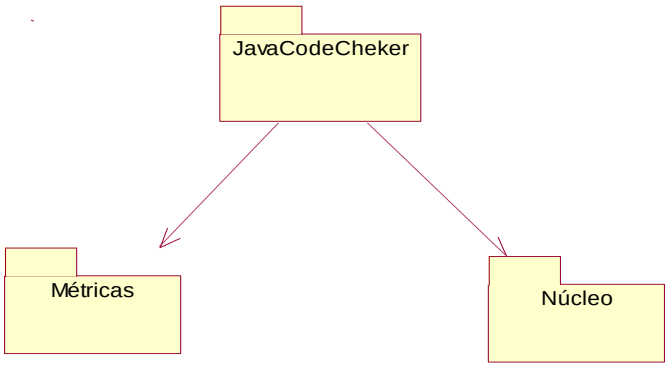

Figura 19: Organización por paquetes.

Se organizó por paquetes el proyecto en general, permitiendo una mayor claridad para la comprensión de las funcionalidades del reconocedor sintáctico creado.

Fase de prueba

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan el desempeño de un programa. Involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados, es por eso que la realización de las mismas a los software es un factor de vital importancia.

La fase de pruebas es la que añade al producto final el valor para afirmar que ya se ha alcanzado la calidad requerida. Gran porciento de los programas que se desarrollan tienen errores, y es en la fase de pruebas donde se descubren, ese es el valor que añade esta etapa. Es por ello que probar es una de las fases más importantes para que un producto salga con la calidad máxima y sin errores.

Objetivos

El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- ✓ Verificar la interacción de componentes.
- ✓ Verificar la integración adecuada de los componentes.
- ✓ Verificar que todos los requisitos se han implementado correctamente.
- ✓ Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- ✓ Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: probar la funcionalidad de los primeros prototipos, probar la estabilidad, cobertura y rendimiento de la arquitectura, probar el producto final, entre otras. Lo que conduce al principal beneficio de la prueba: proporcionar retroalimentación mientras hay todavía tiempo y recursos para hacer algo.

Pruebas de Caja Negra o Funcionales

También conocidas como Pruebas de Comportamiento o Pruebas Inducidas por los Datos, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El sistema se ve determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable, se selecciona un conjunto de ellas sobre las que se realizan las pruebas.

Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, hay que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en el sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos. Entonces, dado que la prueba exhaustiva es imposible, el objetivo final es pues, encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible.

El objetivo de realizar este tipo de prueba al sistema es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación.

El proceso de pruebas de Caja Negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

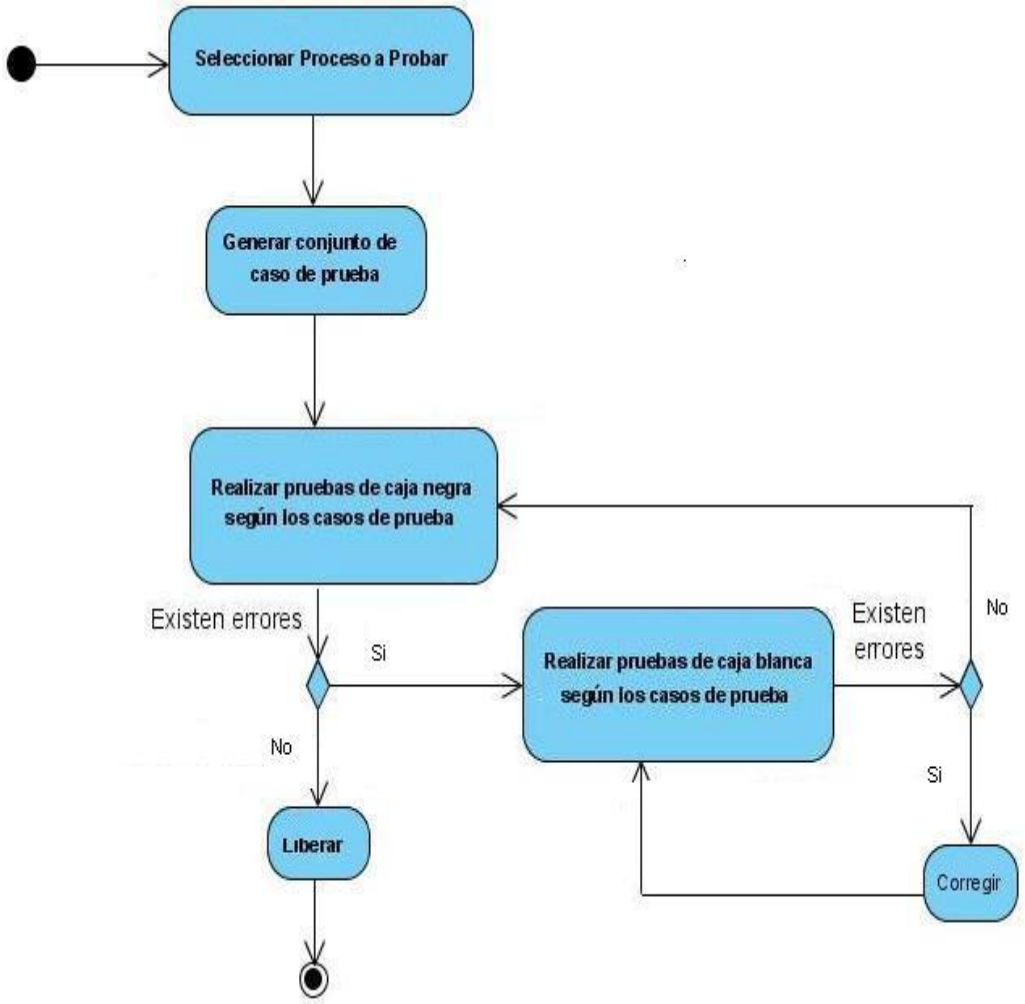

Figura 20: Ciclo de prueba de Caja Negra

Las pautas para desarrollar casos de prueba con esta técnica son:

- Si una condición de entrada especifica un rango de valores, se diseñarán casos de prueba para los dos límites del rango, y otros dos casos para situaciones justo por debajo y por encima de los extremos.
- Si una condición de entrada especifica un número de valores, se diseñan dos casos de prueba para los valores mínimo y máximo, además de otros dos casos de prueba para valores justo por encima del máximo y justo por debajo del mínimo.
- Aplicar las reglas anteriores a los datos de salida.

- Si la entrada o salida de un programa es un conjunto ordenado, habrá que prestar atención a los elementos primero y último del conjunto.

Diseño de casos de prueba

A continuación se detallan los pasos seguidos para la realización del diseño de casos de pruebas para las secciones correspondientes a los Casos de Usos definidos para el sistema.

Paso 1: Definir las secciones y escenarios de los Casos de Usos.

Nombre de la sección	Escenarios de prueba de la sección	Descripción de la funcionalidad	Flujo central
SC1: Formar Árbol Sintáctico.	EC1: Formar Árbol Sintáctico.	El sistema genera un árbol sintáctico a partir del código analizado.	El probador, carga un archivo Java de forma única o a través de un directorio de carpetas, luego el sistema analiza el fichero, extrayendo la secuencia de token correspondiente, y a la vez forma el árbol sintáctico clasificando cada construcción de código según su tipo y composición. Finalmente se visualiza el árbol construido.
SC2: Calcular parámetros de código.	EC2: Determinar parámetros de código.	Se obtienen los parámetros de un archivo Java introducido previamente por el probador.	Una vez que el probador cargue un archivo Java ya sea de forma única o mediante un directorio, y que esté visualizado el árbol sintáctico, el probador da clic derecho sobre el fichero que desee conocer los parámetros de código; ya sea Li_Henrry, Líneas de código (LOC) o Halstead, el sistema visualiza el resultado.
SC3: Exportar	EC: 3 Exportar XML.	Se exporta el archivo XML donde	Una vez que se haya visualizado el árbol sintáctico,

XML.		se guarda la estructura del árbol generado previamente.	el probador da clic derecho sobre este y accede a la opción exportar el XML. El sistema te permite además salvarlo en cualquier parte de la computadora y muestra finalmente un mensaje indicando que la operación tuvo éxito.

Paso 2: Realizar el diseño de casos de pruebas.

Id Escenario de prueba	Escenario	Respuesta del sistema	Resultado de la prueba
EC1	Formar Árbol Sintáctico.	El sistema genera un árbol sintáctico a partir del código analizado.	Satisfactoria.
EC2	Determinar parámetros de código.	Se obtienen los parámetros de un archivo Java introducido previamente por el probador.	Satisfactoria.
EC3	Exportar XML.	Se exporta el archivo XML donde se guarda la estructura del árbol generado previamente.	Satisfactoria.

Durante la fase de prueba no se detectó ninguna no conformidad evidenciándose que el sistema construido está preparado para ser usado como apoyo del proceso de pruebas de liberaciones llevadas a cabo por el Grupo de Calidad del CESIM.

En este capítulo se obtuvo el diagrama de componentes que representan cada parte modular del sistema y las relaciones entre ellas. La distribución física del sistema pudo ser visualizada en el diagrama de despliegue, como también los requisitos no funcionales de hardware en que debe ser ejecutada la herramienta, se mostró el diagrama de paquetes del sistema destacando la organización requerida para el logro de un mejor trabajo.

Además se dio una explicación del uso de la herramienta JavaCodeChecker, dividiendo cada una de las principales funciones del reconocedor sintáctico. Por último la fase de prueba dio a conocer el cumplimiento de cada funcionalidad del software construido, no observándose no conformidades y preparado para ser usado por el Grupo de Calidad del CESIM.

Conclusiones generales

- Se indagó a nivel internacional, nacional y en la Universidad de las Ciencias Informáticas, las tecnologías similares existentes en la actualidad, evidenciándose la necesidad de la construcción del sistema creado.

- Se definieron los parámetros de código relacionados con el tamaño del mismo, y se logró la implementación de las funcionalidades de cada uno de ellos.

- Se usó un generador de reconocedores sintácticos para la construcción del sistema. Utilizando para la generación del reconocedor una gramática de Java existente, y de esta forma lograr mayor calidad en el resultado final.

- Luego del análisis de código, la salida que se obtiene como resultado es en un XML, que representaría el código fuente analizado en forma de árbol, para que posteriormente dicho XML pueda ser utilizado por los probadores del Grupo de Calidad del CESIM en aras de lograr la calidad del software.

- Quedó ejemplificado las principales metodologías de desarrollo de software, lenguajes, entornos de desarrollo integrado, sistema de control de versiones y tecnologías o frameworks de persistencia existentes, haciendo una caracterización profunda, comparando y seleccionando en cada caso la más adecuada para utilizarla en la obtención de un producto con los costes de tiempo y calidad requeridos.

- Se implementaron todas las funcionalidades previstas, viendo en cada una un resultado satisfactorio. Y por último, se validó la implementación de la aplicación mediante pruebas de Caja Negra, quedando listo el sistema para ser utilizado en el CESIM.

Recomendaciones

Ya vista la importancia de la construcción del sistema desarrollado, se recomienda llevar a cabo otra investigación similar con el mismo fin; crear un sistema para realizar el reconocimiento sintáctico. Con la diferencia de que analice otro tipo de lenguaje de

programación, por ejemplo el C# ó PHP, ya que conjuntamente con Java son los más utilizados por el equipo de desarrolladores de software pertenecientes al CESIM.

Reflejar en el árbol de sintaxis abstracta los detalles del formato de código; tipo de letra, estilo de letra, entre otros.

Realizar las implementaciones que permitan extraer otros parámetros de código.

Referencias bibliográficas

1. **Giraldo, Juan Pablo.** Monografias. [En línea] http://www.monografias.com/trabajos15/ingenieria-software/ingenieria-software2.shtml.

2. [En línea] 2010. http://www.uco.es/users/ma1fegan/pl/practicas/ANTLR/Introduccion-al-analisis-sintactico-con-ANTLR.pdf.

3. [En línea] http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico.

4. Programación Extrema. [En línea] 2010. http://extre.blogspot.com/2006/11/objetivos-de-la-programacin-extrema.html.

5. IDEM referencia 2

6. [En línea] http://people.cs.uchicago.edu/~borja/pubs/revistaeside2002.pdf.

7. *Ecma International. Standard ECMA-334 C# Language Specification. s.l: Ecma International.* 2010.

8. [En línea] http://msdn.microsoft.com/en-gb/vstudio/bb265237.aspx.

9. [En línea] 2010. http://www.onuva.com/files/OnuvaConfl/OnuvaConflDesarrollo.pdf.

10. Microsoft Corporation . MSDN Microsoft Developer Network . Visual Studio Developer Center. *Microsoft Corporation.* [En línea]

11. **Zerpa, José, Escobar, Javier y Mendoza, Yonel.** Desarrollo de Aplicaciones Basadas en Ambientes Virtualizados Bajo Software Libre y Estándares Abiertos, a Través de la Metodología de Producción Acelerada SCRUM. [En línea]

12. Control de Versiones SVN. [En línea] Student portfolios and software quality metrics in computer science education. Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771.

13. Prueba de Programas. [En línea] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm

14. A Metric Suite for Object Oriented Design. Chidamber, Shyam R. and Kemerer, Chris F. 6, s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. [En línea] 1994.

15. [En línea] http://urriellu.net/es/articles-software/csharp-advantages.html.

16. Ver referencia 12.

17. *Mitkov R, the Oxford Handbook of Computational Linguistics. Oxford University Press.* New York : s.n., 2003.

18. *Ingeniería de Software 2. Conferencia #1. Continuación del FT Análisis y Diseño. Modelo de Diseño. .* Curso 2008-2009.

19. *Rational Enterprise Edition, Ayuda extendida.*

20. *Student portfolios and software quality metrics in computer science education. Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771.*

Bibliografía

A Metric Suite for Object Oriented Design. Chidamber, Shyam R. and Kemerer, Chris F. 6, s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. [En línea] 1994.

Control de Versiones SVN. [En línea] Student portfolios and software quality metrics in computer science education. Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771.

Ecma International. Standard ECMA-334 C# Language Specification. s.l: Ecma International. 2010.

[En línea] 2010. http://www.uco.es/users/ma1fegan/pl/practicas/ANTLR/Introduccion-al-analisis-sintactico-con-ANTLR.pdf.

[En línea] http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico.

[En línea] http://people.cs.uchicago.edu/~borja/pubs/revistaeside2002.pdf.

[En línea] 2010. http://www.onuva.com/files/OnuvaConfl/OnuvaConflDesarrollo.pdf.

[En línea] http://msdn.microsoft.com/en-gb/vstudio/bb265237.aspx.

[En línea] http://urriellu.net/es/articles-software/csharp-advantages.html.

Free Download Software. (2004-2007). Recuperado el 10 de enero de 2010, de http://www.antlr.org/download.html

Giraldo, Juan Pablo. Monografias. [En línea] http://www.monografias.com/trabajos15/ingenieria-software/ingenieria-software2.shtml.

[En línea] http://translate.google.com.cu/translate?hl=es&sl=en&u=http://sourceforge.net/projects/antlr-mode/&ei=DPITTliMNcL48AaCicWeCg&sa=X&oi=translate&ct=result&resnum=9&ved=0CEoQ7gEwCA&prev=/search%3Fq%3Dantlr%2Bdownload%26hl%3Des

[En línea] http://www.masadelante.com/faqs/sistema-operativo

Ingeniería de Software 2. Conferencia #1. Continuación del FT Análisis y Diseño. Modelo de Diseño. . Curso 2008-2009.

JProbe. [En Línea] http://www.quest.com/jprobe/

JProbe. [En Línea] http://www.componentsource.com/products/jprobe-suite/index.html

JTest. [En Línea] http://www.als-es.com/home.php?location=herramientas%2Fentorno-desarrollo%2Fjtest

MyGnet. (Agosto de 2005). Recuperado el 25 de diciembre de 2009, de http://www.mygnet.net/codigos/java/compiladoreseinterpretes/analizador_sintactico_xml_v2.2423

Mitkov R, the Oxford Handbook of Computational Linguistics. Oxford University Press. New York : s.n., 2003.

Microsoft Corporation . MSDN Microsoft Developer Network . Visual Studio Developer Center. *Microsoft Corporation.* [En línea]

Monografías. [En Línea] http://www.monografias.com/trabajos11/compil/compil.shtml.

pedels, desde las amígdalas. (2009). Recuperado el 21 de enero de 2010, de http://pedels.net/2008/01/16/informatica/jsqlparser-un-parser-analizador-sintactico-para-sql-en-java-continuacion/

PRESSMAN, **Roger** "Ingeniería del Software. Un enfoque práctico". McGraw-Hill/Interamericana de España.2002.

Programación Extrema. [En línea] 2010. http://extre.blogspot.com/2006/11/objetivos-de-la-programacin-extrema.html.

Prueba de Programas. [En línea] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm

Rational Enterprise Edition, Ayuda extendida.

RSM [En Línea] http://www.fileheaven.com/descargar/resource-standard-metrics-for-c-c-and-java/33666.htm

Student portfolios and software quality metrics in computer science education. Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771

Velásquez, **Juan R (Prof.); Martínez Igor (Ing.); Batista, Hilda Elizabeth (Ing.); Reynoso, Daina Ivette (Lic.)**. Métricas en Ingeniería de Software. MO-TIC. Gestión de Proyectos.

Villena Román, Julio. Laboratorio de Programación. Pruebas de Programas.

Zerpa, José, Escobar, Javier y Mendoza, Yonel. Desarrollo de Aplicaciones Basadas en Ambientes Virtualizados Bajo Software Libre y Estándares Abiertos, a Través de la Metodología de Producción Acelerada SCRUM. [En línea]