UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

INTÉRPRETE PARA LA VALIDACIÓN DE LAS VARIABLES DE LOS ENSAYOS CLÍNICOS EN EL SISTEMA ALASCLÍNICAS

TESIS EN OPCIÓN AL TÍTULO DE MÁSTER EN INFORMÁTICA APLICADA

Autora: Ing. Lucía Rodríguez García

Tutor: MSc. Ing. Iván Pérez Mallea

La Habana, 2011

DECLARACIÓN JURADA DE AUTORÍA Y AGRADECIMIENTOS

Yo Lucía Rodríguez García, con carné de identidad 83082517458, declaro que soy la autora principal del resultado que expongo en la presente tesis titulada “Intérprete para la validación de las variables de los Ensayos Clínicos en el sistema alasClínicas”, para optar por el título de Máster en Informática Aplicada.

Este trabajo fue desarrollado durante el período 2010 – 2011 en colaboración con mis compañeros de equipo: Ing. Erislán Martínez Jera y Msc. Martha Denia Hernández Ramírez; quienes me reconocen la autoría principal del resultado expuesto en esta investigación.

A todos mis amigos les estoy muy agradecida. En especial deseo agradecer a mi tutor por ayudarme en mi formación como máster y a mi compañero Erislán Martínez Jera por su ayuda en el desarrollo del trabajo. Además, deseo agradecer a todas las personas que de una forma u otra contribuyeron a mi crecimiento profesional y humano en general, colegas, amigos y familiares, les doy las más sinceras gracias.

Finalmente declaro que todo lo anteriormente expuesto se ajusta a la verdad, y asumo la responsabilidad moral y jurídica que se derive de este juramento profesional. Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los __ días del mes de _____ del año ____.

_____

Firma

RESUMEN

Los Ensayos Clínicos son estudios de investigación clínica que evalúan la eficacia y seguridad de nuevos fármacos. En Cuba, el Centro de Inmunología Molecular desarrolla estos estudios específicamente en personas con enfermedades como el cáncer, que atentan contra el sistema inmunológico. En cada ensayo se recoge un conjunto de datos de cada sujeto incluido en el estudio, los cuales constituyen la base de las investigaciones. De aquí que la validez de los datos se convierte en un requisito indispensable para lograr que los resultados de las investigaciones sean lo más certeros posible. En el mundo se han desarrollado varios sistemas de gestión de ensayos clínicos, más conocidos como CTMS por sus siglas en inglés, los cuales permiten establecer una validación muy sencilla de los datos.

La Universidad de las Ciencias Informáticas en colaboración con el Centro de Inmunología Molecular, desarrolla el sistema alasClínicas, el cual es un CTMS que permite la creación de ensayos clínicos, la inserción de sujetos en los estudios y la gestión de los datos de los sujetos. Para lograr una mayor validez de estos datos, se determinó que el gerente del estudio, encargado del diseño del mismo, debe establecer reglas de validación en el sistema que definan los rangos de valores que pueden tomar los datos. Estas reglas una vez establecidas deben ser ejecutadas por el sistema. Teniendo en cuenta que serán definidas en un lenguaje de alto nivel, cercano al lenguaje natural, la presente investigación propuso el desarrollo de un intérprete que en sus funciones de traductor, realice el análisis y ejecución de las reglas para cada uno de los datos. El intérprete fue implementado con la tecnología J2EE, la misma que soporta al sistema alasClínicas. En el desarrollo del mismo se definió el lenguaje que permite la declaración de las reglas, se implementaron: los autómatas para el reconocimiento de los tokens, las reglas de la gramática y las clases para el desarrollo del árbol de sintaxis abstracta. Se utilizó el patrón Visitor para recorrer el árbol, y la Notación Polaca para la generación del código intermedio. Se hizo uso de la programación orientada a objetos que facilita la reutilización del código y se tuvo en cuenta la detección de errores en cada una de las fases del intérprete. La herramienta será utilizada primeramente en la declaración de las reglas para verificar que las mismas sean definidas correctamente, y posteriormente en la ejecución de las reglas, analizando el cumplimiento de estas en los datos. Finalmente se evaluó el funcionamiento del intérprete por un grupo de expertos, concluyendo como resultado, que la herramienta contribuye a la validación de datos de ensayos clínicos en el sistema alasClínicas y por consiguiente, a la obtención de estudios clínicos más certeros en el Centro de Inmunología Molecular.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En el mundo de la Informática, los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que usando lenguajes de bajo nivel como “lenguaje máquina” o “ensamblador”. Un código escrito en lenguaje de alto nivel es independiente de la máquina. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras. Son lenguajes en los que las instrucciones enviadas para que el ordenador ejecute ciertas órdenes, son similares al lenguaje humano. [4] Dado que el ordenador no es capaz de reconocer estas órdenes surgen los traductores. Un traductor en informática, es un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente (o lenguaje de alto nivel) hasta un texto o programa escrito en un lenguaje destino produciendo, si cabe, mensajes de error. Los traductores engloban tanto al compilador como al intérprete.

Hoy día el uso de compiladores o intérpretes puede extenderse a cualquier sistema o aplicación que necesite la traducción de un código o programa, de un lenguaje a otro.

La Universidad de las Ciencias Informáticas (UCI) en el campo de la informática médica, desarrolla el proyecto Ensayos Clínicos en coordinación con el Centro de Inmunología Molecular (CIM). Este proyecto desarrolla el software alasClínicas con el objetivo de viabilizar la gestión de ensayos clínicos (EC) en dicho centro. Este sistema se encarga entre otras cosas, de informatizar la recogida de los datos de los ensayos almacenándolos en Cuadernos de Recogida de Datos (CRD) electrónicos. El Gerente de Datos es el encargado de la confección de los cuadernos en el sistema donde establece el conjunto de variables que serán recogidas en los mismos.

Una de las principales metas que persigue el desarrollo del sistema alasClínicas, es lograr que los datos que se recogen en las variables de los CRD, contengan la menor cantidad de errores posible. En aras de alcanzar este fin, y con el objetivo de concretar un modelo que defina cómo se recopila la información en los CRD, se realizó un estudio de las variables que generalmente son recogidas en varios de los ensayos que se han realizado en el CIM. A partir de este estudio fueron definidos los tipos de datos que se recogen en las variables de los cuadernos, las posibles dependencias entre las variables, y por consiguiente, se arribó a la siguiente conclusión: para lograr una mayor efectividad en la entrada de los datos, el Gerente de Datos previamente deberá establecer “reglas de validación” en el sistema, que definan los rangos de valores que pueden tomar cada una de las variables, teniendo en cuenta el tipo y la dependencia

entre ellas. Una vez definidas estas reglas, a la hora de entrar los datos en los cuadernos, el sistema se valdrá de las mismas para detectar la existencia de errores.

Teniendo en cuenta lo explicado anteriormente se plantea el **problema científico** de la investigación: ¿Cómo lograr que el sistema alasClínicas, ejecute las reglas de validación definidas por el Gerente de Datos para las variables que se recogen en los CRD, de manera que contribuya a identificar errores en la entrada de los datos en el sistema?

Dentro de los traductores de lenguajes de alto nivel, específicamente el intérprete es un programa que analiza y ejecuta simultáneamente, sentencias escritas en un lenguaje fuente. Cualquier intérprete tiene dos entradas: un programa escrito en un lenguaje fuente y los datos. A partir de dichas entradas, mediante un proceso de interpretación, el intérprete va produciendo resultados. [4]

Teniendo en cuenta que las reglas de validación definidas por el Gerente de Datos serán escritas en un lenguaje de alto nivel, cómodo y cercano al lenguaje natural, el uso de un intérprete para la interpretación y ejecución de las reglas por el sistema alasClínicas podría ser de gran utilidad, por lo que se determinó como **objeto de estudio** de la investigación**:** los intérpretes como traductores de lenguajes de alto nivel, enmarcando el **campo de acción** en la validación de datos de ensayos clínicos a través de intérpretes.

Además se definió como **objetivo general** de la investigación: desarrollar un intérprete para la validación de los datos de los ensayos clínicos que se recogen en los cuadernos de recogida de datos, en el sistema alasClínicas**.**

Hipótesis: El desarrollo de un intérprete para la validación de los datos de los ensayos clínicos, contribuirá a identificar errores en la entrada de los datos a los CRD en el sistema alasClínicas.

Para dar cumplimiento al objetivo planteado se definieron las siguientes **tareas**:

- Análisis de las variables que se recogen en los ensayos clínicos así como de la importancia de la validación de las mismas.
- Análisis de la gestión de datos en los Sistemas de Gestión de Ensayos Clínicos, principalmente en el sistema alasClínicas, y de las reglas de validación que se definirán en este sistema.
- Análisis de los intérpretes como traductores de lenguajes de alto nivel.
- Determinación de un método para evaluar el intérprete.
- Definición de un lenguaje de alto nivel para el establecimiento de las reglas de

validación.

- Definición de las reglas de producción para el intérprete.
- Implementación de las clases del intérprete, incluyendo los analizadores léxico, sintáctico y semántico.
- Evaluación del intérprete.

Para el desarrollo de la investigación se utilizaron los siguientes **métodos y técnicas**:

- Métodos teóricos:

Método Analítico – Sintético: durante el proceso investigativo, se realizaron búsquedas bibliográficas donde se analiza y sintetiza la información relacionada con el tema.

Método Histórico – Lógico: en la investigación se realiza un estudio de la evolución de los intérpretes así como de las tendencias actuales en el uso de los mismos.

- Métodos empíricos:

Método Delphi: consiste en la utilización sistemática del juicio intuitivo de un grupo de expertos para obtener un consenso de opiniones informadas. Se utiliza durante la investigación para validar la propuesta de solución.

- Técnicas de recopilación de información:

Entrevista: fue utilizada para recopilar información a través de expertos en la gestión de datos de ensayos clínicos.

Novedad y aporte teórico:

- Estructura o lenguaje para la definición de reglas de validación en el sistema alasClínicas.

Novedad y aporte práctico:

- Implementación de un intérprete para la validación de los datos de los ensayos clínicos.
- Identificación de errores en la entrada de los datos de los EC.
- Disminución de errores en los datos de los ensayos clínicos.

Estructura capitular del documento:

El documento está estructurado en: introducción, tres capítulos y conclusiones. A continuación se presenta lo que aborda cada capítulo.

- En el primero se realiza un análisis de las variables de los ensayos clínicos y de la importancia de la validez de las mismas. Además, se realiza un estudio de la gestión de datos en los Sistemas de Gestión de Ensayos Clínicos, específicamente en el sistema alasClínicas, y de las reglas de validación que

se propone definir en este sistema. En este mismo capítulo, se realiza un estudio de los intérpretes como traductores de lenguajes de alto nivel. Finalmente se determina el uso de un método para evaluar el intérprete que propone la investigación.

- En el segundo capítulo, se presenta el desarrollo del intérprete en cada una de sus fases, así como sus características, estructura y las clases que lo componen.
- En el tercer capítulo se enuncian las pruebas realizadas al intérprete y se evalúa el mismo con el método seleccionado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

El siguiente capítulo presenta un estudio de las variables de los EC, de la gestión de datos en los Sistemas de Gestión de Ensayos Clínicos, así como del sistema alasClínicas y sus funcionalidades. Además aborda temas relacionados con los Intérpretes como traductores de lenguajes de alto nivel, su estructura y características, así como herramientas para la generación de fases de un intérprete.

1.6 Gestión de ensayos clínicos en el CIM

El objetivo principal de las investigaciones en el Centro de Inmunología Molecular es la búsqueda de nuevos productos para el diagnóstico y tratamiento del cáncer y enfermedades relacionadas con el sistema inmunológico. Los proyectos de investigación básica están concentrados en la inmunoterapia del cáncer, especialmente en el desarrollo de vacunas moleculares, ingeniería de anticuerpos, ingeniería celular, bioinformática y regulación de la respuesta inmune.

El CIM realiza, en hospitales altamente especializados, ensayos clínicos para el tratamiento de cáncer de diferentes orígenes y otras enfermedades del sistema inmune. [10]

Los ensayos clínicos son estudios de investigación que prueban el funcionamiento de los nuevos enfoques clínicos en las personas. Cada estudio responde preguntas científicas e intenta encontrar mejores formas de prevenir, explorar, diagnosticar o tratar una enfermedad.

Cada ensayo clínico tiene un protocolo o plan de acción para llevarlo a cabo. El plan describe lo que se hará en el estudio, cómo se hará y por qué cada parte del estudio es necesaria. [26]

1.2. Las variables de los ensayos clínicos

Cada EC tiene asociado un Cuaderno de Recogida de Datos (CRD). Los Cuadernos de Recogida de Datos (más conocidos como Case Report Forms: CRF, por sus siglas en inglés) son documentos impresos o electrónicos usados en los ensayos clínicos para coleccionar y almacenar la información definida por el protocolo de un estudio clínico. [22] En estos se recoge toda la información relacionada con el estudio de un sujeto incluido en el ensayo. El Gerente de Datos es el encargado de la confección de los cuadernos donde establece el conjunto de variables que deben ser llenadas para los sujetos.

El tipo de las variables

Para un mejor análisis de las variables de los EC, las mismas fueron clasificadas según el tipo en: variables enteras (recogen datos numéricos enteros), reales (recogen datos numéricos decimales), de tipo fecha (recogen fechas) o de tipo cadena de caracteres (pueden recoger caracteres de todo tipo).

La dependencia entre las variables

En los CRD pueden existir dependencias entre las variables, esto se refiere a que una variable tomará valores en dependencia del valor de otra u otras variables. Por ejemplo la variable “fiebre”, tomará su valor: baja, media o alta, en dependencia del valor de la variable “temperatura”.

Rangos de las variables

Además, las variables tienen rangos específicos para los datos, por ejemplo: la variable “Edad” debe estar entre 1 y 100 años, la variable “raza” debe tomar uno de los valores: blanca, negra o mestiza.

Pudiera darse un ejemplo como este: si a la variable “tipo de persona” se le asigna el valor “joven”, entonces a la hora de llenar la variable “edad”, esta debe estar en el rango de 1 a 30 años, y así sucesivamente. En este caso se ven reflejados tanto la dependencia entre las variables (“edad” depende de “tipo de persona”), como el rango de valores que puede tomar la variable “edad”.

Así mismo, un dato puede ser el resultado de la concatenación de otros, o el resultado de una fórmula que pudiera o no incluir otros datos recogidos con anterioridad.

1.2.1 La validación de los datos recogidos en las variables

Los CRD contienen un gran volumen de información, en algunos se recogen más de mil variables para cada paciente, lo que trae consigo que a la hora de llenar los datos, puedan ocurrir múltiples errores. La veracidad de los datos para estos estudios clínicos es de suma importancia, debido a que el estudio tiene como objetivo la valoración de fármacos en la cura de enfermedades malignas en los seres humanos.

Teniendo en cuenta las características de las variables y la importancia de la autenticidad de los datos recogidos en las mismas, el proyecto Ensayos Clínicos de la UCI en acuerdo con el CIM, decide que para cada variable de un ensayo clínico, deben definirse reglas que contribuyan a validar que los datos entrados son los correctos. Estas reglas de validación serán definidas por el Gerente de Datos y deberán cumplirse a la hora de entrar los datos en los cuadernos.

1.3 Sistemas de Gestión de Ensayos Clínicos

Los ensayos clínicos constituyen el componente más costoso de todo el proceso de desarrollo de fármacos. El manejo de los mismos requiere el uso intensivo de diversos recursos dentro de una compañía biofarmacéutica y a menudo, implican procesos tediosos como recopilación manual de datos, agregación y racionalización de la información de una amplia variedad de fuentes.

El manejo electrónico de datos en estudios clínicos tiene su origen en los años setenta y desde entonces se han venido desarrollando, a nivel internacional, herramientas informáticas que le dan apoyo. Una referencia importante ha sido el marco regulatorio adelantado por la Administración de Alimentos y Medicamentos (FDA, por sus siglas en ingles) de los Estados Unidos de Norteamérica, con la publicación, en marzo de 1997, de la Regulación 21 CFR parte 11 y en abril de 1999, de la Guía para Sistemas Computarizados; los cuales representan un modelo para los estándares globales de manejo de datos en investigación clínica.

En tal sentido se han desarrollado los Sistemas de Gestión de Ensayos Clínicos (CTMS por sus siglas en inglés) a nivel mundial.

Un CTMS provee la eficiencia y el ahorro inmediato de costes de operaciones clínicas, al proporcionar un sistema único y centralizado para organizar las actividades operacionales y administrativas. Un CTMS permite a las compañías biofarmacéuticas, gestionar de forma inteligente la complejidad de los ensayos clínicos. En lugar de depender de diferentes islas de información, que a menudo generan conflictos, un CTMS permite a los usuarios compartir los datos de un ensayo en toda la empresa. Esto permite a la organización patrocinadora agilizar los procesos, responder apropiadamente a los estados actuales del ensayo y gestionar proactivamente sus decisiones. [12]

A continuación se mencionan ejemplos de CTMS:

-DataLabsXC 4.0 (de la Compañía DataLabs basada en la tecnología de Microsoft): es una plataforma de manejo de datos clínicos que incluye módulos para el diseño de estudios clínicos, la captura electrónica y el manejo de datos de EC. Esta aplicación tiene la flexibilidad de ser un sistema basado en la tecnología Web que incluye las funcionalidades de un CTMS que reduce el tiempo y costo relacionado con la gestión de ensayos clínicos. [7] Este sistema es utilizado por CARÁCTER International, la cual es una de las principales organizaciones de investigación clínica con más de 2.500 empleados que trabajan en sus oficinas de Norteamérica, Europa, Sudamérica, África

y Australia. [30]

-Medidata RAVE: este sistema es una solución de la Compañía Medidata Solutions (proveedor mundial de soluciones de gestión de datos clínicos en formato electrónico con clientes en 40 países [25]), destinada a mejorar la eficacia de los ensayos, racionalizando el proceso de recogida, verificación, consolidación y análisis de datos de investigaciones clínicas. El sistema ha sido evaluado por IBM Global Services y actualmente es usado por Bayer Healthcare AG. [9] Esta última es una filial de Bayer AG, empresa alemana, líder mundial en el ámbito de la salud y la fabricación de medicamentos y productos médicos, para el diagnóstico, la prevención y el tratamiento de enfermedades. [8]

-Çkbee.docs (Archivo electrónico de Documentación de Ensayos Clínicos): es una solución especializada para la gestión documental segura, controlada y costo efectiva de los proyectos de EC. Ofrece beneficios de ahorro de gastos, cumplimiento efectivo de normas regulatorias, ahorro de tiempo, trazabilidad y control. [21]

Los sistemas presentados anteriormente además de otros como Hipócrates y Oracle ClinTrial, son sistemas propietarios a través de los cuales las compañías que los promueven ofrecen sus servicios online.

Además de estos sistemas se presentan a continuación otros de código abierto:

-Clinical Conductor (de la compañía Clinical Systems): es una herramienta Web que ayuda a las organizaciones a coleccionar datos específicos de ensayos clínicos, a diseñar estudios, gestionar eventos adversos, monitorear los datos de los ensayos, entre otras. Entre sus beneficios, brinda visibilidad completa del estado actual de los ensayos, fácil comunicación entre los especialistas médicos, monitores, pacientes y agencias regulatorias, y aumento de la eficacia, seguridad y calidad de los ensayos. [11]

-OpenClinica (de la compañía Akaza): OpenClinica es un software de captura electrónica de datos y de gestión de datos clínicos para ensayos clínicos y estudios de investigación. Esta aplicación web está diseñada para dar cobertura a todo tipo de estudios clínicos. Está desarrollado a partir de los estándares de más prestigio para alcanzar altos niveles de interoperabilidad con otros servicios y plataformas. Su arquitectura modular, transparencia y su modelo de desarrollo colaborativo ofrecen una gran flexibilidad a la vez que permiten desplegar soluciones de alto rendimiento y escalabilidad. [28][32] Este CTMS permite la creación de EC, el manejo simultáneo de múltiples estudios, el diseño de los CRD, la inserción de pacientes en los ensayos; así

como la recogida de los datos y la realización de reportes de los mismos. Actualmente es el líder de los CTMS “Open Source”, con una comunidad de usuarios de 76 países. [3]

OpenClinica fue estudiado por el equipo de proyecto Ensayos Clínicos en conjunto con los especialistas del CIM. Como conclusión del estudio se determinó que el software cumple con el 70% de las funcionalidades necesarias para la gestión de ensayos clínicos que se realiza en el centro, por lo que se decidió desarrollar alasClínicas a partir del mismo.

1.3.1 La gestión de datos de ensayos clínicos en los CTMS

Hoy día el proceso de recoger información desde una fuente de datos e introducirla a un sistema informático, es denominado: Proceso de Entrada de Datos, generalmente es llamado por su nombre en inglés: “Data Entry”.

“Data Entry”, es definido como el proceso de entrada de información en una computadora por medio del teclado u otra herramienta electrónica. Permite capturar de varias formas los datos desde documentos en papel, imágenes, manuscritos, entre otros. [15]

El proceso “Data Entry” o de entrada de datos, para la gestión de ensayos clínicos, tiene como objetivo: lograr el manejo de datos clínicos (CDM, por sus siglas en inglés), haciendo uso del proceso de captura electrónica de datos (EDC, por sus siglas en inglés).

Los CTMS desarrollan este proceso de varias formas:

- En algunos casos se introducen los datos al CTMS a partir de los datos recogidos en los cuadernos de recogida de datos en papel. Cuando ocurre esto, primeramente se preparan bases de datos adecuadas a la investigación o ensayo clínico, que se está ejecutando. Luego se desarrolla el proceso de entrada de datos por un conjunto de personas, especialistas en entrada de datos en sistemas informáticos, más conocidas en inglés como “Data Entries”; las cuales se encargan de desarrollar el engorroso proceso de transcribir desde el papel hacia el computador, a través del teclado. Para lograr una mayor integridad de los datos en estos casos, se realiza una doble entrada de los mismos en las bases de datos; de esta forma se eliminan errores.
- En otros casos, los mismos especialistas médicos que recogen la información de los sujetos incluidos en el estudio, se conectan a la aplicación web e introducen los datos al sistema directamente desde el teclado, sin pasar por el

proceso de registrar los datos previamente en los CRD en papel. En estos casos, la captura electrónica de los datos, se realiza a través del proceso de entrada de datos en línea, más conocido como: online data entry (ODE, por sus siglas en inglés).

- Otros CTMS, utilizan tecnologías más avanzadas, permitiendo entrar los datos al sistema desde una herramienta electrónica que se le da a cada sujeto incluido en el estudio. A través de esta herramienta, el sujeto desde el lugar donde se encuentre, entra los datos que se le solicitan cada día, y de forma remota, los datos son registrados en el sistema. Esta es otra modalidad del proceso online data entry.

Varios de los sistemas que utilizan los procesos “Data Entry”, permiten el diseño de formularios para recoger los datos. Específicamente en varios CTMS, se le permite al usuario, determinar el conjunto de variables que recopilará la información de los ensayos clínicos en los formularios.

Estos sistemas permiten validar los datos recogidos, permitiendo establecer rangos de valores para las variables y otras especificidades que favorecen que los datos contengan menor cantidad de errores. Sin embargo, esta validación es sencilla, no permitiendo definir dependencias entre variables, ni rangos de valores complejos que incluyan operaciones matemáticas entre los datos. El CTMS OpenClinica por ejemplo, a partir del cual se creó alasClínicas, permite definir rangos para las variables de tipo entero y real a la vez que se diseñan los formularios, sin embargo este sistema no permite establecer dependencias que condicionen los valores de las variables a partir de los valores de otras variables.

1.4 El sistema alasClínicas

alasClínicas, es un CTMS desarrollado en la Universidad de las Ciencias Informáticas, abarca las funcionalidades de OpenClinica y además brinda otras como: la realización de un cronograma de ejecución general para los ensayos y la generación del cronograma específico para cada paciente, el monitoreo de los datos recogidos, el reporte de las trazas de las acciones realizadas en el sistema; así como la restricciones de las direcciones IP desde las cuales se puede acceder al mismo, entre otras.

Este software ha sido desarrollado con la tecnología J2EE, partiendo del código del sistema OpenClinica, basando su arquitectura en el patrón Modelo Vista Controlador.

1.4 1 La gestión de datos en alasClínicas

En este sistema, el proceso de captura electrónica de datos, se realiza online. Primeramente un usuario con el rol Gerente de Datos, es el encargado de diseñar los cuadernos de recogida de datos electrónicos donde se recoge la información. En un documento Excel que el sistema contiene por defecto, el Gerente establece las variables que serán llenadas, además de otras especificaciones propias de un CRD. Luego el sistema carga el documento y construye el formulario web en dependencia de las especificaciones establecidas. Una vez diseñados los formularios, los especialistas médicos se conectan a la aplicación para entrar los datos recogidos de los sujetos incluidos en el estudio. Esta información es almacenada en la base de datos del sistema.

Uno de los objetivos que persigue el desarrollo del sistema alasClínicas es precisamente, lograr que los datos recogidos en los CRD electrónicos contengan la menor cantidad de errores posible, contribuyendo de esta forma a la validación de los mismos. Para lograr este objetivo, se permitirá al Gerente de Datos establecer reglas de validación para cada una de las variables y el sistema deberá ejecutarlas en el momento de entrar los datos a los CRD electrónicos. A continuación se muestran los tipos de reglas que puede definir el gerente.

Reglas de validación

- Campo obligatorio: establece que el llenado de una variable es obligatorio.
- Cantidad de cifras enteras: establece la cantidad de cifras enteras de un número.
- Cantidad de cifras decimales: establece la cantidad de cifras decimales de un número.
- Cantidad de caracteres: establece la cantidad máxima de caracteres que debe tener un texto.
- Repetir: establece que una variable obtendrá el valor de otra.
- Rangos: establece los rangos de valores admisibles para una variable, estos rangos pueden o no depender del valor de otras variables y de fórmulas matemáticas.
- Fórmula: establece que una variable es el resultado de una fórmula que puede incluir el valor de otras variables.
- Concatenación: establece que un dato es el resultado de la concatenación de otros datos.

Con el propósito de ejecutar las reglas de tipo Rangos, Concatenación y Fórmula, se

propone el desarrollo de un intérprete en el sistema alasClínicas.

1.5 Intérpretes

En informática, un intérprete es un programa capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes suelen contraponerse a los compiladores, ya que mientras que los segundos se encargan de traducir un programa desde su descripción a un lenguaje de programación en código máquina del sistema destino, estos sólo realizan la traducción a medida que sea necesario y normalmente, no guardan el resultado de dicha traducción. [31]

La ventaja del proceso intérprete es que no necesita de dos etapas para ejecutar el programa, sin embargo su inconveniente es que la velocidad de ejecución es más lenta que los compilados ya que debe analizar e interpretar las instrucciones contenidas en el programa fuente, pero a cambio son más flexibles como entornos de programación y depuración y permiten ofrecer al programa interpretado un entorno del propio intérprete (lo que se conoce comúnmente como máquina virtual). Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo).

El intérprete elimina la necesidad de realizar una corrida de compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores.

Perl, PHP, Javascript, Logo, ASP (hasta la versión 3) y Python son ejemplos de lenguajes que son normalmente interpretados en vez de compilados.

1.5.1 Tipos de intérpretes

Intérpretes puros

Los intérpretes puros son los que analizan y ejecutan sentencia a sentencia todo el programa fuente. Siguen el modelo de interpretación iterativa y, por tanto, se utilizan principalmente para lenguajes sencillos.

Intérpretes avanzados

Los intérpretes avanzados o normales incorporan un paso previo de análisis de todo el programa fuente. Generando posteriormente un lenguaje intermedio que es ejecutado por ellos mismos.

De esta forma en caso de errores sintácticos no pasan de la fase de análisis. Se utilizan para lenguajes más avanzados que los intérpretes puros, ya que permiten realizar un análisis más detallado del programa fuente (comprobación de tipos, optimización de instrucciones, entre otros).

Intérpretes incrementales

Existen ciertos lenguajes que, por sus características, no se pueden compilar directamente. La razón es que pueden manejar objetos o funciones que no son conocidos en tiempo de compilación, ya que se crean dinámicamente en tiempo en ejecución. Entre estos lenguajes, pueden considerarse Smalltalk, Lisp o Prolog. Con el propósito de obtener una mayor eficiencia que en la interpretación simple, se diseñan compiladores incrementales. La idea es compilar aquellas partes estáticas del programa en lenguaje fuente, marcando como dinámicas las que no puedan compilarse. Posteriormente, en tiempo de ejecución, el sistema podrá compilar algunas partes dinámicas o recompilar partes dinámicas que hayan sido modificadas. [29]

Definición del tipo de intérprete a desarrollar:

Analizando cada uno de los tipos de intérpretes, se decide implementar uno de tipo Avanzado teniendo en cuenta que este es más completo que los puros al permitir además la comprobación de tipos, la cual será necesaria para verificar que las operaciones que se definen en las reglas, se realicen entre los tipos de datos correctos. Además, la realización de un análisis previo del código fuente, permitirá la detección de errores, garantizando que las reglas de validación se declaren de forma correcta en el sistema.

No se opta por desarrollar un intérprete de tipo incremental debido a que en la solución no se generarán objetos o funciones en tiempo de ejecución.

1.5.2 Estructura de un intérprete

Los intérpretes de tipo avanzado, generalmente están estructurados en varias fases: Análisis Léxico, Análisis Sintáctico, Análisis Semántico, Generación de Código Intermedio y Evaluación. A continuación se describen cada una de estas fases.

Análisis Léxico

La primera fase de un intérprete se llama Análisis Léxico o de Exploración. El Analizador Léxico lee la secuencia de caracteres que componen el programa o código de origen, y los grupos de caracteres en secuencias significativas son llamados

lexemas. Para cada lexema, el analizador léxico produce como salida un token[1] y de esta forma pasa a la fase siguiente. [2]

Los tokens que constituyen variables o constantes se guardan en una Tabla de Símbolos, donde además se puede guardar el valor u otras características de cada token. La información guardada en la tabla de símbolos es necesaria para el análisis semántico y la generación de código.

Estructura con que se representan los tokens:

(tipo del token, atributo)

El primer componente del token (tipo del token) es una categoría sintáctica (por ejemplo: identificador, constante) y se utiliza en la fase de análisis de la sintaxis, y el segundo componente (atributo) brinda información relacionada con el token en particular (por ejemplo: valor, índice en la tabla de símbolos).

Ejemplo de Análisis Léxico:

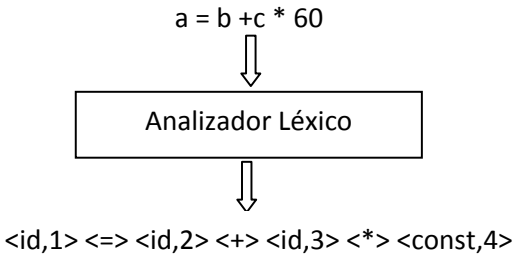


Tabla 1: Tabla de símbolos

Índice	Lexema	Categoría	Tipo	Valor
1	“a”	id	Real	
2	“b”	id	Real	
3	“c”	id	Real	
4	“60”	constante	Real	60

Análisis Sintáctico

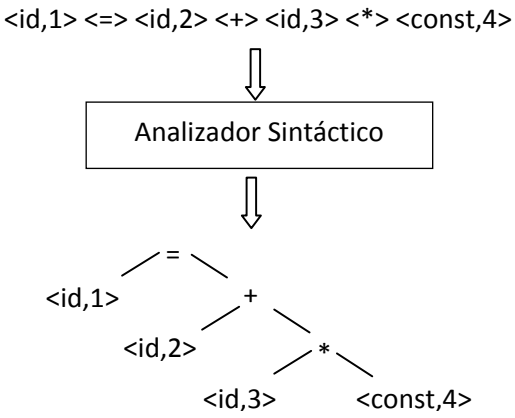
La segunda fase del intérprete es el Análisis Sintáctico. El analizador utiliza los tokens producidos por el Analizador Léxico para crear una representación intermedia en

[1]En español: Ficha o elemento. Más usado con el término en inglés: token.

forma de árbol que representa la estructura gramatical. Una representación típica es un árbol de sintaxis en la que cada nodo interior representa una operación y los hijos de los nodos representan los argumentos de la operación. El árbol de sintaxis constituye la salida del Análisis Sintáctico. [2]

En esta fase se examina la secuencia de tokens (utilizando sólo la primera componente de los tokens) para determinar si el orden de esa secuencia es correcto de acuerdo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje.

Ejemplo de Análisis Sintáctico:



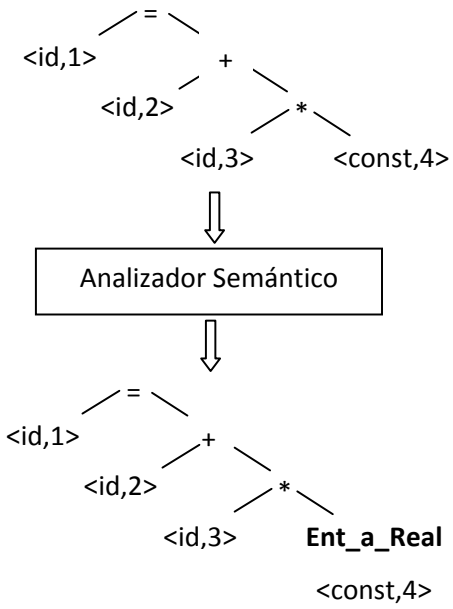
Análisis Semántico

El analizador semántico utiliza el árbol de sintaxis y la información de la tabla de símbolos para revisar el código del programa. También recoge el tipo de información y lo guarda ya sea en el árbol de sintaxis o en la tabla de símbolos, para su posterior utilización en la generación de código intermedio. [2]

Ejemplo de errores que pueden ser detectados en el proceso de análisis semántico son los casos de compatibilidad entre la declaración de un identificador y su uso (chequeo de tipos), la concordancia entre la definición de una función y su activación o llamada, entre otros. [1]

En el ejemplo que se está analizando, se puede notar que haciendo un chequeo de tipos, la constante 60, al ser una constante entera debe ser convertida a real (60.0) para poder ser operada, y las demás variables no tienen dificultad alguna para ser utilizadas al ser todas reales.

Ejemplo de Análisis Semántico:



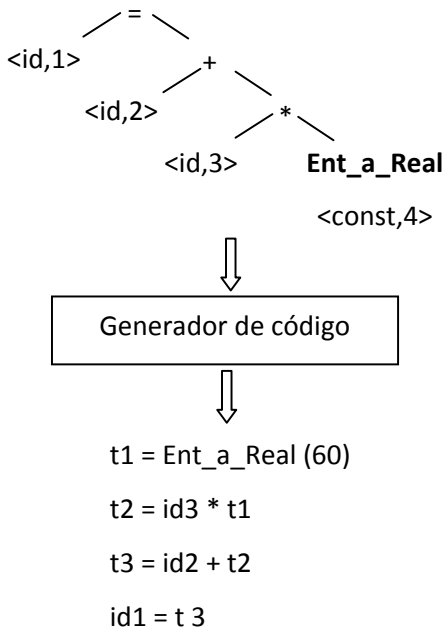
Generación de Código Intermedio

En el proceso de traducir un programa fuente a un código destino, un intérprete puede construir una o más representaciones intermedias, que pueden tener una variedad de formas. Árboles de sintaxis son una forma de representación intermedia, son de uso común durante la sintaxis y el análisis semántico.

Después de los análisis sintácticos y semánticos del programa de origen, muchos intérpretes generan una representación intermedia explícita de bajo nivel que se puede considerar como la representación de un programa para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir en el equipo de destino. [2]

Un método elegante y efectivo para generar el código intermedio a partir del árbol sintáctico es el denominado Traducción Sintáctica Directa. A cada nodo n se le asocia un código intermedio C(n). El código del nodo n se construye concatenando el código de sus descendientes. Luego, la traducción se hace de abajo hacia arriba. [1]

Ejemplo de generación de código intermedio:



En general, se puede hablar de formas de representación de código intermedio, orientadas a pilas, y orientadas a variables temporales como la técnica de Traducción Sintáctica Directa explicada anteriormente y la Notación de Cuádruples. Entre la representación de código intermedio orientada a pila se encuentra la Notación Polaca, en esta notación los operandos aparecen primero que los operadores, obteniendo finalmente la llamada cadena polaca. [1]

Expresión:

id1 = id2 + id3 * 60

Notación Polaca:

id1, id2, id3, 60, *, +, =

Evaluación

Finalmente, esta etapa es la encargada de ejecutar el código intermedio obtenido en la fase anterior, generando el resultado de la interpretación.

1.5.3 Herramientas para desarrollar las fases de un intérprete

Con el desarrollo de la Informática, se ha logrado la implementación de herramientas que son capaces de generar algunas fases de un intérprete. A continuación se describen algunas de estas herramientas que generan el código en lenguaje Java.

Jlex

El Jlex es un generador de analizadores léxicos diseñado para Java, creado en la Universidad de Princeton. Esto significa que Jlex es capaz de generar automáticamente un analizador léxico a partir de una especificación léxica del lenguaje. Un analizador léxico toma como entrada una cadena de caracteres y la transforma en una secuencia de tokens. Jlex toma como entrada un archivo con una especificación exp, y con ella crea un archivo fuente Java exp.lex.java correspondiente al analizador léxico, que tras ser compilado da como resultado el analizador Yylex.class. [26]

Jflex

Jflex es otro generador de analizadores léxicos, el cual toma como entrada una cadena de caracteres, y lo convierte en una secuencia de tokens. Para la generación de estos tokens se define previamente el fichero de análisis léxico, en el cual se añaden reglas gramaticales que definirán el reconocimiento de patrones los cuales serán los que dicten la formación de tokens. Es decir, en el fichero ".lex" se definen reglas tales como que el valor entrado es un identificador, un entero, una cadena, entre otros. Estas reglas formarán los tokens, palabras reservadas o símbolos terminales de la gramática.

Una vez definidas estas reglas, se le pasa un flujo de datos el cual será examinado carácter a carácter. Si alguna secuencia de estos caracteres coincide con alguna regla especificada en el fichero de especificación léxica, el JFLEX formará un token y se lo devolverá al programa desde el cual se instanció el objeto de JFLEX. En el fichero de especificación la prioridad es descendente, esto quiere decir que es posible que una secuencia de los caracteres que se están analizando pueda coincidir con varias reglas definidas, pues bien, siempre se elegirá la regla situada más arriba en el fichero para formar un token. [19]

Java CUP

JavaCUP fue desarrollado en la Universidad de Pricenton por Scott Hudson, 1995. Basa su diseño en el analizador YACC (creado en lenguaje C), pero está completamente implementado en java. Es una herramienta desarrollada para crear analizadores sintácticos LALR[2]. Genera dos clases en Java, por defecto: sym y parser. [24]

[2] Analizadores sintácticos LALR (Look Ahead Left to Rigth): analizadores sintácticos ascendentes de búsqueda hacia adelante. (30)

• parser.java: contiene la implementación del analizador sintáctico que se ha definido previamente dentro del fichero especificacionSintactica.cup (el nombre de este fichero es elegido por el usuario).
• sym.java: contiene el conjunto de símbolos terminales que se han definido en la especificación sintáctica.

Las especificaciones sintácticas que toma CUP a su entrada se dividen en tres partes principales:

• Definición de símbolos terminales.
• Definición de símbolos no terminales.
• Especificación de la gramática. [17]

Jflex y JavaCUP

La unión del analizador léxico generado con JFLEX y del analizador sintáctico generado por CUP es sencilla. Basta con:
• Emplear el mismo juego de símbolos terminales tanto en la especificación léxica realizada para JFLEX como en la especificación sintáctica que se le pretende pasar a CUP.
• Habilitar la compatibilidad de JFLEX con CUP mediante la directiva %cup.
• Devolver los tokens leídos mediante la sentencia “return” de Java dentro de la especificación léxica creando instancias de la clase sym, clase que maneja CUP para la gestión interna de los tokens recibidos desde el analizador léxico.
• Crear una instancia del analizador léxico creado y pasárselo como argumento al analizador sintáctico, de tal forma que éste le vaya proporcionando los tokens leídos desde el fichero de entrada. [23]

ANTLR

ANTLR (Another Tool for Language Recognition; en español: “otra herramienta para reconocimiento de lenguajes”), es una herramienta que opera sobre lenguajes, proporcionando un marco para construir reconocedores (parsers), intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos.

ANTLR genera un programa que determina si una sentencia o palabra pertenece a dicho lenguaje (reconocedor), utilizando algoritmos LL(*)[3].

[3] Algoritmos LL(*): leen la entrada de izquierda a derecha y producen una derivación por la izquierda. El * se relaciona con el número de símbolos necesarios en el pre-análisis. (30)

Proporciona facilidades para la creación de estructuras intermedias de análisis (como árboles de sintaxis abstracta), para recorrer dichas estructuras, y provee mecanismos para recuperarse automáticamente de errores y realizar reportes de los mismos. [34][33]

Desventajas del uso de estas herramientas

Como desventaja que poseen estas herramientas, el mantenimiento del código que generan resulta complicado, es más difícil de depurar en caso de realización de cambios en alguna estructura del lenguaje o de la gramática; esto implica que un pequeño cambio conlleve a generar nuevamente los analizadores; o en caso contrario, el programador deberá entender toda la estructura del código para realizar modificaciones y esto suele ser engorroso. Analizando estas desventajas se determina no utilizar alguna de estas herramientas para el desarrollo del intérprete.

1.6 Método para evaluar el intérprete en la validación de los datos

Una vez desarrollado el intérprete, el mismo debe ser evaluado para verificar su funcionamiento en la validación de los datos de los ensayos clínicos, para lo cual se propone el uso de un método de evaluación.

Dentro de los métodos generales de prospectiva cabe destacar aquellos que se basan en la consulta a expertos, que reciben la denominación de Métodos de Expertos. Estos métodos utilizan como fuente de información un grupo de personas a las que se supone un conocimiento elevado de la materia que se va a tratar.

Los métodos de expertos tienen las siguientes ventajas:

- La información disponible está siempre más contrastada que aquella de la que dispone el participante mejor preparado, es decir, que la del experto más versado en el tema. Esta afirmación se basa en la idea de que varias cabezas son mejor que una.
- El número de factores que es considerado por un grupo, es mayor que el que podría ser tenido en cuenta por una sola persona. Cada experto podrá aportar a la discusión general, la idea que tiene sobre el tema debatido desde su área de conocimiento.

Estos métodos también presentan deficiencias relacionadas con: la desinformación que puede tener cada individuo, la presión social que el grupo ejerce sobre los participantes en particular, en ocasiones los grupos son vulnerables a la posición y personalidad de algunos de los individuos, hay veces que el argumento que triunfa es el más citado en lugar del más válido, entre otras. [20]

1.6.1 Método Delphi

El método de expertos ideal sería aquel que extrajese los beneficios de la interacción directa de los participantes y eliminase sus inconvenientes. Esta intenta ser la filosofía del método Delphi.

Características

El método Delphi pretende extraer y maximizar las ventajas que presentan los métodos basados en grupos de expertos y minimizar sus inconvenientes. Para ello se aprovecha la sinergia del debate en el grupo y se eliminan las interacciones sociales indeseables que existen dentro de todo grupo. De esta forma se espera obtener un consenso lo más fiable posible del grupo de expertos.

Este método presenta tres características fundamentales:

1. Anonimato: durante un Delphi, ningún experto conoce la identidad de los otros que componen el grupo de debate. Esto tiene una serie de aspectos positivos, como son:
   - Impide la posibilidad de que un miembro del grupo sea influenciado por la reputación de otro de los miembros o por el peso que supone oponerse a la mayoría. La única influencia posible es la de la congruencia de los argumentos.
   - Permite que un miembro pueda cambiar sus opiniones sin que eso suponga una pérdida de imagen.
   - El experto puede defender sus argumentos con la tranquilidad que da saber que en caso de que sean erróneos, su equivocación no va a ser conocida por los otros expertos.
2. Iteración y realimentación controlada: la iteración se consigue al presentar varias veces el mismo cuestionario. Como, además, se van presentando los resultados obtenidos con los cuestionarios anteriores, se consigue que los expertos vayan conociendo los distintos puntos de vista y puedan ir modificando su opinión si los argumentos presentados les parecen más apropiados que los suyos.
3. Respuesta del grupo en forma estadística: la información que se presenta a los expertos no es sólo el punto de vista de la mayoría, sino que se presentan todas las opiniones indicando el grado de acuerdo que se ha obtenido.

Aunque la formulación teórica del método Delphi propiamente dicho comprende varias etapas sucesivas de envíos de cuestionarios, de vaciado y de explotación, en buena parte de los casos puede limitarse a dos etapas, lo que sin embargo no afecta a la calidad de los resultados tal y como lo demuestra la experiencia acumulada en estudios similares.

Como es sabido, el objetivo de los cuestionarios sucesivos, es disminuir el espacio intercuartil (cuánto se desvía la opinión del experto de la opinión del conjunto), precisando la mediana de las respuestas obtenidas. El objetivo del primer cuestionario es calcular el espacio intercuartil. El segundo suministra a cada experto las opiniones de sus colegas, y abre un debate transdisciplinario, para obtener un consenso en los resultados y una generación de conocimiento sobre el tema. Cada experto argumentará los pros y los contras de las opiniones de los demás y de la suya propia. [13]

Teniendo en cuenta las ventajas y características del método Delphi, sería provechosa la utilización del mismo en la presente investigación, donde un grupo de expertos del Centro de Inmunología Molecular podrían ser los encargados de valorar la utilidad del intérprete en la validación de datos de los ensayos clínicos.

Conclusiones:

En el presente capítulo se realizó un estudio de las características de las variables que se recogen en los ensayos clínicos, se hizo un análisis de los intérpretes como traductores de lenguajes de alto nivel, y se arribó a las siguientes conclusiones:

- Se determina la implementación de un intérprete en el sistema alasClínicas, que analice y ejecute las reglas de validación de tipo: Rangos, Concatenación y Fórmula, definidas para los datos de los EC.
- Analizando las desventajas que presentan las herramientas generadoras de código, se decide no utilizarlas para el desarrollo del intérprete.
- De esta forma se determinó realizar la implementación de todo el código necesario para el desarrollo de un intérprete de tipo avanzado para la validación de los datos de los ensayos clínicos, en el sistema alasClínicas.
- Se evaluará la funcionalidad del intérprete haciendo uso del método de expertos Delphi.

CAPÍTULO 2: DESARROLLO DE LA SOLUCIÓN

Introducción

El presente capítulo aborda el desarrollo del intérprete en cada una de sus fases, así como las características del mismo. Una vez implementado, será utilizado para la declaración y ejecución de las reglas de tipo Rangos, Concatenación y Fórmula; por lo que se utilizará en dos etapas:

- Primeramente, en la declaración de las reglas, para determinar si las mismas han sido especificadas correctamente por el Gerente de Datos. En caso contrario, se mostrarán mensajes de error al usuario.
- Posteriormente, en la ejecución de las reglas, cuando una vez definidas, sean invocadas en el llenado de los datos en los CRD. En esta etapa, el intérprete verificará el cumplimiento de las reglas en la entrada de los datos. Los mensajes de error indicarán que los datos no cumplen con las reglas requeridas.

2.1 Fase: Análisis Léxico

El analizador léxico es la primera fase de un intérprete. Su función principal es leer los caracteres de entrada y producir como salida una secuencia de componentes léxicos (tokens) que el parser luego utiliza en el proceso de análisis sintáctico. La interacción entre ambos componentes (analizador léxico y analizador sintáctico) se resumen de la siguiente forma:

Ante un pedido de un componente léxico (mensaje: “próximo componente léxico”), el analizador léxico lee una secuencia de caracteres del código fuente del programa hasta identificar un componente léxico, el cual es retornado como respuesta al pedido del analizador sintáctico. [1]

Para llevar a cabo la tarea del analizador léxico, primeramente se procede a definir el lenguaje que será interpretado. A partir del mismo, se determinan las expresiones regulares que representan los tokens.

2.1.1 Definición del lenguaje

El lenguaje utilizado para la definición de las reglas de validación, es un lenguaje regular con estructuras claramente definidas y determinadas por sus reglas gramaticales (sintácticas y semánticas).

Las reglas de validación de tipo Rangos, establecen los rangos de valores que pueden tomar las variables de tipo entero, real y fecha. Para la definición de los rangos se utiliza el siguiente formato:

Vx > V1 + V2 * 3

Donde Vx constituye la variable para la cual se define la regla y V# son las variables de las cuales depende la misma y que fueron definidas con anterioridad en el sistema.

Las reglas de validación de tipo Concatenación, pueden ser aplicadas a las variables de tipo texto y determinan que un dato es el resultado de la concatenación de otros datos. Para representar esta regla se utiliza el siguiente formato:

V1-V3-V2

En este ejemplo, se permite concatenar los datos recogidos en las variables V1, V2 y V3 en el orden en que se muestra.

Las reglas de tipo Fórmula, pueden ser aplicadas a las variables de tipo entero y real, y establecen que un dato es el resultado de una fórmula que involucra a otros datos. Esta regla se puede representar de la siguiente forma:

V1 + Ln (V2) * 5

Características del lenguaje según el tipo de la regla:

Características	Reglas
El lenguaje admite la variable: Vx.	Rangos.
El lenguaje admite variables: V#, donde # representa un número entero.	Rangos, Concatenación y Fórmula.
Se admiten constantes (valores numéricos: enteros y reales).	Rangos y Fórmula.
Las operaciones que admiten las reglas son: suma (+), resta (-), división (/), multiplicación (*), logaritmo (Ln), raíz cuadrada (Sqrt).	Rangos y Fórmula.
Los operadores relacionales usados para las comparaciones son: mayor (>), menor (<), mayor e igual (>=), menor e igual (<=), igual (=) y diferente (¡=).	Rangos.
Se utilizan los operadores lógicos: and (&) y or (|).	Rangos.

Se utilizan los paréntesis para definir la prioridad de las operaciones.	Rangos y Fórmula.
Se utiliza el singo – para la concatenación de los datos.	Concatenación.
Se utiliza el formato “dd/mm/aaaa” para las fechas.	Rangos.

Ejemplos de reglas de tipo Rangos:

- (Vx > V1) & (Vx < V2) En este caso, la variable deberá ser mayor que V1 y menor que V2.
- (Vx > V1) | (Vx < V2) En este caso, la variable deberá ser mayor que V1 o menor que V2.

De igual forma se podrá tener un caso como el siguiente:

- ((Vx > V1) & (Vx < V2)) | (Vx < (V1 + 100))

En las reglas tipo Rangos:

- para las variables de tipo entero y real, se utilizará en el sistema el siguiente componente (Figura 1):

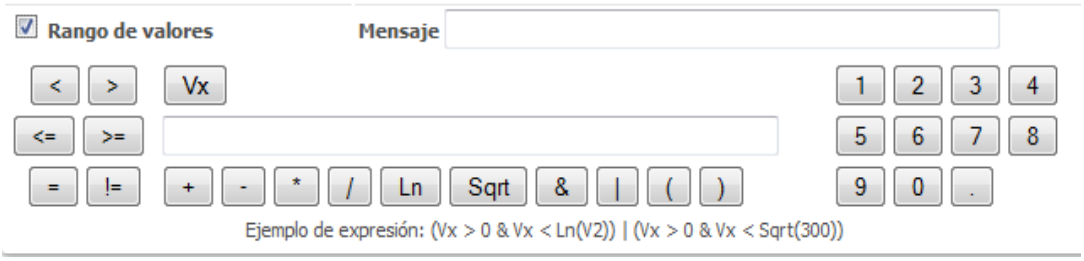


Figura 1: Rango de valores.

- para las variables de tipo fecha no se utilizan operaciones, sólo los operadores de comparación. Se utiliza en el sistema el siguiente componente (Figura 2):

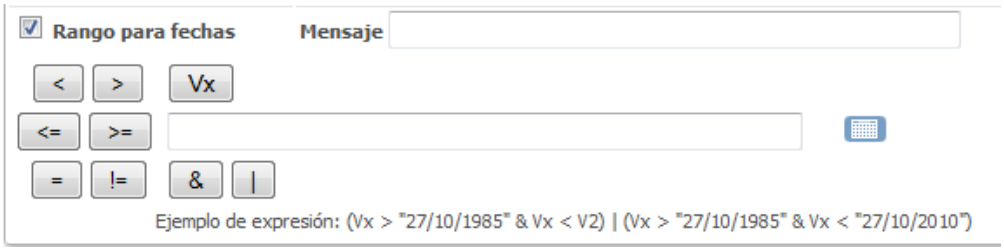


Figura 2: Rango de valores para fechas.

Para las reglas de tipo Concatenación, se utilizará el componente (Figura 3):

Figura 3: Concatenación.

Para las reglas de tipo fórmula, se utilizará el componente (Figura 4):

Figura 4: Fórmula.

2.1.2 Expresiones regulares

A continuación se describen los tipos de token utilizando expresiones regulares, teniendo en cuenta las características del lenguaje.

Tipo de Token = Expresión regular	Descripción
P_ABIERTO = (	Paréntesis abierto.
P_CERRADO =)	Paréntesis cerrado.
O_SUMA = +	Operador de suma.
O_RESTA = -	Operador de resta.
O_MULTIPLICACION = *	Operador de multiplicación.
O_DIVISION = /	Operador de división.
O_MENOR = <	Operador relacional “menor”.
O_MENOR_I = <=	Operador relacional “menor o igual”.
O_MAYOR = >	Operador relacional “mayor”.
O_MAYOR_I = >=	Operador relacional “mayor o igual”.
IGUAL = =	Operador relacional “igual”.

DISTINTO = !=	Operador relacional “distinto”.
AND = &	Operador lógico “and”.
OR = |	Operador lógico “or”.
VARIABLE = Vx | V#	Variable: Vx, V# (# representa un número entero).
LN = Ln	Logaritmo Neperiano.
SQRT = Sqrt	Raíz cuadrada.
CONSTANTE = *número entero o real*	Número entero o real.
FECHA = “dd/mm/aaaa”	Fecha.

Una vez definido el lenguaje y las expresiones regulares que representan los tokens, se procede a desarrollar un esquema reconocedor, el cual tiene como tarea principal: la identificación de los tokens en la cadena que se desee interpretar.

2.1.3 Esquema reconocedor de tokens

Un reconocedor para el lenguaje L es un programa (o mecanismo) que toma como entrada una cadena x y responde “si”, si x pertenece al lenguaje L, o “no”, si no pertenece. [1]

Existe una gran variedad de tipos de reconocedores, entre ellos se encuentran los Autómatas Finitos, los Autómatas de Pila y la Máquina de Turing.

En la presente investigación se hace uso de un Autómata Finito (AF) para reconocer cada uno de los tokens. Este autómata es el más utilizado cuando se trata de lenguajes regulares. En este caso, cada uno de los tokens sigue una regularidad determinada, dada por la expresión regular que lo caracteriza.

Un autómata finito o máquina de estado finito es una herramienta abstracta que se utiliza para reconocer un determinado lenguaje regular. Es un modelo matemático de un sistema que recibe una cadena constituida por caracteres de cierto alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce. [1]

Si un lenguaje es posible describirlo a través de una expresión regular, entonces es posible también obtener el autómata finito que lo reconoce. Este resultado es de gran importancia para construir el analizador léxico de un intérprete, pues para su implementación, bastaría con describir los tokens del lenguaje a través de expresiones regulares y obtener los autómatas finitos correspondientes.

Un autómata finito está definido por una 5-tupla N = (S, ∑, δ, S_0, F). Donde:

S: Es un conjunto finito de estados.

∑: Alfabeto del lenguaje. Conjunto de símbolos de entrada del autómata.

δ : Función de transición. δ (x ∈ ∑, S_n ∈ S) → S_m:. Es la función que determina a partir de un estado y un símbolo del alfabeto, un nuevo estado.

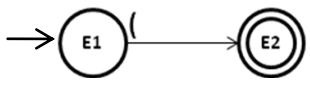
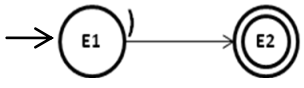
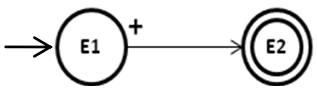
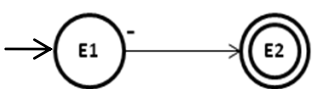
S_0: Es el estado inicial del autómata S_0 ∈ S.

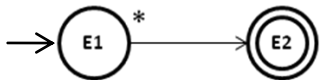
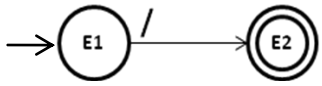
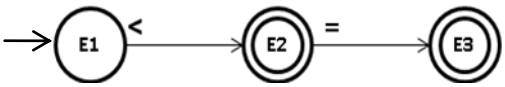
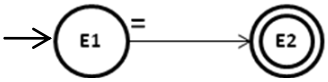
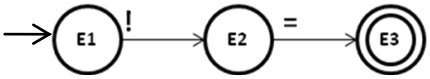
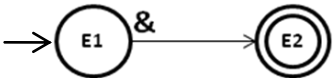
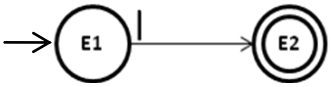
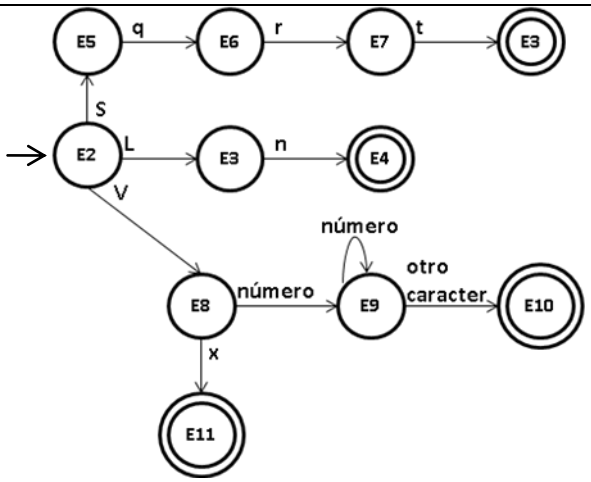
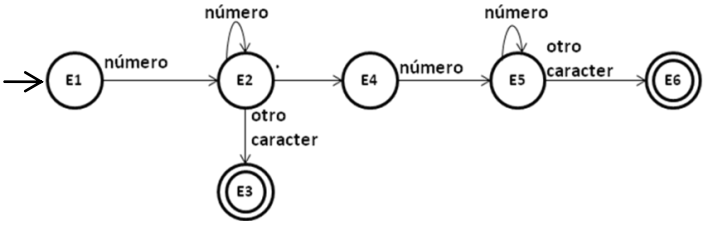
F: Es el conjunto de estados finales o de aceptación del autómata. F ⊆ S.

Existen varias formas de representar un autómata finito. Está la representación gráfica conocida como diagrama de Moore. En dicha representación los estados se representan por círculos rotulados con el nombre del estado. El estado inicial tiene asociado un arco entrante y los finales se representan por un doble círculo. Además las transiciones se representan con arcos entre dos estados etiquetados por el símbolo por el cual se pasa a dicho estado.

Inicialmente el AF se encuentra en el estado inicial y a medida que procesa cada símbolo de la cadena va cambiando de estado de acuerdo a lo determinado por la función de transición. Cuando se ha procesado el último de los símbolos de la cadena de entrada, el autómata se detiene. Si el estado en el que se detuvo es un estado de aceptación o final, entonces la cadena pertenece al lenguaje reconocido por el autómata; en caso contrario, la cadena no pertenece a dicho lenguaje.

A continuación se presentan los autómatas independientes para reconocer cada uno de los tokens.

Token = Expresión regular	**Autómata**
P_ABIERTO = (	
P_CERRADO =)	
O_SUMA = +	
O_RESTA = -	

O_MULTIPLICACION = *	
O_DIVISION = /	
O_MENOR = < O_MENOR_I = <=	
O_MAYOR = > O_MAYOR_I = >=	
IGUAL = =	
DISTINTO = !=	
AND = &	
OR = |	
VARIABLE = Vx | V# LN = Ln SQRT = Sqrt	
CONSTANTE = *número entero o real*	

FECHA = “dd/mm/aaaa”	 En este caso se utilizó una librería de Java para comprobar la fecha.

2.1.4 Descripción de las clases utilizadas en el análisis léxico

Clase: Token

A partir de esta clase es posible crear los objetos Token con los atributos: tipo de token, lexema y la posición del mismo en la tabla de símbolos (para los tokens de tipo constante o identificador).

Clase: Simbolo

Se puede decir que los símbolos constituyen los tokens de tipo constante o identificador (variable), que contienen la información relacionada con: el tipo de dato, lexema, si está declarado o no, si está inicializado o no y la dirección donde se encuentra su valor en memoria.

Clase: TSimbolo

Esta clase contiene la lista de símbolos identificados en la cadena, garantizando una posición para cada símbolo y que los mismos no se repitan en la lista.

Clase: ALexico

Esta clase es la encargada de analizar cada uno de los caracteres de la cadena entrada a través de los autómatas implementados. A medida que va analizando cada caracter, va creando los tokens. Contiene una referencia a la instancia tabla de símbolos de la clase TSimbolos y va guardando los símbolos encontrados en la tabla.

2.1.5 Detección de errores

En esta fase, a medida que se examina cada caracter de la cadena, el analizador va reconociendo los tokens del lenguaje. En caso de que un caracter no sea el que se espera para formar un token, o el caracter no pertenezca al lenguaje, el analizador léxico crea alguno de los siguientes errores:

- "El caracter “x” no pertenece al lenguaje."
- En caso de formar una frase incorrecta, se muestra el mensaje: "Frase no reconocida: “x”".
- Para el caso de las fechas, cuando ocurre un error se muestra el mensaje: "No se pudo obtener una fecha a partir de la cadena: “x”".

- En el caso particular en que se desee formar un token de tipo constante (número entero) con una cadena muy larga de números, se muestra el error: “No se pudo formar un numero con la cadena " x ", es demasiado larga.".

Mientras el analizador va reconociendo errores, los va guardando en una lista y continúa analizando el siguiente carácter de la cadena. De esta forma, se garantiza que el analizador no se detenga, sino que continúe con el análisis y la detección de errores. Al final devuelve el token de fin de fichero.

2.2 Fase: Análisis Sintáctico

La función principal de la fase de Análisis Sintáctico es comprobar si la secuencia de tokens que representa al texto de entrada, en base a una gramática dada, es correcta. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico. [1]

2.2.1 Aspectos importantes en el desarrollo del Analizador Sintáctico

Antes de proceder a la implementación del Analizador Sintáctico, se realizó un estudio de aspectos importantes que se deben tener en cuenta para que el analizador desarrollado cumpla eficazmente con su función.

Gramática libre de contexto

Para realizar la función principal del Analizador Sintáctico, se debe especificar la gramática que será utilizada para el análisis, a partir de la cual se determina si la secuencia de tokens es correcta.

Existen varios tipos de gramáticas, la utilizada en la presente investigación es una Gramática Libre de Contexto (GLC).

Una GLC es una gramática formal en la que cada regla de producción es de la forma:

$Exp \rightarrow x$

Donde Exp es un símbolo no terminal y x es una cadena de terminales y/o no terminales. El término independiente del contexto se refiere al hecho de que el no terminal Exp puede siempre ser sustituido por x sin tener en cuenta el contexto en el que ocurra. Los lenguajes libres de contexto constituyen la base teórica para la sintaxis de la mayoría de los lenguajes de programación. Definen la sintaxis de las declaraciones, las proposiciones, las expresiones, entre otros (es decir, la estructura de un programa).

Una GLC está compuesta por 4 elementos:

1. Símbolos terminales (elementos que no generan nada).

2. No terminales (elementos del lado izquierdo de una producción, antes de la flecha).

3. Producciones (sentencias que se escriben en la gramática).

4. Símbolo inicial (primer elemento de la gramática) [20] [13].

Una GLC puede ser utilizada para especificar la estructura sintáctica de un lenguaje de programación y además constituye la base para los diferentes esquemas de traducción.

Durante el proceso de interpretación, se puede utilizar la estructura sintáctica que una GLC le impone a un programa, para determinar su traducción. Utilizando las reglas sintácticas y la secuencia de producciones usadas para derivar una cadena, se puede obtener la estructura sintáctica de la cadena (o programa). O sea, el analizador sintáctico es un mecanismo que trata de determinar si una cadena puede ser derivada, a través de una GLC. [1]

El árbol sintáctico o de derivación de una gramática

Además de la definición de las reglas sintácticas de la gramática para analizar la secuencia de tokens, los analizadores sintácticos construyen implícitamente un árbol. A este árbol se le conoce como árbol sintáctico y contiene todos los tokens que aparecen en la cadena de entrada. Sin embargo no todos estos tokens son necesarios en las fases posteriores (Análisis Semántico), ejemplo de ellos son los delimitadores que tienen una función desde el punto de vista sintáctico pero dejan de ser útiles en fases posteriores. El objetivo es diseñar una representación intermedia que se produzca en el proceso de análisis sintáctico, que contenga solo las estructuras útiles a las fases posteriores. A esta representación se le llama Árbol de Sintaxis Abstracta, más conocida por sus siglas en ingles (AST). En un AST las relaciones sintácticas quedan definidas implícitamente en la estructura de árbol y como criterio se almacenan solo los elementos más importantes.

Para imaginar cómo debe lucir un AST se utilizan en su diseño algunos criterios definidos.

- Densos: no contienen nodos innecesarios.
- Convenientes: fáciles de recorrer.
- Significantes: enfatiza en los operandos y operadores, y en las relaciones entre ellos, más que en artefactos de la gramática.

La idea central detrás de una estructura de AST es que todos los tokens que representen operaciones u operadores sean tomados como raíces de subárboles. El resto de los tokens serán operandos (hijos de nodos operadores). [1]

Analizador sintáctico descendente

Existen dos tipos de analizadores sintácticos: los ascendentes y los descendentes, clasificados teniendo en cuenta la forma en que construyen el árbol sintáctico.

Los analizadores sintácticos descendentes son lo que recorren el árbol sintáctico de la sentencia a reconocer, de una forma descendente, comenzando por el símbolo inicial o raíz, hasta llegar a los símbolos terminales que forman la sentencia usando derivaciones más a la izquierda.

El analizador sintáctico ascendente, intenta recorrer el árbol sintáctico, empezando desde las hojas (la cadena) y ascendiendo hacia la raíz. Lo que es lo mismo: que intentar obtener una reducción desde una cadena hasta llegar al axioma (primer símbolo no terminal por el que se deriva). [14]

La diferencia entre uno y otro, es que del primer tipo se pueden construir manualmente analizadores más eficientes con mayor facilidad, mientras que del segundo se puede manejar una mayor cantidad de gramáticas y esquemas de traducción. [5]

Teniendo en cuenta estas diferencias, y que el intérprete que se desarrolla no posee una gramática de complejidad alta, se decide desarrollar un analizador sintáctico descendente. A pesar de las ventajas que tiene este analizador, el análisis descendente posee los siguientes inconvenientes, que pueden solucionarse.

- La recursividad a izquierdas da lugar a un bucle infinito de recursión.
- Problemas de indeterminismo: cuando varias alternativas en una misma producción comparten el mismo prefijo, no se sabría cuál elegir. Se probaría con una y si se produce un error ocurriría un retroceso.

Lo ideal es que el método fuera determinista para lo cual, habría que evitar estos problemas.

Analizador sintáctico descendente de tipo LL1

Como se mencionó anteriormente, un problema que se presenta con el análisis sintáctico descendente, es que a partir del nodo raíz, el analizador sintáctico no elija las producciones adecuadas para alcanzar la sentencia a reconocer. Cuando el analizador se da cuenta de que se ha equivocado de producción, se tienen que deshacer las producciones aplicadas hasta encontrar otras producciones alternativas,

volviendo a tener que reconstruir parte del árbol sintáctico. A este fenómeno se le denomina retroceso, vuelta atrás, o en inglés: backtracking.

Para eliminar el retroceso (backtracking) en el análisis descendente, se ha de elegir correctamente la producción correspondiente para cada no terminal que se expande. Es decir que el análisis descendente ha de ser determinista, y sólo se debe permitir tomar una opción en la expansión de cada no terminal. Para lograr esto las gramáticas han de cumplir ciertas condiciones:

- No recursiva a izquierda: se refiere a que el primer elemento de la derivación de la regla no sea el no terminal de la regla.
- No ambigua: se refiere a que una misma sentencia, ejemplo a+b*c, no produzca dos árboles diferentes.
- Factorizada por la izquierda: si una regla posee varias producciones, no hayan dos o más producciones que empiecen iguales.

Estas condiciones generalmente conducen a un tipo de gramática especial dentro de las gramáticas libres de contexto, denominada LL1. [1]

Analizador sintáctico predictivo recursivo

Los analizadores sintácticos predictivos son capaces de decidir qué regla de producción aplicar a cada paso en función de los elementos terminales que se encuentran en la cabeza de lectura de la cadena de entrada. Para realizar la predicción, se debe tener en cuenta las características antes mencionadas de la gramática LL1.

La recursividad en estos analizadores, se evidencia en la definición de las reglas de la gramática, las cuales pueden ser recursivas, es decir, la gramática queda expresada a través de llamadas explícitas a distintas funciones asociadas a los no terminales. [18]

2.2.2 Desarrollo del analizador sintáctico descendente predictivo recursivo de tipo LL1

Un parser descendente predictivo recursivo de tipo LL(1) analiza la estructura de una secuencia de tokens que forman una frase, teniendo en cuenta en cada paso el primero de los tokens no reconocidos. [1]

Teniendo en cuenta las características de este tipo de analizadores sintácticos, se definieron las reglas de la gramática de forma que cumplan con las características de una Gramática Libre de Contexto de tipo LL1.

Reglas de la gramática

Se considera a cada regla de la gramática como la definición de una función o método que reconocerá al No Terminal de la parte izquierda. El lado derecho de la producción especifica la estructura del código para ese método o función.

1. Los símbolos terminales corresponden a concordancias con la entrada.
2. Los símbolos no-terminales con llamadas a funciones o métodos.
3. Las diferentes alternativas corresponden a casos condicionales en función de lo que se esté examinando en la entrada. [1]

A continuación se muestran las reglas especificadas:

Reglas
<expresión> -> <Operando-Lógico> <Más-Operandos-Lógicos>
<Más-Operandos-Lógicos> -> tk_or <Operando-Lógico> <Más-Operandos-Lógicos> | e
<Operando-Lógico> -> <Otro-Operando-Lógico> <Otros-Operandos-Lógicos>
<Otros-Operandos-Lógico> -> tk_and <Otro-Operando-Lógico> <Otros-Operandos-Lógico> | e
<Otro-Operando-Lógico> -> <Operando-Relacional> <Otro-Operando-Relacional>
<Otro-Operando-Relacional> -> <Operador-Relacional> <Operando-Relacional> | e
<Operando-Relacional> -> <Término> <Más-Término>
<Más-Termino> -> <Operador1> <Término> <Más-Término> | e
<Término> -> <Factor> <Más-Factor>
<Más-Factor> -> <Operador2> <Factor> <Más-Factor> | e
<Factor> -> (<expresión>) | tk_n <expresión> | tk_ln (<expresión>) | tk_sqrt (<expresión>) | id | literal-Entero | literal-Real | literal-Fecha
<Operador-Relacional> -> < | >= | > | <= | = | !=
<operador1> -> + | -
<operador2> -> * | /

Una vez definidas las reglas de la gramática, el Analizador Sintáctico podrá realizar su función principal: analizar la secuencia de los tokens de una cadena.

2.2.3 Descripción de las clases utilizadas en el análisis sintáctico

Clase: ALexico

Esta clase brinda al analizador sintáctico, cada uno de los tokens que se generan en el análisis léxico.

Clase: ASintáctico

Esta clase le pide al analizador léxico cada uno de los token a medida que va analizando la secuencia de los mismos. El analizador léxico realiza su función en la identificación de token y los va retornando al sintáctico.

El analizador sintáctico toma cada uno de los de token y va analizando sintácticamente que la secuencia en la que aparecen los mismos es correcta, a partir de las reglas de la gramática definidas anteriormente; notificando errores en caso de que el orden de los tokens no sea el correcto. Paralelamente construye el AST comenzando por las hojas del árbol hasta la raíz.

Nodos

Para la construcción del AST, el analizador sintáctico se auxilia de las siguientes clases, a partir de las cuales se pueden crear cada uno de los nodos del árbol.

- AST
- ExpresionAST
- EnteroARealExpresionAST
- ExpresionBinariaAST
- ExpresionUnariaAST
- SimboloAST
- IdValorAST
- ValorEnteroAST
- ValorFechaAST
- ValorRealAST

Las clases anteriores representan las operaciones, las cuales pueden ser de tipo binarias (se realizan entre dos operandos, por ejemplo la suma), de tipo unaria (se realiza a un solo operando, por ejemplo el logaritmo) y la operación que convierte una expresión de entero a real. Además se definieron cuatro clases que representan los valores de los identificadores y las constantes de tipo entero, real y fecha.

Clase: Token

El analizador sintáctico utiliza esta clase para verificar el tipo de token en el análisis.

2.2.4 Detección de errores

En esta fase, la detección de errores ocurre cuando se está analizando el cumplimiento de las reglas de la gramática, específicamente cuando el analizador chequea que el orden de los tokens en la cadena es el correcto. Los mensajes de error que se generan son los siguientes:

- Se esperaba un token de tipo “x”, en vez de uno de tipo “y”.
- Se esperaba un token de tipo “x”.
- No se esperaba un token de tipo “x”.

Cada vez que el Analizador Sintáctico encuentra un error, lo guarda en la lista de errores y continúa analizando. Al final del análisis, se devuelve el AST.

2.3 Patrón Visitor

En el desarrollo de las clases que constituyen los nodos del AST y en la implementación de las fases posteriores: Análisis Semántico y Generación de Código Intermedio, se hizo uso del patrón de diseño Visitor el cual facilita el desarrollo de las funcionalidades que permiten recorrer el árbol de sintaxis abstracta.

El patrón de diseño Visitor (Figura 1) es capaz de modelar distintas operaciones a llevar a cabo sobre una misma estructura de datos, permitiendo definir nuevas operaciones sin modificar la estructura del diseño.

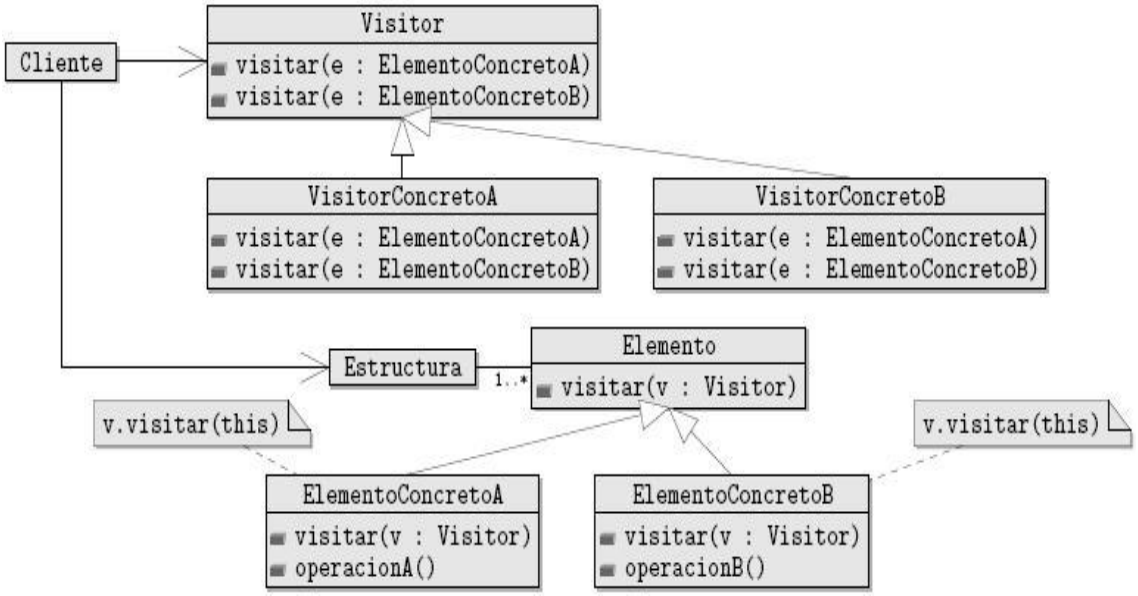


Figura 5: Patrón Visitor.

Las distintas abstracciones del diseño son las que siguen:

Clase Elemento: comúnmente abstracta que define el método polimórfico visitar, recibiendo un visitor como parámetro. Es la clase base de todo nodo del AST.

Elementos Concretos: son cada una de las estructuras concretas a recorrer. Son cada una de las estructuras sintácticas del lenguaje, representadas como nodos del AST.

Estructura: es una colección de elementos. En el caso de un AST, cada nodo poseerá la colección de sus elementos mediante referencias a la clase Elemento y, por tanto, esta abstracción estará dentro de los nodos.

Clase Visitor: comúnmente abstracta, que define una operación de visita por cada elemento concreto de la estructura a recorrer. Cada uno de sus métodos se deberá redefinir (en las clases derivadas) implementando el recorrido específico de cada nodo.

Visitor Concreto: cada uno de los recorridos de la estructura será definido por un Visitor concreto, separando su código en clases distintas. La especificación de los recorridos vendrá dada por la implementación de cada uno de sus métodos de visita, que definirán el modo en el que se recorre cada nodo del AST. Estos podrán acceder a los nodos de la estructura mediante el parámetro recibido.

Siguiendo este patrón, el analizador sintáctico creará el AST con la estructura descrita por Elemento y sus clases derivadas. Después, para cada una de las distintas pasadas, se creará un Visitor concreto y se le pasará el mensaje aceptar al nodo raíz del AST con el Visitor como parámetro.

El Visitor concreto definirá el orden y modo de evaluar un conjunto de atributos del AST. Como el orden de ejecución de cada una de las pasadas es secuencial, los atributos calculados que sean necesarios de una pasada a otra, podrán asignarse a atributos de cada uno de los objetos del AST. Por ejemplo, el tipo de cada una de las expresiones inferido en la fase de análisis semántico, es necesario para llevar a cabo tareas de generación de código. [1]

En el desarrollo del intérprete, las clases Elementos Concretos, constituyen cada uno de los nodos del AST, que heredan de la clase AST; las clases Visitor Concreto, son las clases: Chequeador (implementada para la fase de Análisis Semántico) y la clase Codificador (implementada para la fase de Generación de Código Intermedio). Estas últimas dos clases implementan la interfaz IVisitador. En ellas se define el orden y modo de evaluar los atributos del AST. En las fases posteriores se explica el funcionamiento de cada una de estas clases.

2.4 Fase: Análisis Semántico

El Análisis Semántico en un procesador de lenguaje, es la fase encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. También, de un modo convencional y menos formal, se suele afirmar que la sintaxis de un lenguaje de programación es aquella parte del lenguaje que puede ser descrita

mediante una gramática libre de contexto, mientras que el análisis semántico es la parte de su especificación que no puede ser descrita por la sintaxis. [1]

Durante el análisis semántico se recoge información que resulta de utilidad para fases posteriores. Cuando esta información se almacena en el propio AST, este se denomina AST "decorado", ver la Figura 6. El la figura se representa el AST correspondiente a la operación de asignación (a = b+c), en este caso se almacena la información necesaria para, en un recorrido posterior del AST, poder chequear la compatibilidad de tipos en las operaciones de suma y asignación entre las variables b, c y a respectivamente.

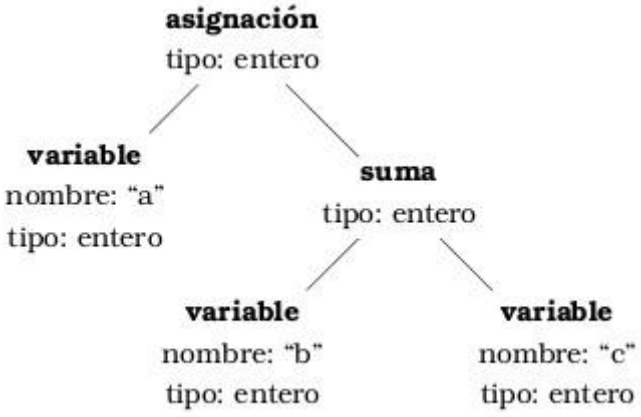


Figura 6: AST decorado.

2.4.1 Decoración del AST

La decoración del AST en el intérprete, se realizó asignando a cada nodo o expresión (Ln, SQRT, suma, resta,…, identificadores, constantes), el atributo: tipo de dato, que indica si la expresión es real, entera o booleana.

2.4.2 Comprobación de tipos

Sin duda, este tipo de comprobaciones es el más exhaustivo y amplio en la fase de análisis semántico. De un modo somero, el analizador semántico deberá llevar a cabo dos tareas relacionadas con los tipos: [1]

1. Comprobar las operaciones que se pueden aplicar a cada construcción del lenguaje. Dado un elemento del lenguaje, su tipo identifica las operaciones que sobre él se pueden aplicar. Por ejemplo, en el lenguaje Java el operador de producto no es aplicable a una referencia o a un objeto, mientras el operador punto si es válido.

Para realizar esta tarea se definieron las siguientes premisas:

- Las operaciones de tipo: suma, resta, multiplicación, división, logaritmo (Ln) y raíz cuadrada, son aplicables a las expresiones de tipo entero o real. La operación de resta es además aplicable a las expresiones de tipo texto, lo cual ocurre en la concatenación.
- Las operaciones de comparación, son aplicables a las expresiones de tipo entero, real y fecha.
- Las operaciones lógicas, son aplicables a las expresiones de tipo booleanas.

2. Inferir el tipo de cada construcción del lenguaje. Para poder implementar la comprobación anterior, es necesario conocer el tipo de toda construcción sintácticamente válida del lenguaje. Así, el analizador semántico deberá aplicar las distintas reglas de inferencia de tipos descritas en la especificación del lenguaje de programación, para conocer el tipo de cada construcción del lenguaje.

Para inferir el tipo de una expresión, se tuvo en cuenta que:

- El resultado de las operaciones de suma, resta, división y multiplicación, entre expresiones de tipo entero, es atribuible a expresiones de este mismo tipo.
- El resultado de las operaciones de suma, resta, división y multiplicación, entre expresiones de tipo real, o enteras y reales, es atribuible a expresiones de tipo real.
- El resultado de las operaciones de logaritmo y raíz cuadrada, de expresiones (sin importar el tipo), es atribuible a expresiones de tipo real.
- El resultado de las operaciones de comparación, es atribuible a expresiones booleanas.
- El resultado de operaciones lógicas, es atribuible a expresiones booleanas.
- El resultado de la resta de dos textos, es decir, la concatenación de los textos, se atribuye a expresiones de tipo texto.

2.4.3 Descripción de las clases utilizadas en el análisis semántico

Clase: Chequeador

Esta clase es la encargada de recorrer el árbol de sintaxis abstracta y comprobar los tipos. Esto lo realiza preguntando a cada nodo del árbol y a sus hijos, su tipo. De esta manera comprueba, de forma ascendente, que las operaciones se realicen entre los tipos correctos y va infiriendo el tipo de cada expresión a partir del resultado de cada operación. En caso de que las expresiones sean los propios identificadores o constantes, es decir, las hojas del árbol, se obtiene el tipo de dato de la tabla de símbolos.

Clase: TSimbolos

Brinda a la clase Chequeador, el tipo de dato de los identificadores y las constantes.

2.4.4 Detección de errores

En esta fase, la identificación de errores ocurre mientras se recorre el árbol y se realiza la comprobación de tipos. Se analiza si las operaciones que se realizan son correctas en dependencia del tipo de los datos de los operandos, teniendo en cuenta además si las variables han sido declaradas e inicializadas. Los mensajes de error que se muestran son los siguientes:

- "El operador “x” no se puede aplicar sobre el tipo “y”".
- "El operador “y” no se puede aplicar sobre tipos diferentes".
- "La variable “x” no ha sido declarada".
- "La variable “x” no ha sido inicializada".

Cuando se identifica un error, este se guarda en la lista de errores. Al final del análisis se devuelve el AST decorado.

2.5 Fase: Generación de Código Intermedio

Hasta el momento, la cadena entrada por el usuario, ha sido analizada léxica, sintáctica y semánticamente. En caso de que no se hayan detectado errores en estas fases, se procede a generar el Código Intermedio (también llamado Forma Intermedia), a partir del AST decorado.

En la presente investigación se utilizó la representación de código intermedio orientada a pila denominada Notación Polaca. Esta notación tiene la característica de que en la cadena polaca que se va generando a partir del AST, los operandos se guardan primero que los operadores.

2.5.1 Descripción de las clases utilizadas en la generación de código

Clase: EntidadEjecutable

Para cada tipo de operación se creó una clase entidad ejecutable que recibe como parámetro el contexto de ejecución (que se explicará posteriormente) y permite ejecutar cada uno de los operadores. Además se creó una entidad genérica “EjecutableValor” y una para cada tipo de dato que heredan de esta genérica, estas clases permiten gestionar el valor de los identificadores y las constantes.

Clase: Codificador

Esta clase recorre el AST en post-orden. Cuando se encuentra en cada nodo, añade elementos a la cadena polaca que son representados por las entidades ejecutables.

Para las constantes de tipo entero, real y fecha, guarda la dirección en memoria en la tabla de símbolos. De esta forma se va generando el código intermedio en la cadena polaca.

Clase: TSimbolo

Esta clase permite gestionar la dirección en memoria de los símbolos.

Clase: TipoToken

Esta clase permite conocer el tipo de cada token.

2.6 Fase: Evaluación

La evaluación es la fase en la que se ejecuta la cadena polaca generada en la fase anterior.

El evaluador del intérprete fue desarrollado atendiendo a las características principales de un intérprete para la forma interna basada en la Notación Polaca:

- Utiliza una pila de ejecución.
- Accede a la Notación Polaca.
- Utiliza la semántica definida a los operadores de la Notación Polaca.
- Consta de un ciclo principal, en cada iteración del ciclo se ejecuta uno de los operadores de la Notación Polaca. [1]

2.6.1 Descripción de las clases utilizadas en la evaluación

Clase: Memoria

Posee una Lista con los valores de los identificadores y las constantes.

Clase: Contexto

La clase Contexto posee la cadena polaca que contiene las entidades ejecutables, crea una pila y tiene una instancia de la clase memoria.

La pila se utiliza mientras se recorre el código intermedio. Esta va guardando los elementos de la cadena polaca; cuando el elemento sea un operador, se ejecuta la operación tomando los valores guardados en la pila y se guarda el resultado en la misma. Así sucesivamente, la pila siempre almacena los valores que se van a utilizar para realizar las operaciones.

Clase: Evaluador

Esta clase posee un objeto de la clase Contexto. Contiene un ciclo que permite recorrer la cadena polaca y en cada iteración ejecuta un operador de la cadena, pasando como parámetro el contexto a la función Ejecutar de cada entidad ejecutable.

2.6.2 Detección de errores

A la vez que se van ejecutando las entidades de la cadena polaca, se genera un mensaje en el caso de que ocurra alguno de los siguientes errores:

- División por cero.
- Logaritmo de un número menor o igual que cero.
- Raíz cuadrada de un número negativo.

2.7 Funcionamiento general del intérprete

El funcionamiento de las fases del intérprete es guiado por la clase Intérprete, la cual recibe una cadena de caracteres a interpretar. A continuación se presentan cada uno de los métodos de esta clase que permiten su funcionamiento.

Primeramente, esta clase se auxilia de la clase CodigoFuente para gestionar la cadena de caracteres recibida.

Método AdicionarEntradas(ArrayList<Entrada> entradas)

Este método recibe una lista de entradas, las cuales constituyen los identificadores. La clase Entrada contiene los atributos: tipo de dato, lexema y valor. La función del método es recorrer la lista de entradas y llamar al método AdicionarEntrada, pasando como parámetro cada uno de los elementos de la lista.

Método AdicionarEntrada(Entrada entrada)

Este método adiciona a la tabla de símbolos cada identificador y reserva un espacio en memoria para su valor, auxiliándose de la clase Entrada.

Método Compilar()

Este método es el encargado de invocar los analizadores sintáctico y semántico, y en caso de no encontrar errores, invoca al Codificador para obtener el código intermedio.

Primeramente invoca al método Parse() del analizador sintáctico (ASintactico) para realizar el análisis, y devuelve el árbol de sintaxis abstracta.

Luego este árbol se le pasa como parámetro al método Analizar del Chequeador y este a su vez devuelve el AST decorado.

Finalmente el método Compilar() analiza si la lista de errores generados en los análisis léxico, sintáctico y semántico está vacía. En caso de que esté vacía, llama al Codificador, al cual se le pasa el árbol decorado; este genera el código intermedio y retorna la cadena polaca con las entidades ejecutables.

Método Interpretar()

La primera función de este método es llamar al método Compilar(), y recibe como resultado el código intermedio generado en la cadena polaca. Luego este código es pasado al contexto a través de un objeto de esta clase. Posteriormente se invoca a la clase Evaluador, pasándole el contexto como parámetro, y por último se llama al método Interpretar del evaluador, el cual finalmente interpreta el código intermedio haciendo uso del contexto.

Luego de la interpretación, el método retorna como resultado, el elemento que quedó en la pila utilizada en el contexto. En esta entidad se encuentra el resultado de la ejecución de la regla de validación. Este resultado representa:

- para el caso de la regla Rangos, si la variable a la cual se le asignó la regla se encuentra en el rango especificado.
- para las demás reglas, el resultado de la fórmula o de la concatenación.

Método Salida()

Este método crea un objeto de tipo Salida. Esta clase contiene los atributos: tipo y valor, donde se guarda el resultado final de la interpretación. Esta clase sirve como interfaz entre el Intérprete y el sistema alasClínicas; este último recibirá como resultado de la interpretación un objeto de tipo Salida que luego podrá analizar y utilizar.

Método getErrores()

Este método retorna la lista de errores encontrados en el análisis léxico, sintáctico y semántico.

2.7.1 El interprete en el sistema alasClinicas

El intérprete es invocado desde el modulo Validación del sistema alasClinicas, específicamente en la clase Validación donde se gestionan todas las reglas. El mismo se invoca cuando se desea declarar o ejecutar una regla de tipo Rangos, Fórmula o Concatenación.

En la declaración, solo se ejecutan las fases de análisis léxico, sintáctico y semántico, analizando la existencia de errores en la especificación de las reglas. Se devuelve como resultado la lista de errores encontrados.

La ejecución de las reglas ocurre cuando se entran los datos en las hojas CRD en el sistema. Se ejecutan todas las fases del intérprete devolviendo como resultado final, para el caso de las reglas de tipo Rangos: un valor booleano que indica si el valor

entrado se encuentra en el rango especificado, y para el caso de las demás reglas: un valor entero o real que indica el resultado de la fórmula o un valor de tipo texto que indica el resultado de la concatenación.

Conclusiones

En el presente capítulo se mostró un resumen del desarrollo del intérprete en cada una de sus fases, donde se evidenciaron las características del mismo así como las clases que fueron implementadas y los posibles tipos de error en los que se puede incurrir durante la declaración y ejecución de las reglas.

La estructura modular del intérprete, que cuenta en cada fase con una entrada y una salida, permite tener el código organizado de forma óptima, garantizando que para la realización de cambios se modifique solo la fase del intérprete que lo requiera.

La detección de errores en las fases de análisis léxico, sintáctico y semántico, contribuye a que las reglas de validación sean declaradas correctamente.

El patrón Visitor utilizado en las fases de análisis semántico y la generación de código intermedio, permite definir las diferentes operaciones de los analizadores sin modificar la estructura de diseño.

La programación orienta a objetos, permitió la organización del código de forma que pueda ser reutilizado en futuras modificaciones del intérprete. Tal es el caso de la fase Evaluación, en la cual, a la hora de añadir o modificar alguna regla de validación, bastaría con añadir o modificar entidades ejecutables, sin modificar el contexto de ejecución.

CAPÍTULO 3: EVALUACIÓN DEL INTÉRPRETE

Introducción

En el presente capítulo se aplica el método Delphi para evaluar a través de la opinión de expertos, el intérprete desarrollado. Se detallan cada una de las etapas por las que se transita en el empleo del método, y los resultados finalmente alcanzados.

Para la ejecución del método se desarrollaron las siguientes fases:

1. Formulación del objetivo que se desea alcanzar con la aplicación del método.
2. Elección de los expertos que evaluarán el intérprete.
3. Realización de pruebas al intérprete por los expertos.
4. Elaboración y lanzamiento de los cuestionarios a través de los cuales se obtienen las opiniones de los expertos.
5. Explotación de los resultados.

3.1 Formulación del objetivo a alcanzar

En esta fase es importante determinar claramente el objetivo que se persigue con la utilización del método, de manera que las siguientes etapas giren en torno a este objetivo y se obtengan resultados correctos.

El objetivo de la utilización del método en la investigación es: analizar a través de expertos, la medida en la que el intérprete desarrollado contribuirá a la validación de los datos de los EC y por consiguiente a la detección de errores, y de esta manera, validar la hipótesis planteada en la investigación.

3.2 Elección de expertos

Esta etapa es importante en cuanto a que el término "experto" es ambiguo. Con independencia de sus títulos, su función o su nivel jerárquico, el experto debe ser elegido por su capacidad de encarar el futuro y sus conocimientos sobre el tema consultado. La falta de independencia de los expertos puede constituir un inconveniente; por esta razón los expertos generalmente son aislados y sus opiniones son recogidas de forma anónima; así pues se logra obtener la opinión real de cada experto y no la opinión más o menos falseada por un proceso de grupo (se trata de eliminar el efecto de los líderes).

Aunque no hay forma de determinar el número óptimo de expertos para participar en una encuesta Delphi, estudios realizados por investigadores de la Rand Corporation[4] señalan que si bien parece necesario un mínimo de siete expertos, habida cuenta que el error disminuye notablemente por cada experto añadido hasta llegar a los siete expertos, no es aconsejable recurrir a más de 30 expertos, pues la mejora en la previsión es muy pequeña y normalmente el incremento en coste y trabajo de investigación no compensa la mejora. [6]

Se infiere, como requisito básico para la selección de un experto, que éste tenga experiencia en el tema a consultar, dado por sus años de trabajo (praxis), y que puedan ser complementados con: conocimientos teóricos adquiridos a través de las distintas formas de superación, y grado académico o científico alcanzado en relación al tema, entre otros. [27]

Se seleccionó un total de 12 personas del Centro de Inmunología Molecular, con al menos 3 años de experiencia en la gestión de EC, generalmente con el rol Gerente de Datos, que desempeñan sus funciones en el diseño de los Cuadernos de Recogida de Datos para los ensayos clínicos y por ende serán los usuarios finales del sistema, encargados de validar los datos que se recogen en las variables de dichos cuadernos.

Para la selección dentro de este grupo de personas, de los expertos que formaron parte del panel que participó en la validación del intérprete, se tuvo en cuenta el coeficiente de competencia K, dado por el coeficiente de conocimiento Kc y el de argumentación Ka; utilizando la fórmula: K=½ (Kc + Ka).

Si 0,8 < K < 1,0 el coeficiente de competencia es alto.

Si 0,5 < K < 0,8 el coeficiente de competencia es medio.

Si K < 0,5 el coeficiente de competencia es bajo.

Para la determinación de Kc y Ka, se realizó una encuesta a los posibles expertos (ver Anexo 1) donde se analizaron los siguientes criterios:

- **Para el análisis del Kc:**
  Establecer en una escala de 1 a 10 puntos el grado de conocimientos acerca del tema: Validación de datos de ensayos clínicos.
  Luego este valor se multiplica por 0.1 y se obtiene el resultado del coeficiente de conocimientos.

[4] Rand Corporation (Research ANd Development): Compañía nortemericana dedicada a investigaciones y desarrollos científicos.

- **Para el análisis del Ka:**

  Analizar el grado de influencia, en los expertos, de cada una de las siguientes fuentes:

Fuentes de argumentación	**Grado de influencia de cada una de las fuentes en su conocimiento**		
	Alto	**Medio**	**Bajo**
Análisis teóricos realizados por usted acerca del tema.	0.4	0.3	0.2
Su experiencia obtenida.	0.4	0.3	0.2
Trabajos de autores nacionales.	0.05	0.04	0.03
Trabajos de autores extranjeros.	0.05	0.04	0.03
Su propio conocimiento del tema en el extranjero.	0.05	0.04	0.03
Su intuición.	0.05	0.04	0.03

  La suma de los valores obtenidos, representa el coeficiente de argumentación.

Una vez calculado el coeficiente de competencia, a partir de la fórmula correspondiente, se escogieron los 12 expertos que obtuvieron la calificación del coeficiente entre 0.5 y 1.0, es decir los expertos de coeficiente alto y medio, quedando todos los expertos seleccionados para la aplicación del cuestionario.

3.3 Realización de pruebas al intérprete por los expertos

Luego de la selección de los expertos, se impartió un curso de capacitación a los mismos con el objetivo de que aprendieran a trabajar con la herramienta.

Posteriormente se creó en el sistema alasClínicas el estudio: Evaluación de la eficacia y seguridad de la vacuna NGlicolilGM3/VSSP en pacientes con cáncer de mama metastásico, donde se diseñaron 27 Cuadernos de Recogida de Datos. Luego se definieron por los expertos, 124 reglas de validación para las variables de los cuadernos, las cuales fueron registradas correctamente en el sistema. En la definición de las reglas, se tuvieron en cuenta los tres tipos de estas, que existiera dependencia entre las variables y que estuvieran presentes los cuatro tipos de datos (entero, real, fecha y texto). Además, los expertos probaron algunas reglas con errores para verificar el funcionamiento del intérprete en la identificación de estos.

Una vez definidas las reglas, se introdujeron los datos de 24 pacientes en el sistema y se verificó el funcionamiento del intérprete en la detección de errores en los datos, teniendo en cuenta lo especificado en cada una de las reglas.

Los expertos tuvieron un periodo de un mes para trabajar con la herramienta, familiarizarse con ella y probar otros casos reales.

3.4 Elaboración y lanzamiento de los cuestionarios

3.4.1 Elaboración del cuestionario

Los cuestionarios se elaboran de manera que faciliten, en la medida en que la investigación lo permita, la respuesta por parte de los consultados.

En varias ocasiones, es conveniente recurrir a respuestas categorizadas (Si/No; Mucho/Medio/Poco; Muy de acuerdo/ De acuerdo/ Indiferente/ En desacuerdo/Muy en desacuerdo) y después se tratan las respuestas en términos porcentuales tratando de ubicar a la mayoría de los consultados en una categoría. [6]

Para la evaluación del intérprete en la validación de datos de EC, se verifica que el mismo cumpla con los siguientes requisitos:

R1: El lenguaje definido para el establecimiento de las reglas de validación es de fácil comprensión y abarca todas las posibles reglas a definir, de tipo Rangos, Concatenación y Fórmula.

R2: La elaboración de las reglas en el sistema es de fácil manejo para el usuario.

R3: El intérprete permite detectar errores en la elaboración de las reglas.

R4: El intérprete permite en la declaración de las reglas, establecer dependencias entre las variables.

R5: El intérprete ejecuta todas las reglas, correctamente definidas, por el Gerente de Datos.

R6: El intérprete permite detectar errores en los datos de los EC entrados en los CRD electrónicos, en el sistema alasClínicas.

R7: El intérprete contribuye a la validación de los datos de los EC, permitiendo obtener resultados más certeros en los estudios de investigación clínica que se realizan en el CIM.

Teniendo en cuenta estos requisitos se desarrolló el cuestionario con 7 preguntas, cada una de las cuales interroga respectivamente, el cumplimiento de cada uno de los requisitos anteriores (ver cuestionario en el Anexo 2).

El resultado de las preguntas del cuestionario se clasifica en: Muy de acuerdo, De acuerdo, Indiferente, En desacuerdo y Muy en desacuerdo. Sin embargo para ayudar a los expertos a facilitar su respuesta, se le pidió que evaluaran el cumplimiento de los requisitos en una escala de 1 a 5 puntos, luego cada valor se hizo coincidir respectivamente con la clasificación antes mencionada (5=Muy de acuerdo, 4=De acuerdo, 3=Indiferente, 2=En desacuerdo, 1=Muy en desacuerdo).

3.4.2 Lanzamiento del cuestionario

Al final de la realización de las pruebas al intérprete, se le presentó a cada experto el cuestionario para que respondiera las preguntas de forma anónima con sus criterios particulares. A partir de aquí se obtuvieron los siguientes resultados:

Resumen de resultados

Requisitos evaluados	Criterios				
	C1	C2	C3	C4	C5
R1	10	2			
R2	11	1			
R3	8	4			
R4	12				
R5	12				
R6	8	4			
R7	12				

C1, C2, C3, C4 y C5, responden a los criterios: Muy de acuerdo, De acuerdo, Indiferente, En desacuerdo y Muy en desacuerdo, respectivamente.

Los valores de las celdas representan la cantidad de expertos que evaluaron con cada criterio a cada uno de los requisitos.

Luego de la encuesta, algunos expertos recomendaron, que para futuras versiones del intérprete, el mismo permita que en las reglas de tipo Rangos, se puedan definir rangos más complejos entre las variables de tipo fecha; por ejemplo establecer que una fecha sea tres meses anterior, o cuatro años posterior, a otra fecha definida.

3.4.3 Determinación de la concordancia de los expertos

Para que el resultado final de la evaluación realizada tenga una mayor validez, es necesario que exista un adecuado nivel de concordancia entre los integrantes del panel de expertos. Para ello, se procede a calcular el grado de coincidencia de las valoraciones realizadas.

Primeramente se presenta una tabla donde se muestra la evaluación dada por cada experto en cada uno de los criterios evaluados:

Experto/Criterio	R1	R2	R3	R4	R5	R6	R7
Experto 1	5	4	5	5	5	5	5
Experto 2	5	5	5	5	5	5	5
Experto 3	5	5	4	5	5	4	5
Experto 4	4	5	5	5	5	5	5
Experto 5	5	5	4	5	5	5	5
Experto 6	5	5	5	5	5	4	5
Experto 7	5	5	5	5	5	5	5
Experto 8	5	5	5	5	5	4	5
Experto 9	5	5	4	5	5	5	5
Experto 10	4	5	5	5	5	5	5
Experto 11	5	5	5	5	5	5	5
Experto 12	5	5	4	5	5	4	5
Total (S_j)	**58**	**59**	**56**	**60**	**60**	**56**	**60**

Luego se calcula el coeficiente de concordancia de Kendall a partir de la fórmula:

$$W = \frac{12S}{k^2(N^3 - N)}$$

- Donde S es la suma de los cuadrados de las desviaciones dada por la fórmula:

  $$S = \sum_{j=1}^{n} (S_j - \overline{S})^2$$

- $\overline{S}$ constituye la media calculada por la fórmula: $\overline{S} = \dfrac{\sum_{j=i}^{n} S_j}{N}$

- S_j es el total calculado a partir de las evaluaciones de los expertos, para cada criterio (58, 59, 56, 60, 60, 56, 60).

- N es el número de preguntas realizadas a los expertos, que en este caso es el número de requisitos analizados (7).
- k es el número de expertos (12).

Según los cálculos realizados, el coeficiente de Kendall obtiene el valor 0,005 (W=0,005).

Luego se procede a calcular Chi cuadrado real, el cual se compara con el de las tablas estadísticas de Siegel para obtener el resultado final.

Si $\chi^2_{real} < \chi^2_{(\alpha, N-1)}$ entonces existe concordancia en la evaluación de los expertos.

$\chi^2_{real} = K (N – 1)W = 0,36$

Tomando como nivel de significación: $\alpha = 0.01$, el resultado de Chi cuadrado obtenido de las tablas, es: $\chi^2_{(0,01, 6)} = 0, 872$

Luego se demuestra que $\chi^2_{real} < \chi^2_{(\alpha, N-1)}$ (0,36 < 0,872), por tanto existe concordancia entre los expertos.

3.5 Explotación de resultados

Para realizar el análisis de los resultados, se hizo uso de una herramienta programada en Microsoft Excel 2003, la cual facilita el cálculo del resultado de la evaluación de los expertos en cada pregunta.

A partir de los resultados, primeramente se procedió a calcular las frecuencias absolutas acumuladas:

Requisitos evaluados	**Criterios**				
	C1	**C2**	**C3**	**C4**	**C5**
R1	10	12			
R2	11	12			
R3	8	12			
R4	12				
R5	12				
R6	8	12			
R7	12				

Luego se calcularon las frecuencias relativas acumuladas:

Requisitos evaluados	Criterios				
	C1	C2	C3	C4	C5
R1	0,83333333	0,9999			
R2	0,91666667	0,9999			
R3	0,66666667	0,9999			
R4	0,9999				
R5	0,9999				
R6	0,66666667	0,9999			
R7	0,9999				

Con estos resultados, se desarrolló una tabla denominada Puntos de Corte, la cual muestra el cálculo del resultado final de la evaluación de los expertos en cada pregunta. Teniendo en cuenta que la evaluación puede ser clasificada en cinco categorías, solo se necesitan cuatro puntos de corte, por tanto se eliminó la columna C5.

Requisitos evaluados							N=0.84	
	C1	C2	C3	C4	Suma	P	N-P	Evaluación
R1	0,97	3,72			4,69	2,34	-1,51	**Muy de acuerdo**
R2	1,38	3,72			5,10	2,55	-1,72	**Muy de acuerdo**
R3	0,43	3,72			4,15	2,07	-1,24	**Muy de acuerdo**
R4	3,72				3,72	3,72	-2,88	**Muy de acuerdo**
R5	3,72				3,72	3,72	-2,88	**Muy de acuerdo**
R6	0,43	3,72			4,15	2,07	-1,24	**Muy de acuerdo**
R7	3,72				3,72	3,72	-2,88	**Muy de acuerdo**
Suma:	**14,37**	**14,88**			**29,24**			
P. de corte:	**2,05**	**3,72**						

Los valores de las celdas determinadas por la intersección de los requisitos y los criterios fueron calculados a partir de la fórmula DISTR.NORM.ESTAND.INV, que brinda el Excel. Esta fórmula calcula el inverso de la distribución acumulativa normal estándar tomando como parámetro las probabilidades dadas por las frecuencias

relativas acumuladas, devuelve el resultado de k que hace que la probabilidad de que x < k, sea p, es decir: P(x<k)=p.

La P calculada en cada fila, representa el promedio dado por la Suma entre la cantidad de criterios que evalúan cada requisito.

La N fue calculada dividiendo la sumatoria de todas las sumas, entre la multiplicación de la cantidad de preguntas por la cantidad de criterios: N=29,24/7*5=0,84.

Los puntos de corte en cada criterio o columna, fueron calculados dividiendo la Suma entre la cantidad de preguntas (requisitos) respondidas con este criterio.

Los puntos de corte fueron: 2,05 y 3,72. Solo se calcularon estos dos puntos de corte debido a que las respuestas de los expertos estuvieron concentradas solo en dos criterios.

Resultado de la evaluación:

Teniendo en cuenta los dos puntos de corte encontrados, se procede a analizar el valor de N-P.

Si N-P< 2,05: la evaluación dada por los expertos es: Muy de acuerdo

Si 2,05<N-P<3,72: la evaluación es: De acuerdo

Al evaluar N-P en estos intervalos, se determina para todas las preguntas, que los expertos estuvieron muy de acuerdo con el cumplimiento de los requisitos analizados, validando de esta forma el funcionamiento del intérprete como herramienta para la validación de datos de ensayos clínicos. Observando que el resultado obtenido en la evaluación es el mejor y que existe un nivel adecuado de concordancia en la opinión de los expertos, no se procedió a realizar una segunda ronda de cuestionarios.

Conclusiones

En el presente capítulo se aplicó el Método Delphi para evaluar al intérprete como herramienta para la validación de datos de ensayos clínicos, a partir de la opinión de expertos del Centro de Inmunología Molecular. Para la evaluación se tuvo en cuenta un grupo de requisitos que el intérprete debe cumplir.

Luego de realizar un conjunto de pruebas a la herramienta, los expertos confirmaron que esta cumple con todos los requisitos y por tanto se determina que la misma contribuye a la validación de datos de ensayos clínicos.

Expertos proponen que las reglas de tipo Rangos, permitan establecer otros rangos entre las variables de tipo fecha que presenten una mayor complejidad.

CONCLUSIONES GENERALES

La investigación presentó el desarrollo de un intérprete para la validación de los datos de los ensayos clínicos creados en el sistema alasClínicas. El intérprete fue desarrollado en cada una de sus fases, y será utilizado en la declaración y ejecución de las reglas en el sistema.

Las fases de análisis léxico, sintáctico y semántico del intérprete verifican la declaración correcta de las reglas de validación.

El intérprete con todas sus fases, permite la ejecución de las reglas en el sistema alasClínicas, mostrando los posibles errores que pueden existir en cada uno de los datos recogidos en un ensayo.

La estructura modular del intérprete, que divide el código en varias fases, favorece la realización de futuros cambios, así como la reutilización del código.

Finalmente la herramienta fue evaluada por un grupo de expertos del Centro de Inmunología Molecular, los cuales alegaron que la misma elevará la validez de los datos que se recogen en los ensayos contribuyendo al desarrollo de estudios de investigación clínica más certeros. Algunos proponen que las reglas de tipo Rangos, permitan establecer rangos más complejos entre las variables de tipo fecha.

RECOMENDACIONES

- Se recomienda que para futuras versiones del intérprete, las reglas de tipo Rangos, permitan establecer rangos más complejos para los datos de tipo fecha.
- Guardar en la base de datos del sistema, el código intermedio que se genera después de analizar léxica, sintáctica y semánticamente la declaración de las reglas; de manera que en la ejecución de estas, solo sea necesario ejecutar el código intermedio.
- Utilizar el intérprete en otros sistemas que necesiten la validación de datos, donde se puedan definir reglas de validación.

REFERENCIAS BIBLIOGRÁFICAS

1. Acevedo, L. *Programación IV: Tema Compilación.* Manual de apoyo a la docencia. Universidad de las Ciencias Informáticas. Departamento Docente Central de Técnicas de Programación. 2011. [citado: 2011 Ene 28]
2. Aho, A; Lam, Mónica; Sethi, R; Ullman, J. *Compilers: principles, techniques, and tools.* 2º ed. Boston, San francisco, New York: 2007. [citado: 2011 Ene 28]
3. Akaza Research Open Clinica. *Open Source for Clinical Research*. 2007. [Citado: 2011 Abr 11]. Disponible en: http://www.openclinica.org/section.php?sid=1
4. Alvaredo, S; Delgado, A; Aira, S; Rubio, A; Herrero, J; Mozo, LM. *Lenguajes de alto nivel*. 2007. [Citado: 2011 Feb 5].
5. *Análisis sintáctico descendente y ascendente*. 2010. [citado: 2011 Sep 18]. Disponible en: http://wi7max.wordpress.com/2010/12/14/analisis-sintactico-descendente-y-ascendente/
6. Astigarraga, Eneko. *El Método Delphi*. Universidad de Deusto, Facultad de CC.EE. y Empresariales. 2008. [citado: 2011 Mar 24]. Disponible en: http://www.echalemojo.org/uploadsarchivos/metodo_delphi.pdf
7. Barr, Sherold. *DataLabs Introduces Single Data Management System to Unify Paper Data Entry and Electronic Data Capture*. [Citado: 2011 Abr 10]. Disponible en: http://www.datalabs.com/
8. *Bayer AG*. [Citado: 2011 Abr 10]. Disponible en: http://www.bayer.com
9. *Bayer Healthcare estandariza la solución de gestión de datos clínicos Medidata Solutions RAVE*. [citado: 2011 Abr 10]. Disponible en: http://www.prnewswire.co.uk
10. CIMAher. *CIMAher: Centro de Inmunología Molecular*. 2009. [Citado 2011 Feb 5]. Disponible en: http://www.cimaher.com/index.php?action=cim
11. Clinical Conductor Site. *CTMS*. [Citado: 2011 Abr 10]. Disponible en: http://www.bio-optronics.com/CTMSClinicalTrialsManagementSoftware/
12. *Clinical Force*. [Citado: 2011 Feb 7]. Disponible en: http://www.perceptive.com/ctms/

13. Cortés Fuentes, Carlos Arturo. *Gramáticas independientes del contexto*. 2008. [citado: 2011 Mar 18]. Disponible en: http://cnx.org/content/m16320/latest/

14. Cortez, Augusto. *Análisis comparativo entre un analizador sintáctico LL y un analizador sintáctico LR para un lenguaje formal*. Investig. Sist. Inform. RISI 2 (2). Pág. 60-68. 2005. [citado: 2011 Sep 18]. Disponible en: http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/risi/n2_2005/a09.pdf

15. *Data Entry Services India*. [Citado: 2011 Jun 2]. Disponible en: http://www.dataentryservices.co.in/paper_data_entry.htm

16. Gálvez. *JavaCUP-Análisis Léxico: JLEX*. [citado 2011 Feb 5]. Disponible en: http://www.lcc.uma.es/~galvez/theme/cup/anlexcup/jlex.html

17. Gálvez, S y Mora, MA. *Java a tope. Compiladores. Traductores y compiladores con Lex/Yacc, JFlex/Cup y JavaCC*. Universidad de Málaga: 2005. [citado: 2011 Ene 28]. Disponible en: http://www.lcc.uma.es/~galvez/ftp/libros/Compiladores.pdf

18. García Cota, Enrique José. *Guía práctica de ANTLR*. 2003. [citado: 2011 Sep 28]. Disponible en: http://www.lsi.us.es/~troyano/documentos/guia.pdf

19. González. *Tutorial JFlex*. 2010 [citado 2011 Feb 5]. Disponible en: http://uempl.wordpress.com/2010/05/04/tutorial-jflex-2/

20. Gonzalo García, Martha. *Clasificación de lenguajes formales de Chomsky*. 2001. [citado: 2011 Mar 18]. Disponible en: http://docs.google.com/viewer?a=v&q=cache:y4kCtcB6e8gJ:tux.uis.edu.co/lenguajes/doc/chomsky.doc+Clasificaci%C3%B3n+de+lenguajes+formales+de+Chomsky.&hl=es&gl=cu&pid=bl&srcid=ADGEESiKPlpwJdvMaQCXsgdx0Xyl5moMVkq2Eew9GOQ5nyfuSA71hg29QOW8VrTWMblBj70_0s5Wi5D7FuNHs_y1QIl9YA8wf0gwGnAT6yclTr1IncJsUsNAemQmtbYa0eClvzbuM8JR&sig=AHIEtbTPni1x518IeEiijUrOq3nfsWi02w

21. *Kbeedocs*. [Citado: 2011 Abr 10]. Disponible en http://www.kbeedocs.com 11

22. Kyung-hee, Kelly Moon. *Techniques for Designing Case Report Forms in Clinical Trials*. ScianNews Vol. 9, No. 1 Fall 2006. [citado: 2011 Feb 7].

23. Manrique, M. *Curso Teoría de compiladores. Analizador Léxico Lex*. [citado: 2011 Ene 28] Disponible en: http://biblioteca.uns.edu.pe/saladocentes/archivoz/curzoz/Sesion_5.pdf

24. Maythe. *Java Cup - Documentos*. 2010 [citado 2011 Feb 5]. Disponible en: http://www.buenastareas.com/ensayos/Java-Cup/871197.html

25. *Medidata Solutions.* [Citado: 2011 Abr 10]. Disponible en: http://www.mdsol.com

26. MedlinePlus. *Ensayos clínicos: MedlinePlus en español*. 2009. [Citado: 2011 Feb 5]. Disponible en: http://www.nlm.nih.gov/medlineplus/spanish/clinicaltrials.html

27. Moráguez Iglesias, Arabel. *Otros conceptos de economía*. 2006. [citado: 2011 Sep 18]. Disponible en: http://www.gestiopolis.com/canales6/eco/metodo-delphi-estadistica-de-investigacion-cientifica.htm

28. *Open Clínica: La nueva generación en captura electrónica de datos y ensayos clínicos.* [Citado: 2011 Abr 10]. Disponible en: http://www.openclinica.es/

29. Pinto, M. *Intérpretes*. 2006. [citado: 2011 Ene 20].

30. *PRA International*. [citado: 2011 Abr 10]. Disponible en: http://www.prainternational.com

31. Remache, Katherine. *Compiladores-traductores-lenguajes-de-programación*. 2008. [citado: 2011 Ene 20]. Disponible en: http://blogs.utpl.edu.ec/metodologiadeprogramacion/files/2009/05/compiladores-traductor-lenguajes-de-programacion.pdf

32. s4research. *¿Qué es OpenClinica?* [citado: 2011 Abr 11]. Disponible en: http://www.s4research.es/que-es-openclinica-solutions-cro-barcelona-espana-open-source-ensayos-clinicos

33. Vélez Reyes, Javier. *Procesadores de lenguaje: Análisis Sintáctico*. 2010. [citado: 2011 Ene 8]. Disponible en: http://www.lsi.uned.es/procleng/apuntes/2010-2011/PDL.07.Tema4.AnalisisSintacticoDescendente.pdf

34. Veritosoto. *ANTLR.* 2011. [citado: 2011 Sep 9] .Disponible en: http://veritosoto.wordpress.com/2011/02/03/antlr/

BIBLIOGRAFÍA

1. Acevedo, L. *Programación IV. Tema Compilación.* Manual de apoyo a la docencia. Universidad de las Ciencias Informáticas. Departamento Docente Central de Técnicas de Programación. 2011.
2. Aho, A; Lam, Mónica; Sethi, R; Ullman, J. *Compilers: principles, techniques, and tools.* 2º ed. Boston, San francisco, New York. 2007.
3. Akaza Research Open Clinica. *Open Source for Clinical Research.* 2007. Disponible en: http://www.openclinica.org/section.php?sid=1
4. Alvaredo, S; Delgado, A; Aira, S; Rubio, A; Herrero, J; Mozo, LM. *Lenguajes de alto nivel*. 2007.
5. *Análisis sintáctico descendente y ascendente.* 2010. Disponible en: http://wi7max.wordpress.com/2010/12/14/analisis-sintactico-descendente-y-ascendente/
6. Astigarraga, Eneko*. El Método Delphi..* Universidad de Deusto, Facultad de CC.EE. y Empresariales. 2008. Disponible en: http://www.echalemojo.org/uploadsarchivos/metodo_delphi.pdf
7. Barr, Sherold. *DataLabs Introduces Single Data Management System to Unify Paper Data Entry and Electronic Data Capture*. Disponible en: http://www.datalabs.com/
8. Bavera, F; Nordio, D; Arroyo, M y Aguirre, J. *JTLex un Generador de Analizadores Léxicos Traductores.* Disponible en: http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r44700.PDF
9. *Bayer AG.* Disponible en: http://www.bayer.com
10. *Bayer Healthcare estandariza la solución de gestión de datos clínicos Medidata Solutions RAVE.* Disponible en: http://www.prnewswire.co.uk
11. Berk, EJ y Scott, C. *JLex: A Lexical Analyzer Generator for Java(TM).* 2003. Disponible en: http://www.cs.princeton.edu/%7Eappel/modern/java/JLex/.
12. CIMAher. *Centro de Inmunología Molecular*. 2009. Disponible en: http://www.cimaher.com/index.php?action=cim
13. *Clinical Conductor Site CTMS*. Disponible en: http://www.bio-optronics.com/CTMSClinicalTrialsManagementSoftware/

14. *Clinical Force*. Disponible en: http://www.perceptive.com/ctms/

15. Cortez, Augusto. *Análisis comparativo entre un analizador sintáctico LL y un analizador sintáctico LR para un lenguaje formal*. Investig. Sist. Inform. RISI 2 (2). Pág. 60-68. 2005. Disponible en: http://sisbib.unmsm.edu.pe/bibvirtualdata/publicaciones/risi/n2_2005/a09.pdf

16. Cortés Fuentes y Carlos Arturo. 2008. *Gramáticas independientes del contexto*. Disponible en: http://cnx.org/content/m16320/latest/

17. *Data Entry Services India*. Disponible en: http://www.dataentryservices.co.in/paper_data_entry.htm

18. de la Torre, R. *Tutorial Jflex/Jlex.* 2010. Disponible en: http://uempl.files.wordpress.com/2010/05/roberto-de-la-torre-tutorial-jflex-pl.pdf.

19. Donnelly, C y Stallman, R. *Bison: El Generador de Analizadores Sintácticos compatible con YACC.* 2006. Disponible en: http://es.tldp.org/Manuales-LuCAS/guides/bison-guide/bison-es-1.27.html.

20. Felip, L. *Generador de compiladores basado en analizadores ascendentes.* Universidad Autónoma de Barcelona. 2009. Disponible en: http://ddd.uab.es/record/69760?ln=en

21. Formella, Arno. *Teoría de autómatas y lenguajes formales.* Universidad de Vigo. 2006. Disponible en: http://trevinca.ei.uvigo.es/~formella/doc/talf05/talf.pdf

22. Gajardo, Luis y Mateu, Luis. *Análisis semántico de programas escritos en Java.* Theoria. Vol. 13: 39-49. 2004. Disponible en: http://www.ubiobio.cl/theoria/v/v13/3.pdf

23. Gálvez. *JavaCUP-Análisis Léxico: JLEX.* Disponible en: http://www.lcc.uma.es/~galvez/theme/cup/anlexcup/jlex.html

24. Gálvez, S y Mora, MA. *Java a tope. Compiladores. Traductores y compiladores con Lex/Yacc, JFlex/Cup y JavaCC*. Universidad de Málaga: 2005. Disponible en: http://www.lcc.uma.es/~galvez/ftp/libros/Compiladores.pdf

25. García, CI. *Introducción a los compiladores.* http://www.scribd.com/doc/4750526/INTRODUCCION-A-COMPILADORES.

26. García Cota, Enrique José. *Guía práctica de ANTLR*. 2003. Disponible en: http://www.lsi.us.es/~troyano/documentos/guia.pdf

27. González. *Tutorial JFlex*. 2010 Disponible en: http://uempl.wordpress.com/2010/05/04/tutorial-jflex-2/

28. Gonzalo García, Martha. *Clasificación de lenguajes formales de Chomsky*. 2001. Disponible en: http://docs.google.com/viewer?a=v&q=cache:y4kCtcB6e8gJ:tux.uis.edu.co/lenguajes/doc/chomsky.doc+Clasificaci%C3%B3n+de+lenguajes+formales+de+Chomsky.&hl=es&gl=cu&pid=bl&srcid=ADGEESiKPlpwJdvMaQCXsgdx0Xyl5moMVkq2Eew9GOQ5nyfuSA71hg29QOW8VrTWMblBj70_0s5Wi5D7FuNHs_y1QIl9YA8wf0gwGnAT6yclTr1IncJsUsNAemQmtbYa0eClvzbuM8JR&sig=AHIEtbTPni1x518IeEiijUrOq3nfsWi02w

29. *Kbeedocs.* Disponible en http://www.kbeedocs.com

30. Kyung-hee, Kelly Moon. *Techniques for Designing Case Report Forms in Clinical Trials*. ScianNews Vol. 9, No. 1 Fall 2006.

31. Labra, J; Cueva, J; Izquierdo, M; Fuente, A; Luengo, M y Ortin, Francisco. *Intérpretes y diseño de lenguajes de programación.* 2004. Disponible en: http://www.di.uniovi.es/~labra/FTP/Interpretes.pdf

32. Manrique, M. *Curso Teoría de compiladores. Analizador Léxico Lex*. Disponible en: http://biblioteca.uns.edu.pe/saladocentes/archivoz/curzoz/Sesion_5.pdf

33. Maythe. *Java Cup - Documentos*. 2010. Disponible en: http://www.buenastareas.com/ensayos/Java-Cup/871197.html

34. *Medidata Solutions*. Disponible en: http://www.mdsol.com

35. MedlinePlus. *Ensayos clínicos: MedlinePlus en español*. 2009 Disponible en: http://www.nlm.nih.gov/medlineplus/spanish/clinicaltrials.html

36. Moráguez Iglesias, Arabel. *Otros conceptos de economía*. 2006. Disponible en: http://www.gestiopolis.com/canales6/eco/metodo-delphi-estadistica-de-investigacion-cientifica.htm

37. Open Clínica. *La nueva generación en captura electrónica de datos y ensayos clínicos.* Disponible en: http://www.openclinica.es/

38. Pamos Garrido, A. *Utilidades para el desarrollo y prueba de programas. Compiladores. Intérpretes. Depuradores.* Disponible en: http://www.apgtic.es/ASI/FPR/Tema_06.pdf

39. Pérez, A. *Procesadores de lenguajes. Cup y el análisis semántico*. 2008. Disponible en: http://www.slideshare.net/launasa/anlisis-semntico-con-cup

40. Pérez, Antonio. *Introducción a la práctica de Procesadores de Lenguajes*. 2010. Disponible en: http://www.escet.urjc.es/~procesal/pracs/pr1011/IntroduccionPractica1011_3.pdf

41. Pinto, M. *Intérpretes.* 2006.

42. Pisabarro, AM. *El generador de analizadores sintácticos Yacc.* Universidad de Valladolid. 2007. Disponible en: http://www.infor.uva.es/~mluisa/talf/docs/labo/L11.pdf

43. *PRA International.* Disponible en: http://www.prainternational.com

44. Puchaicela, LP. *Traductores de lenguaje.* Universidad Técnica Particular de Loja. 2007. Disponible en: http://blogs.utpl.edu.ec/metodologiadeprogramacion/files/2009/05/compiladores-traductores-lenguajes_gberru.pdf

45. Ramírez, KD. *Analizador Léxico*. Disponible en: http://www.kramirez.net/RI/Material/Presentaciones/Analizador%20Lexico.pdf

46. Remache, Katherine. *Compiladores-traductores-lenguajes-de-programación*. 2008. Disponible en: http://blogs.utpl.edu.ec/metodologiadeprogramacion/files/2009/05/compiladores-traductor-lenguajes-de-programacion.pdf

47. s4research. *¿Qué es OpenClinica?* Disponible en: http://www.s4research.es/que-es-openclinica-solutions-cro-barcelona-espana-open-source-ensayos-clinicos

48. Sánchez, S y Rodríguez, D. *JLex. Disponible en:* http://www.cc.uah.es/ie/docencia/ProcesadoresDeLenguaje/2.3.JLex.pdf.

49. Simmross, Federico. *El generador de analizadores sintácticos yacc. Teoría de Autómatas y lenguajes formales.* Disponible en: http://www.infor.uva.es/~mluisa/talf/docs/labo/L8.pdf

50. Vélez Reyes, Javier. *Procesadores de lenguajes. Análisis Sintáctico*. 2010. Disponible en: http://www.lsi.uned.es/procleng/apuntes/2010-2011/PDL.07.Tema4.AnalisisSintacticoDescendente.pdf

51. Veritosoto. *ANTLR*. 2011. Disponible en: http://veritosoto.wordpress.com/2011/02/03/antlr/

ANEXOS

Anexo 1: Encuesta de autovaloración para determinar el coeficiente de conocimientos de los expertos.

Compañero (a):

Con el objetivo de evaluar una herramienta informática desarrollada para validar los datos de los ensayos clínicos, se decide seleccionar un conjunto de expertos del Centro de Inmunología Molecular. Usted es uno de los posibles expertos a seleccionar, por lo cual se hace necesario que responda las preguntas que se muestran a continuación, relacionadas con el grado de dominio que presenta sobre el tema de la investigación.

Nombre (s) y Apellidos: Departamento:

Labor que realiza: Años de experiencia:

Calificación profesional: Ingeniero ___ Licenciado___ Máster ___ Doctor ___

Especialidad:

1. Seleccione en una escala del 1 al 10, el valor que corresponda con el grado de conocimientos que usted posee acerca de la **Validación de datos de ensayos clínicos**, considerando 1 como no tener ningún conocimiento y 10 si posee un pleno conocimiento acerca del tema.

1	2	3	4	5	6	7	8	9	10

2. Valore el grado de influencia que cada una de las siguientes fuentes ha tenido en su conocimiento o criterio sobre el tema.

Fuentes	**Grado de influencia de cada una de las fuentes en su conocimiento**		
	Alto	**Medio**	**Bajo**
Análisis teóricos realizados por usted acerca del tema.			
Su experiencia obtenida.			
Trabajos de autores nacionales.			
Trabajos de autores extranjeros			

Su propio conocimiento del tema en el extranjero.			
Su intuición.			

Anexo 2: Encuesta a los expertos sobre la valoración del intérprete en la validación de datos de ensayos clínicos.

Compañero (a):

Con la presente encuesta se pretende evaluar a partir de la opinión anónima de un grupo de expertos, la herramienta desarrollada para la validación de los datos de los ensayos clínicos. Su valiosa opinión será de gran utilidad para la investigación que se desarrolla y será tomada en cuenta para medir la efectividad de dicha herramienta. Para ello debe responder la siguiente cuestión:

Evalúe en una escala de 1 a 5 puntos, cómo la herramienta desarrollada cumple cada uno de los siguientes requisitos. La evaluación de 1 punto reflejará que usted está completamente en desacuerdo con el cumplimiento del requisito y el valor 5, que usted está muy de acuerdo con el cumplimiento del requisito.

Número	Requisito	Evaluación
1	El lenguaje definido para el establecimiento de las reglas de validación es de fácil comprensión y abarca todas las posibles reglas a definir.	
2	La elaboración de las reglas en el sistema es de fácil manejo para el usuario.	
3	La herramienta permite detectar errores en la elaboración de las reglas.	
4	La herramienta permite en la declaración de las reglas, establecer dependencias entre las variables de tipo entero y real y entre las variables de tipo fecha.	
5	La herramienta ejecuta todas las reglas correctamente definidas, por el Gerente de Datos.	
6	La herramienta permite detectar errores en los datos de los EC entrados en los CRD electrónicos del sistema alasClínicas.	
7	La herramienta contribuye a la validación de los datos de los EC, permitiendo obtener resultados más certeros en los estudios de investigación clínica que se realizan en el CIM.	