UNIVERSIDAD DE HOLGUÍN "OSCAR LUCERO MOYA"

FACULTAD DE INFORMÁTICA Y MATEMÁTICA

BIBLIOTECA DE CLASES PARA EL RECONOCIMIENTO Y EVALUACIÓN DE EXPRESIONES ARITMÉTICAS, RELACIONALES Y LÓGICAS.

TESIS EN OPCIÓN AL TÍTULO DE MÁSTER EN MATEMÁTICA APLICADA E INFORMÁTICA PARA LA ADMINISTRACIÓN.

Autor: Lic. Yoel Caises Almaguer.

Tutores: Dr. Mauro García Pupo.

MSc. Eduardo Escofet Batista.

2006

Resumen

La toma de decisiones en una entidad es un proceso vital que garantiza en gran medida su éxito. Con el desarrollo de la informática este proceso se ha visto favorecido considerablemente a tal punto que pudiera asegurarse que ninguna entidad está exenta de la informatización de sus áreas.

Este proceso requiere de herramientas de desarrollo lo suficientemente eficientes, que garanticen la creación de buenas aplicaciones informáticas en el menor tiempo posible. Desafortunadamente, existen algunas limitaciones que en ocasiones imposibilitan tal propósito. Tal es el caso de sistemas informáticos que trabajen con expresiones aritméticas, relacionales y lógicas, propensas a variar su estructura con facilidad.

Los lenguajes de programación actuales no cuentan con herramientas para reconocer expresiones válidas de este tipo, ni tampoco para su evaluación. Esto obliga a los desarrolladores a crearlas o sencillamente a limitar a los sistemas al trabajo con definiciones estáticas de expresiones, soluciones que incurren en gastos de tiempo adicionales. En esta tesis se propone una biblioteca de clases con tal finalidad.

Abstract

Taking of decisions in an entity is a vital process that guarantees in great measure their success. With the computer science's development this process has been favored considerably to such a point that could make sure that no entity is exempt of the informatization of its areas.

This process requires of tools that guarantee the creation of good computer applications in the smallest possible time. Unfortunately, some limitations that disable such a purpose in occasions exist. Such it is the case of computer systems that work with arithmetic, relational and logical expressions, prone to change their structure with easiness.

The current programming languages don't have tools to recognize valid expressions of this type, neither for their evaluation. This obligate to the developers to create them or simply to limit the systems to the work with static expressions, solution that incur in additional time. This work intends a library of classes to parse and to evaluate those expressions.

Índice

Introducción

La toma de decisiones concernientes a sistemas complejos a menudo supera las capacidades cognitivas del ser humano debido a la cantidad de variables involucradas y a sus sutiles interdependencias. También es conocido que el juicio intuitivo humano está lejos de ser óptimo, viéndose deteriorado con la complejidad y el estrés.

Debido a la importancia que las decisiones de calidad tienen en muchas situaciones, ofrecer una ayuda que mitigue las deficiencias anteriores constituye una de las líneas principales de la ciencia a lo largo de la historia. Disciplinas como la estadística, la economía y la investigación operativa han desarrollado varios métodos para la realización de elecciones racionales. Más recientemente, estos métodos, se han visto potenciados por aportes de la informática, la psicología y la inteligencia artificial en forma de programas de computadora, que van desde herramientas aisladas hasta entornos integrados para la toma de decisiones complejas.

Tales sistemas han ganado popularidad en ámbitos como los económicos, de ingeniería, militares, médicos, entre otros, debido a su utilidad para ayudar a la toma de decisiones precisas y óptimas a partir de grandes cantidades de información proveyendo de diferentes fuentes de información, acceso inteligente al conocimiento relevante, estructuración del proceso de decisión o dando una aproximación heurística a problemas intratables formalmente.

La apropiada aplicación de herramientas de ayuda a la decisión incrementa la productividad, eficiencia, efectividad y competitividad haciendo consecuentemente más segura la planificación, organización e inversión. Ello le permite a la empresa o entidad adaptarse y ubicarse en su entorno con un mayor nivel de competitividad, elevando los niveles de eficiencia y calidad en los productos y/o servicios que oferta.

Diversos son los intentos por lograr una sociedad informatizada donde se tenga acceso rápido, seguro y eficiente a toda la información que se desee. Tal es así, que hoy en día, no se concibe una organización que no esté involucrada en este proceso, que se ha convertido en un indicador de la calidad.

La demanda incremental de sistemas informáticos ha dado lugar a un vertiginoso crecimiento de herramientas para su desarrollo. La creación de lenguajes de programación es una prioridad de grandes compañías en el mundo entero, que compiten constantemente por ofrecer productos que faciliten a los desarrolladores la realización de sistemas informáticos eficientes, fáciles de usar y que puedan ser desarrollados en el menor tiempo posible y con un mínimo esfuerzo.

Actualmente existen muchos lenguajes de programación que ofrecen una gran cantidad de potencialidades, lo que posibilita que se requiera de menos esfuerzo para la creación de buenas aplicaciones informáticas. No obstante, a pesar del rápido avance, el proceso de programación sigue siendo complejo y por ende, queda limitado a un reducido grupo de personas.

En este sentido se han ido mejorando las técnicas, y el desarrollo basado en componentes ha jugado un papel importantísimo. Un programador solo se preocupa por cuál problema puede resolverle un componente, sin importarle cómo lo hace. De esta forma, los sistemas informáticos se desarrollan a una mayor velocidad y con una calidad superior.

No obstante, teniendo en cuenta que las necesidades de un programador son muy variadas, pretender un componente para resolver cada problema es una aspiración aún muy lejana. Nuevas variantes se han estado desarrollando en los últimos años basadas en la construcción de componentes que puedan ser adaptados con facilidad a necesidades particulares.

En el entorno empresarial, comúnmente se presenta la necesidad de evaluar varias condiciones con el objetivo de decidir entre una u otra alternativa, o sencillamente determinar el efecto que produce su combinación. Este tipo de situaciones suelen ser descritas utilizando alguna expresión matemática, relacional o lógica. Entre las más elementales están funciones de costo, de demanda, expresiones para el cálculo del comportamiento de determinado renglón, entre otros muchos ejemplos.

Con las herramientas de desarrollo actuales pudieran crearse sistemas informáticos que traten situaciones como las anteriores muy eficientemente. No obstante, en dependencia de la naturaleza de los datos y de su variabilidad, esta eficiencia pudiera ser cuestionada. Tales sistemas generalmente están

diseñados para trabajar con expresiones fijas o estáticas y no permiten modificar su estructura, ni añadir una nueva situación, o eliminar alguna existente, todo esto de forma automática. Ello haría necesario adaptar la aplicación para reflejar el cambio, tarea que pudiera resultar costosa para la empresa involucrada.

Un problema más complejo se presenta, por ejemplo, cuando se desea construir una aplicación que reciba como entrada una determinada expresión matemática, y en dependencia de su significado deba realizar determinadas acciones, o bien tenga almacenada en una base de datos parte de su lógica, de forma tal que modificando esta información implicaría un cambio en el comportamiento de la aplicación completa, sin requerir cambios en el programa.

Cada una de estas aplicaciones debe ser capaz de analizar la entrada y la validez de la salida, informar sobre un error, entender qué significan las entradas y realizar las acciones indicadas, en caso de que esta sea válida. Este proceso no es una tarea sencilla, de ahí que la necesidad de tal funcionalidad implicaría mucho esfuerzo para el desarrollador de la aplicación.

Sería entonces conveniente contar con herramientas flexibles que brinden estas facilidades, permitiéndole al programador centrarse en la implementación de soluciones, que se traduce en un ahorro de tiempo y esfuerzo, o la personalización de las existentes al entorno de la organización, lo cual sería posible si pudiera contarse con el código de la aplicación.

Recientemente se viene trabajando el concepto de software libre, enfocado en contrarrestar la problemática de los grandes costos que implica la informatización. El software libre ha permitido a pequeñas y medianas empresas adentrarse en el proceso de informatizar sus áreas, aumentando la calidad de sus procesos a costos menores que usando software propietario[1], teniendo como gran ventaja además su flexibilidad, pues además de la gratuidad permite ajustarlo a las necesidades propias de la organización.

[1] Mas. Jordi. Software libre: socialmente justo, económicamente viable y económicamente sostenible.

Vale destacar que si bien estas características de cero costo y código abierto constituyen una gran ventaja, sigue siendo la calidad el indicador fundamental para decidirse por este tipo de soluciones.

Dada la situación antes descrita se plantea el siguiente **problema**: ¿cómo facilitar a los desarrolladores de sistemas informáticos el procesamiento y evaluación de expresiones aritméticas, relacionales y lógicas?

Este problema se enmarca en el **objeto** de estudio: herramientas informáticas de base para el desarrollo de sistemas informáticos de apoyo a la toma de decisiones.

Para resolver este problema se propone como **objetivo** el desarrollo de una biblioteca de clases, libre, para el procesamiento y evaluación de expresiones aritméticas, relacionales y lógicas. El objetivo delimita el **campo** de acción: herramientas informáticas para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas.

La investigación se basó en la **hipótesis** de que una biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas, que sea libre, fácil de usar y de adaptar a necesidades particulares, contribuirá a facilitar la realización y/o adaptación de aplicaciones informáticas que requieran esta funcionalidad.

La investigación se llevó a cabo a través de las siguientes tareas:

1. Estudio preliminar del problema.
2. Estudio de factibilidad.
3. Estudio de las principales técnicas para el reconocimiento y evaluación de expresiones.
4. Elaboración de los fundamentos teóricos relacionados con el software libre y metodologías para el desarrollo de sistemas informáticos.
5. Construcción de la aplicación.
6. Desarrollo de una aplicación práctica.

Entre los métodos se usaron:

- Análisis y Síntesis del objeto de estudio, utilizado en la identificación de los principales procesos involucrados en el análisis y traducción de

expresiones aritméticas, relacionales y lógicas, así como las características de cada uno y las relaciones existentes entre ellos.

- Histórico-Lógico, a través del estudio de varios documentos con vistas a lograr la comprensión de los principales conceptos relacionados con la traducción automática de cadenas, así como el estudio de las principales tendencias que apoyan la hipótesis.
- Hipotético-Deductivo, necesario para la formulación de la hipótesis y para trazar acciones encaminadas a demostrar su validez.

El documento está estructurado en Introducción, dos capítulos, Conclusiones, Recomendaciones, Bibliografía y Anexos.

En la introducción se analizan las situaciones que dieron origen a la creación de la biblioteca y se formaliza el diseño de la investigación que se propone.

En el primer capítulo se realiza un estudio de los principales aspectos teóricos relacionados con el análisis y traducción de cadenas, necesario para la construcción de la biblioteca. Se abordan además las características fundamentales del lenguaje de programación utilizado para su desarrollo. Finalmente se propone una metodología de desarrollo para esta aplicación.

En el capítulo II se detalla la estructura de las clases y de los principales métodos construidos. Se muestra un ejemplo de su aplicación en un sistema para el apoyo a la toma de decisiones así como el estudio de factibilidad realizado para el producto, además de los resultados obtenidos en la encuesta realizada a expertos.

Capítulo 1. Estudio de los fundamentos teóricos.

En este capítulo se tratan las principales técnicas para el análisis y traducción automática de cadenas, recogidas en la teoría formal de los lenguajes, así como las tendencias actuales en el desarrollo de productos informáticos, haciendo énfasis en el lenguaje de programación elegido para la implementación. Se presenta además la metodología de desarrollo propuesta para guiar la creación de la biblioteca.

Técnicas para el reconocimiento y evaluación de expresiones.

El análisis y traducción automático de cadenas es una tarea ardua para cualquier programador. El primer paso es partir de la especificación del lenguaje. Esta especificación debe tener en cuenta:

- El conjunto de símbolos básicos que se usan para su escritura.
- El conjunto de expresiones válidas en ese lenguaje.
- El significado de cada expresión válida.

Definir el conjunto de símbolos básicos de un lenguaje es una tarea fácil, más complicado resulta precisar las reglas que rigen la combinación de esos símbolos para formar cadenas válidas y aún más complejo resulta interpretar el significado de la expresión. A todo este proceso suele llamársele proceso de compilación.

El proceso de compilación de una cadena, consta de varias fases, cada una relacionada con las tareas definidas anteriormente para la especificación de un lenguaje. Entre ellas están el análisis lexicográfico, análisis sintáctico así como análisis semántico y generación de código.

Análisis lexicográfico.

Una cadena es una larga secuencia de caracteres, incluyendo espacios, cambios de línea, tabulaciones u otros caracteres especiales, cada elemento de estos con un significado.

Usualmente, se descompone la cadena en estos componentes fundamentales con vistas a asociarles un significado sintáctico en correspondencia. Este

proceso se conoce como análisis lexicográfico y cada una de las secuencias especiales que reconoce es denominada lexema o Token[2].

Aunque en determinados contextos, esta fase puede solaparse con otras, existen varias razones para separar el analizador lexicográfico del resto de las fases de un compilador, entre ellas:[3]

1. Esta separación tiene importancia desde el punto de vista de la flexibilidad entre máquinas. Mientras que el lenguaje propiamente no cambia de una máquina a otra, el conjunto de caracteres utilizados para su representación externa sí puede variar.
2. Como el analizador lexicográfico transforma una secuencia de caracteres en un símbolo para el analizador sintáctico, este último tendrá la información sintetizada en menor cantidad de símbolos. Algunos chequeos de contexto que son necesarios hacer, pueden ser efectuados de forma más artesanal por el analizador lexicográfico sin necesidad de pasar estos al mecanismo formal de análisis sintáctico. Gran parte de la compilación se dedica a recorrer secuencia de caracteres por lo que tener ambos analizadores separados permite hacer más efectivo el trabajo con los caracteres.

La separación permite la programación y la realización del analizador sintáctico y el analizador lexicográfico en forma modular e independiente. Estos pueden trabajar de dos formas[4]:

a. Se hace primero un análisis lexicográfico de toda la cadena, de manera que el analizador sintáctico procese la cadena representada en una forma simbólica interna más compacta.
b. Ambos analizadores trabajan paralelamente.

Realizar la primera variante presupone contar con una estructura datos adicional en la que se almacenen las salidas del analizador lexicográfico. Además, este tipo de variante resultaría más útil cuando se tiene un

---

[2] Parsons, Thomas. Introduction to compiler construction.

[3] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación.

[4] Toro Bonilla, Miguel. Apuntes de compiladores.

mecanismo de recuperación de errores que garantice el análisis de toda la cadena. De lo contrario, pudiera incurrirse en un procesamiento innecesario.

Puesto que la sintaxis de los lexemas es, en general, relativamente sencilla, pueden desarrollarse realizaciones muy efectivas de los analizadores lexicográficos y su construcción tiene su base en un tipo de máquina de estados conocida como *autómata finito*[5].

En el reconocimiento de un elemento a partir de un autómata finito, con frecuencia se presenta el problema conocido como *no determinismo*, dado cuando existen estados en el autómata en los que para un mismo símbolo de la cadena se pueden realizar más de un movimiento.

Es conveniente entonces que el autómata sea determinista; es decir, que en cada estado o configuración exista solo un movimiento posible. Afortunadamente existe un algoritmo para realizar esta conversión[6], lo cual es posible gracias a que la clase de los lenguajes aceptados por los autómatas finitos no deterministas es la misma que la aceptada por los autómatas finitos deterministas.

Este resultado es muy importante, porque la programación de un autómata finito determinista es mucho más sencilla que la de un autómata finito no determinista.

Análisis sintáctico

El analizador sintáctico determina la estructura de la cadena a analizar, es decir, el orden que deben tener los componentes que determina el analizador lexicográfico.

Para definir un lenguaje deben especificarse sus características y para ello se hace necesario establecer el alfabeto de este, y las reglas que rigen sus construcciones, que forman parte de una estructura generativa conocida como gramática.

Existen dos tipos fundamentales de analizadores sintácticos, los descendentes y los ascendentes. Los descendentes inician el proceso de análisis de una

---

[5] Parsons, Thomas. Introduction to compiler construction.

[6] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación.

cadena tomando el símbolo distinguido de la gramática como la raíz del árbol, confrontando los terminales que aparezcan en las hojas de este con los símbolos de la cadena a reconocer, utilizando siempre para las derivaciones el criterio del noterminal más a la izquierda. Finaliza cuando no se puede derivar más la cadena.

Por otra parte, un analizador ascendente parte de la cadena y trata de hacer reducciones, utilizando siempre el noterminal más a la derecha, hasta llegar a la raíz del árbol, que en este caso sería el símbolo distinguido de la gramática.

Análisis sintáctico descendente.

En el proceso de construcción de un analizador sintáctico descendente, pueden presentarse algunos problemas, originados fundamentalmente por la estructura de las reglas de la gramática. Entre estos están las reglas recursivas a la izquierda y el conocido como retroceso o *backtracking*.

La recursividad izquierda se presenta con las producciones del tipo $A \rightarrow A\alpha$, que son reglas recursivas a la izquierda.

El problema radica en que cada símbolo leído de la cadena guía al analizador sintáctico en las acciones a realizar, es decir, le dice cuál regla de la gramática desarrollar. Debido a la estructura de la regla, el analizador no tiene elementos para decidir.

La solución a este problema consiste en hacer transformaciones a la gramática de forma tal que se obtenga una equivalente sin recursividad izquierda en ninguna de sus reglas.

Existen dos tipos de recursividad izquierda, la directa ($A \rightarrow A\alpha$) y la indirecta que tiene la forma[7]:

$A \rightarrow B\alpha$ |...

$B \rightarrow A\beta$ |...

Una vez que se realizan las transformaciones necesarias en una gramática, de forma tal que se eliminen las anomalías anteriores, no puede garantizarse la realización del analizador.

[7] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación

En ocasiones es difícil predecir el camino a seguir para el reconocimiento de una cadena, a pesar de no tener reglas recursivas a la izquierda. La estructura de la gramática puede ser tal que en algún punto del análisis no se elija la regla correcta, generando de esta forma un camino erróneo o sencillamente imposibilitando la construcción de este.

Si a partir de una gramática puede construirse el camino, entonces la cadena pertenece al lenguaje que genera la gramática, lo que obviamente sería un problema si el camino es erróneo debido a que se tomarían como válidas cadenas que no pertenecen al lenguaje. Por otra parte, equivocar el camino, puede conducir a que cadenas correctas pueden no ser reconocidas.

Una solución a este problema consiste en, una vez que se compruebe que no se puede construir el árbol, se retrocede al nivel anterior y se desarrolla otra rama, proceso que se repite hasta explorar todas las posibilidades. Obviamente esta solución no es factible puesto que, además del esfuerzo computacional, solo tendría sentido este retroceso o backtracking para cadenas válidas.

Otra técnica más efectiva consiste en modificar las reglas de la gramática con el objetivo de hacerla determinista y para ello se realiza una factorización en reglas con sufijo común, alternativa que en ocasiones no es posible realizar. Otra variante consiste en determinar dos conjuntos, denominados FIRST y FALLOW, que ayudan a predecir el camino a seguir y que su uso da nombre a un tipo de analizador sintáctico descendente, el predictivo recursivo[8].

La eliminación de la recursividad izquierda y la construcción del FIRST y el FOLLOW hacen del analizador predictivo una herramienta práctica, no obstante, para cada producción de la gramática se necesita una función y cualquier cambio que se introduzca en la gramática requiere que estas funcionalidades sean reprogramadas[9].

A pesar de todas estas alternativas para la solución de posibles errores, en ocasiones no se logran gramáticas que permitan hacer algunas predicciones, característica negativa en los analizadores descendentes. Ello imposibilita su realización y sugiere el uso de otros tipos de analizadores, como es el caso de los ascendentes.

---

[8] Parsons, Thomas. Introduction to compiler construction

[9] Toro Bonilla, Miguel. Apuntes de compiladores

Análisis sintáctico ascendente.

Un analizador ascendente construye, a partir de una expresión dada, una derivación a la extrema derecha[10], aplicando las producciones de la gramática. Si llega al símbolo distinguido es señal de que la expresión pertenece al lenguaje generado por esta.

El modo en que opera consiste en analizar si el elemento leído de la cadena forma la parte derecha de una regla. Se procede entonces a hacer la sustitución por la parte izquierda de esta. Esto trae como consecuencia, que en ocasiones, la lectura de un término en la cadena no componga la parte derecha de una regla, conllevando a la necesidad de ir almacenando en alguna estructura la parte derecha formada hasta ese momento.

Un modo simple de operar es tener una pila que almacene los elementos leídos de la cadena hasta completar la parte derecha de una regla. Esta reducción se realiza extrayendo de la pila la parte derecha de la regla y almacenando la parte izquierda de esta.

Estas operaciones (conocidas como *shift* y *reduce*) son realizadas hasta que en el tope de la pila quede el símbolo distinguido de la gramática.

La parte derecha de la producción a ser reducida se conoce como handle[11] y en cada análisis pueden aparecer varias candidatas. La elección favorecerá a aquella que retroceda un paso en la derivación de la cadena, y que por ende, acerque más al símbolo distinguido.

En caso de $\lambda^{\beta}\alpha$, para que $^{\beta}$ sea un handle, debe ser la parte derecha de una regla. Esto significa que si $A \rightarrow {}^{\beta}$, entonces $\lambda A\alpha \rightarrow \lambda^{\beta}\alpha$ está en la derivación a la extrema derecha. Pero para que la reducción por $\lambda A\alpha$ sea correcta debe verificarse que el símbolo distinguido de la gramática deriva a $\lambda A\alpha$ en uno o más pasos.

Esto ocurre debido a que la reducción $A \rightarrow {}^{\beta}$ tiene que ser capaz de reducir el subárbol cuya raíz es A sin interactuar con el resto del árbol. Si el símbolo distinguido no deriva a $\lambda A\alpha$, entonces se caería en una rama de no éxito. Estos dos requerimientos definen un handle.

---

[10] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación

[11] Parsons, Thomas. Introduction to compiler construction.

Existen varios tipos de analizadores que utilizan esta técnica ascendente. Entre los más poderosos se encuentran el lineal a la derecha o linear right, con las siglas en inglés LR, entre otras variantes.

El modo en que operan los analizadores sugiere la transición entre varios estados[12]. La estructura de la tabla de transiciones facilita el tratamiento de los errores. Cada entrada en blanco en la tabla corresponde a algún tipo de error. De esta forma se puede hacer una biyección entre estos espacios y una jerarquía de errores.

El proceso de recuperación de errores suele ser complejo. La compilación de una cadena pudiera abortarse apenas se detecta el primer error. No obstante, sería factible que para aquellas construcciones en las que la respuesta a este proceso sea lenta, que en una pasada se detectaran todos los errores cometidos en la cadena fuente.

Si puede se construida una tabla de estados para un analizador LR a partir de una gramática, entonces puede decirse que esta es del tipo LR. Si la gramática no lo es, entonces en la construcción de la tabla pudieran presentarse algunos inconvenientes.

Una característica que hace que una gramática no sea LR es la ambigüedad, que ocasiona un no determinismo en la tabla de acciones puesto que la tabla de estados, en un momento determinado, puede indicar hacer la realización de operaciones contrarias[13]. Este conflicto es conocido como *shift-reduce*.

Otro problema que pudiera presentarse es cuando, para alguna configuración, puede efectuarse más de una reducción. En este caso existen varias técnicas para ayudar a decidir, pero en ocasiones son insuficientes imposibilitando la toma de una decisión acertada[14].

En este punto entra a jugar el programador del analizador, quien debe decidir la acción a realizar, en dependencia de las características del lenguaje fuente, y constituye una de las ventajas de este analizador con respecto a los descendentes, en los que la gramática rige el control total del análisis de una cadena.

---

[12] Parsons, Thomas. Introduction to compiler construction.

[13] Toro Bonilla, Miguel. Apuntes de compiladores.

[14] Parsons, Thomas. Introduction to compiler construction.

Comparación entre los analizadores sintácticos.

Cuál es el analizador más indicado es una pregunta difícil de responder. Ello depende de las especificidades del lenguaje a generar. El analizador descendente es fácil de construir manualmente pero la generación de código para la traducción es mucho más compleja que si se usara un analizador ascendente, elemento que facilitaría la adaptación de esta fase por parte de los usuarios de la biblioteca.

No todos los analizadores pueden ser aplicados a cualquier gramática, ya que para poder construir las tablas de análisis correspondientes hay que imponer ciertas restricciones a las reglas. En este sentido, se puede establecer una clasificación entre las gramáticas que pueden ser analizadas según cada uno de los métodos anteriores[15].

Entre todos los métodos existentes, el que cubre un mayor espectro de gramáticas en la práctica es sin duda el método LR[16][17]. Los métodos descendentes imponen muchas restricciones a la gramática. Ello hace que sean menos usados que los LR, aún pese al inconveniente de que la construcción de la tabla es mucho más laboriosa.

Los métodos descendentes suelen ser más simples de construir debido a que la gramática es más sencilla, y no contiene producciones nulas. Sin embargo, tal como se indicó en el punto anterior, puede ser más difícil, si no imposible, la tarea de hallar la gramática que cumpla las restricciones necesarias. Este es el principal problema de los analizadores descendentes.

En cuanto a la detección y recuperación de los errores, los mejores métodos son los LR, ya que detectan el error tan pronto como es posible, y posibilitan un correcto diagnóstico de los mismos. Los métodos basados descendentes, por su parte, brindan un diagnóstico muy pobre, y en algunos casos no detectan el error o realizan la detección tras haber leído una serie de símbolos adicionales, con la dificultad que esto entraña para la adecuada recuperación del proceso de análisis.

---

[15] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación

[16] Parsons, Thomas. Introduction to compiler construction.

[17] Toro Bonilla, Miguel. Apuntes de compiladores

En cuanto al uso de memoria, todos los analizadores sintácticos pueden ser representados mediante tablas. Los analizadores descendentes suelen generar tablas de longitud menor que las correspondientes LR. Sin embargo, por lo general, las gramáticas que se emplean para el análisis contienen mayor número de reglas. Esto se debe a que la eliminación de la recursividad izquierda, o la factorización, necesarias para adaptar la gramática para este tipo de analizador introducen reglas adicionales, por lo que en la práctica se equipara la necesidad de memoria.

En lo que se refiere a la rapidez del análisis, el método descendente es algo más lento que el ascendente debido al mayor número de reglas de producción, y el consiguiente mayor número de movimientos necesarios para alcanzar la cadena final a partir del axioma.

Generación de código intermedio.

El objetivo general de todo este proceso no solo es reconocer si una cadena es correcta, sino transformarla en un código que sea ejecutable en alguna computadora.

Este código objeto puede ser ejecutado directamente por una computadora real, en este caso el lenguaje objeto es el lenguaje de máquina, o puede ser ejecutado por una computadora virtual; es decir, por una computadora no existente realmente. En este caso el lenguaje objeto debe ser interpretado por un programa, conocido como intérprete, que sí es ejecutado en alguna computadora real.

Muchas construcciones traducen a una forma intermedia que puede servir de lenguaje objeto virtual; es decir, que puede ser ejecutada por un programa intérprete, o puede servir para que, a partir de ella, un programa generador de código transforme esta forma interna en instrucciones de máquina, simplificando de esta forma la construcción del compilador, al independizar gran parte de este de las especificidades de una máquina determinada[18].

[18] Parsons, Thomas. Introduction to compiler construction.

Tipos de formas internas.

En la representación interna de toda cadena deben distinguirse dos clases de elementos: operadores y operandos. Esto no es más que el reflejo de lo que en la cadena son las acciones y los datos. Las diferencias en las formas internas están dadas por la forma en que están relacionados los operadores y los operandos.

En la ejecución de un código en lenguaje de máquina, el procesador central realiza la decodificación, interpretación y ejecución de una operación que puede involucrar cero, uno o más operandos. El que no sea confundido un operador con un dato, depende de cómo haya sido escrita la cadena en lenguaje de máquina. Estos mismos conceptos son también aplicables a una forma interna.

La forma interna debe ser interpretada por un programa generador de código, que la pasa a un código de máquina específico o debe ser interpretada por un programa intérprete que directamente hará la ejecución de esta. En ambos casos, este programa debe poder diferenciar los operadores de los operandos.

Una cadena puede tener implícitas varias acciones y varios datos; es decir, varias operaciones y varios operandos. Sin embargo, a nivel de máquina esa estructuración no existe; es decir, la computadora solo puede ejecutar las operaciones en una forma lineal y todas las acciones se reprendan explícitamente.

Si se tiene un multiprocesador lo que se puede hacer es la ejecución en paralelo de varias acciones. Esto significa que en la ejecución del programa, a nivel de máquina, se crean operandos que no corresponden a ningún operando explícito del programa fuente, los cuales deben conservarse para su utilización posterior por otra operación. El ejemplo más elemental lo constituyen las expresiones, donde constantemente se necesita almacenar resultados intermedios como operandos para operaciones posteriores.

Estos operandos no aparecen reflejados explícitamente en la expresión fuente. Una de las diferencias entre las formas internas consiste en cómo se conservan y cómo se trabaja con estos operandos intermedios. Atendiendo a este aspecto se puede hacer énfasis en dos tipos de representaciones internas,

la forma interna orientada a pila y la forma interna orientada a variables temporales.

Forma interna orientada a pila.

En esta forma interna los resultados intermedios son almacenados en una pila. Una de las notaciones más conocidas que trabaja con esta filosofía es la llamada notación polaca o postfija. En esta notación los operandos aparecen antes que los operadores. Los operandos son llevados a una pila, de manera que al interpretarse un operador, este puede suponer que sus operandos están en la pila. El resultado de una operación se deja en la pila[19].

Forma interna orientada a variables temporales.

Otra forma interna de representación, consiste en utilizar variables temporales para la representación de los resultados intermedios; es decir, para la representación de los operandos que no corresponden explícitamente al programa fuente, sino que son creados como resultados de los cálculos intermedios[20].

Al igual que en la notación, los operadores aparecen en el orden en que son evaluados. Sin embargo, los operandos no se referencian implícitamente a través de las posiciones que ocuparán durante la ejecución. Esto implica que los operandos intermedios deben ser dejados explícitamente en variables temporales, para que más tarde puedan ser referenciados por otros operadores.

Una notación característica para esta filosofía de trabajo es la notación de cuádruplos. Esta puede caracterizarse en la forma:

Operador, operando1, operando 2, resultado.

La utilización de una forma interna orientada a pila es mucho más sencilla que una forma interna orientada a variables temporales o notación en cuádruplos, puesto que se evita la generación de variables temporales para operaciones intermedias, tarea que pudiera resultar tediosa y requiere de un estricto control de su manejo.

---

[19] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación

[20] Toro Bonilla, Miguel. Apuntes de compiladores.

Una combinación de ambas pudiera ser una variante razonable para la realización de la forma interna. De esta forma, en determinadas ocasiones se generarían variables para almacenar resultados que luego sean apilados.

Tabla de símbolos.

Una herramienta muy utilizada en el proceso de traducción de cadenas es la tabla de símbolos. Su función es brindar información, necesaria en el proceso de compilación, a las distintas fases[21].

Está compuesta por varias entradas, cada una con la información referente a un elemento del lenguaje diseñado. Se estructura varía en dependencia de las características del lenguaje y del nivel de detalle necesario para que cada una de las fases pueda tomar decisiones acertadas.

Una de las fases que primero hace uno de la tabla de símbolos es el analizador lexicográfico. Como la función de este analizador es identificar los elementos de la cadena de entrada, debe ser capaz de discernir entre una palabra reservada por el lenguaje definido y una variable, cuya estructura es la misma. En este caso puede optar por obligar al programador a cambiar la estructura para alguna de las anteriores, lo que haría más ilegible el lenguaje, o hacer uso de la tabla de símbolos para almacenar las palabras claves.

Otra información necesaria es el nombre de las variables. Si una variable aparece varias veces en una cadena, al ser evaluada debe tomar siempre el mismo valor. De esta forma, solo se necesita tener una localización de memoria para que este valor sea almacenado, y sencillamente en la forma interna se referencia esta dirección.

Tendencias y tecnologías actuales.

En la actualidad, el movimiento del software libre ha estado jugando un papel preponderante en muchos países en el proceso de informatización de sus áreas. En los últimos años este concepto ha ganado gran popularidad atrayendo la atención y la colaboración de centenares de empresas en el mundo entero.

---

[21] Katrib M, Miguel. Lenguajes de programación y técnicas de compilación

El software libre se ha desarrollado orgánicamente y cualquiera de sus instalaciones está conformada de distintos componentes provistos por distintos participantes. Se define el término libre como *la libertad que se les brinda a los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software y no con el precio del software*. De modo más preciso, se refiere a tres libertades de los usuarios del software[22].

- *La libertad de usar el programa*, con cualquier propósito.
- *La libertad de acceder al código fuente* con vistas a adaptarlo a necesidades particulares, sin necesidad de notificar la modificación.
- *La libertad de distribuir copias* a otros usuarios, ya sea gratis o cobrando una cantidad por la distribución. Estas copias deben incluir tanto los ejecutables del programa como su código fuente, ya sean versiones modificadas o sin modificar. Para que las libertades de hacer modificaciones y de publicar versiones mejoradas tengan sentido, se debe tener acceso al código fuente del programa. Por lo tanto, constituye una condición necesaria para el software libre.

En los últimos años han surgido varios lenguajes de programación libres, que han ganado mucha popularidad por las potencialidades que brindan. Entre los más difundidos está el Java, de la compañía Sun Microsystems, que ha acaparado la atención de muchos seguidores en el mundo entero.

Tecnología Java.

Para que un programa se ejecute debe estar expresado en un lenguaje que sea comprensible por la computadora. Este lenguaje, comúnmente llamado lenguaje de máquina o código nativo, está compuesto por cadenas de bits que son interpretadas por mecanismos internos de la computadora. En todo este proceso interviene de manera significativa el sistema operativo, como una capa de abstracción entre los programas y las especificidades del hardware.

Con la existencia de diversos sistemas operativos, el término incompatibilidad comenzó a ser muy usado entre los seguidores de la informática. La realización

[22] Mas, Jordi. Software libre: socialmente justo, económicamente viable y económicamente sostenible.

de herramientas de desarrollo para sistemas operativos específicos constituyó una barrera que aún en los tiempos actuales no ha podido ser obviada.

Con el objetivo de contrarrestar esta problemática, comenzaron a desarrollarse herramientas que podían ser utilizadas en varios sistemas operativos, y fueron bautizadas con el término *multiplataformas*.

Las primeras herramientas de este tipo incluyen entre sus librerías, funciones comunes a los sistemas operativos más usados. De esta manera, un programa realizado en estos lenguajes puede ser compilado en diferentes plataformas sin necesidad de hacer cambios en su código.

Esta solución cuenta con la limitante de que en ocasiones los programas requieren de otras funcionalidades no comunes entre los sistemas operativos. Ante este problema, se debe especificar en el código del programa cuál funcionalidad utilizar, en dependencia del sistema operativo sobre el que la aplicación será ejecutada. Esta variante, aunque novedosa, hace más engorroso el proceso de construcción de los programas debido a la cantidad de especificidades que deben tenerse en consideración.

Otra solución, más ingeniosa, fue aplicada en el desarrollo de un lenguaje de programación Java.
Este lenguaje, a diferencia de los lenguajes convencionales como C, Pascal, entre otros, no convierte sus programas a código que el sistema operativo de la máquina donde se compiló puede entender.

El proceso de compilación tiene otras características: los programas son convertidos a una forma intermedia, término que se introdujo en el epígrafe anterior, y que en el caso de este Java es conocida como *ByteCode,* que no depende del sistema operativo donde es ejecutada la aplicación.

Este código luego es ejecutado por un intérprete, que será el encargado de convertir el código compilado a código particular de la computadora utilizada[23].

Lo novedoso de esta propuesta radica en que el *ByteCode* no tiene ningún vínculo con el sistema operativo, y de esta forma se evita la necesidad de realizar un programa diferente para cada uno, o la necesidad de recompilar el código para la corrida del programa sobre otro sistema, utilizando la alternativa

[23] Thinking in Java.

tratada con anterioridad. De esta forma, solo dependerá del sistema el programa intérprete, conocido como Máquina virtual[24], lograr la comunicación con el sistema operativo. Un aspecto negativo de esta variante es la necesidad de tener un proceso adicional en la fase de ejecución de un programa en java, el intérprete.

Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores. Esta característica hace que muchos expertos opinen que Java es el lenguaje de la informática moderna, porque incluye todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes[25].

Es además un lenguaje que permite el desarrollo de arquitecturas cliente-servidor y distribuidas. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

Su sintaxis es muy parecida a la de los lenguajes más usados, como C++, lo cual facilita su aprendizaje. A diferencia de la mayoría de los lenguajes de programación, no es necesario liberar la memoria que se usa pues presenta un recolector de basura que se encarga de liberar la memoria reservada por las aplicaciones. Elimina además, características de otros lenguajes como son: la aritmética de punteros, las macros, las referencias y la definición de tipos[26].

Puede además ser ejecutado como aplicación independiente, como applet, ejecución como servlet, entre otros. Un applet es una aplicación especial que se ejecuta dentro de un navegador como Netscape Navigator o Internet Explorer al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un servlet es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

---

[24] García Carballeira, Félix. Arquitectura de la Máquina Virtual Java

[25] Thinking in Java

[26] García Carballeira, Félix. Arquitectura de la Máquina Virtual Java

Realiza además, verificaciones en busca de errores tanto en tiempo de compilación como en tiempo de ejecución. Se encarga del manejo de la memoria que usan los programas, liberando al programador de esta responsabilidad. Presenta un buen mecanismo de manejo de excepciones, obligando al programador a manejar las que se puedan producir en determinado momento o especificar que su código puede lanzar esa excepción.

Soporta las tres características básicas del paradigma de programación orientada a objetos: herencia, polimorfismo y encapsulamiento. Para mantener la sencillez del lenguaje no implementa herencia múltiple. Permite además, a través de la creación de interfaces (*interface*), definir el comportamiento de un conjunto de clases que implementen dicha interfaz[27].

Además de la portabilidad que brinda por ser de arquitectura independiente, implementa otros estándares que facilitan el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos diferentes, como Microsoft Windows, Unix/Linux, Mac[28].

Existen implementaciones de la Máquina Virtual Java que se ejecutan en dispositivos PDA, como puede ser un Palm PDA o un teléfono celular, además, muchas grandes compañías están produciendo *ships* y microprocesadores capaces de ejecutar aplicaciones Java sin necesidad de una máquina virtual alcanzando gran velocidad en la ejecución de las aplicaciones y el ahorro de memoria al no ser necesario la instalación de la máquina virtual.

Permite la creación de aplicaciones que tengan más de un hilo de ejecución, con lo cual se logran aplicaciones que puedan realizar más de una tarea de manera simultánea, siempre y cuando la arquitectura de hardware y el sistema operativo lo permitan. Puede ser un hilo que se encargue de manejar la interacción del usuario con la interfaz de la aplicación y otro hilo que realice los cálculos u otras operaciones que deba realizar el sistema.

Permite la definición de los permisos que tendrá una aplicación. El ejemplo típico está en la ejecución de los Applets, pequeñas aplicaciones que se

---

[27] Thinking in Java.

[28] García Carballeira, Félix. Arquitectura de la Máquina Virtual Java

ejecutan en un navegador, los cuales no tienen permisos para acceder a la información almacenada en el disco duro de la máquina donde se estén ejecutando.

Una de las principales características de Java es que es utilizado parta el desarrollo de una gran cantidad de proyectos, muchos de ellos *open source*, lo cual lo convierte en un lenguaje muy amplio y con una cantidad de librerías que permite realizar las operaciones más disímiles.

El entorno de desarrollo de Java.

Existen distintos programas que permiten desarrollar código Java. La compañía Sun, distribuye gratuitamente el Java Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java con posibilidades para la detección y corrección de errores. Los IDEs (Integrated Development Environment), por su parte, son entornos de desarrollo integrados[29].

En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar análisis de la ejecución gráficamente, frente a la versión que incorpora el JDK basada en la utilización de una consola.

Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa.

El compilador es una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java, con extensión .java. Si no encuentra errores en el código genera los ficheros compilados, con extensión *.class. En el JDK de Sun dicho compilador se llama javac.exe. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra.

El Java Runtime Enviroment (JRE), por su parte, provee las librerías, la Máquina Virtual Java y el resto de los componentes necesarios para la ejecución de las aplicaciones y los Applets escritos en Java. Este producto

[29] Thinking in Java.

puede ser redistribuido con las aplicaciones para evitar que las mismas no se ejecuten en las máquinas de los clientes [Java][30].

El JDK incluye el JRE más las herramientas de la línea de comandos usados en el desarrollo de las aplicaciones Java, como son: el compilador y el *debugger*, necesarios para el desarrollo de las aplicaciones y Applets de Java.

Metodologías para el desarrollo de sistemas informáticos.

El desarrollo de todo software constituye un proceso riesgoso y generalmente difícil de controlar. La cantidad de personal; en ocasiones excesiva, otras en déficit, los sistemas de organización, los métodos de control, el dominio sobre el tema y sobre las herramientas de desarrollo, la falta de conocimientos sobre asuntos informáticos por el lado de los clientes así como el tiempo que transcurre hasta la entrega final del producto, son algunos de los factores que afectan el ciclo de desarrollo de una aplicación.

Esto sugiere contar con métodos de trabajo lo suficientemente efectivos como para garantizar el producto que los clientes desean, haciendo un uso óptimo de los recursos. Con tal propósito surgieron las metodologías de desarrollo de software, imponiendo un proceso disciplinado con el fin de hacer más eficiente el proceso de desarrollo de productos informáticos.

Metodologías como ADESA[31], ADOOSI[32], OMT[33], entre otras, han alcanzado gran aceptación por su influencia significativa en la calidad de muchos sistemas informáticos. Sin embargo, ha predominado el criterio de que en determinados contextos, su uso implica mucha laboriosidad, ocasionando un retardo en el tiempo de desarrollo del producto informático. Esto se debe al hecho de que su su uso se limita a específicos de proyectos, y no permiten ser adaptadas a nuevas necesidades[34].

Además, se suma el hecho de que estas pretenden predecir una gran parte del proceso de desarrollo del software en detalle, lo cual se corresponde más con un entorno estático que con las condiciones cambiantes que la realidad impone.

---

[30] García Carballeira, Félix. Arquitectura de la Máquina Virtual Java

[31] ADESA

[32] ADOOSI

[33] Metodología OMT

[34] Métodos Heterodoxos en Desarrollo de Software

Como una alternativa a esta situación, en los últimos años se ha venido trabajando en un nuevo concepto de metodolgía, las ágiles, con el objetivo fundamental de introducir cambios significativos en determinados aspectos de las metodologías tradicionales.

Entre las ventajas de estas puede citarse que exigen una menor cantidad de documentación para una tarea dada y son más orientadas al código, lo cual, en ocasiones resulta conveniente. Otra característica significativa es que todas siguen un patrón de desarrollo iterativo e incremental, lo que facilita que el proceso sea más adaptable al cambio y no se requieran predicciones demasiado detalladas y a largo plazo.

A continuación se describen las metodologías que fueron analizadas en este trabajo.

Metodologías de desarrollo Rational Unified Process (RUP).

Una de las metodologías más usadas en los últimos años ha sido RUP, creada por la compañía Rational Unified Process en la década de los noventa.

En ella se resumieron las características más exitosas de algunas de las metodologías que le precedieron y el resultado de la experiencia de personalidades prestigiosas del mundo de la ingeniería del software.

Esta metodología ofrece un marco general de trabajo, y para ser aplicada a un proyecto, debe ser adaptada a necesidades particulares, lo cual constituye una ventaja frente a sus predecesoras. Impone además, un ciclo de vida de cuatro fases para la realización de un producto[35].

En la primera de ellas, denominada fase de concepción, se centra en la visión del proyecto, encaminada a planificar las restantes fases.
En la fase de elaboración, se determina la arquitectura óptima a través de la identificación de los requerimientos principales del sistema y a partir de estos se construye lo que se conoce como línea base arquitectónica, encaminada a

[35] JACOBSON, Ivar. El Proceso Unificado de Desarrollo de Software

determinar las vías de comunicación entre los principales procesos y partes del sistema[36].

En la fase de construcción se obtiene la capacidad operacional del producto. Finalmente se realiza la fase de transición, que consiste en la entrega al cliente una versión final del producto, que será objeto de pruebas, que pudieran arrojar la necesidad de realizar algunas modificaciones.

En cada una de las fases pueden ser realizadas varias iteraciones. No obstante, en la fase de construcción es donde tienen lugar un mayor número de estas, cuyo número y duración varían en dependencia del proyecto. Para cada iteración se realizan pruebas como confirmación de que el comportamiento hasta el momento es el deseado y que este da respuesta a todas las solicitudes. En la iteración siguiente, se corrigen las deficiencias de la versión actual y se añaden a la versión nuevas funcionalidades.

A pesar de las ventajas de esta metodología respecto a las metodologías tradicionales, se decidió no utilizarla en la construcción de la biblioteca de clases que se propone. Teniendo en cuenta que debido a que la mayor laboriosidad para su realización está dada en la complejidad de los algoritmos utilizados y no en la cantidad de procesos involucrados y que el grupo de desarrollo es reducido, su uso implicaría adaptarla a estas necesidades, proceso que implicaría un gasto de tiempo adicional innecesario.

Programación Extrema (XP).

XP[37] es una de las metodologías de desarrollo de software más exitosa en la actualidad y se aplica a sistemas que requieren ser desarrollados en poco tiempo, por un grupo reducido de programadores.

Esta metodología propone un ciclo de vida de tres etapas para el desarrollo de un producto informático.

- Interacción con el cliente.
- Planificación del proyecto.
- Diseño, desarrollo y pruebas.

[36] JACOBSON, Ivar. El Proceso Unificado de Desarrollo de Software

[37] Wake, William. Extreme Programming Explainned

Entre las características más distintivas de esta metodología está la realización de pruebas a los principales procesos, de tal manera que permita obtener a priori los posibles errores.

Con vistas a agilizar el tiempo de desarrollo, propone la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio. La refactorización, por otra parte, es otra de sus características distintivas, y consiste en la obtención de un código legible, para un mejor entendimiento con vistas a su reutilización y adaptación posteriores.

Esta metodología responde a un modelo de desarrollo iterativo e incremental, con la realización de estimaciones de tiempo y de los recursos requeridos para terminar cada característica. Las iteraciones deben ser cortas, y al igual que en RUP, en cada una se termina una porción pequeña pero funcional del proyecto y se realiza diseño, codificación, pruebas y lanzamiento. Finalmente, cada lanzamiento se integra con los anteriores.

Cada iteración, además, cuenta con un cierto número de historias. Una historia es la unidad de funcionalidad en un proyecto XP, y para cada una se especifica el tiempo y las tareas necesarias para su realización.
En cada iteración se van añadiendo nuevas funcionalidades, que son el resultado de la retroalimentación lograda con la versión anterior y la realización de nuevas historias, planificadas para ese momento.

Esta metodología sugiere un uso muy reducido de artefactos, haciendo el proceso de documentación menos significativo a diferencia, como se analizó con anterioridad[38].

Dadas las características de la biblioteca que se propone en esta investigación, teniendo en cuenta que uno de los objetivos que se persigue es facilitar el proceso de adaptación de esta a nuevas necesidades, lograr un código legible es fundamental. El uso de XP ayudaría a lograr tal aspiración. Además, atendiendo a que el tamaño de la biblioteca es pequeño, y que para su elaboración no se requiere de un grupo de personas elevado, su uso es conveniente.

[38] Wake, William. Extreme Programming Explored.

Conclusiones del capítulo

A partir de la bibliografía consultada se puede concluir que:

- Dividir el proceso de análisis y traducción en tres fases análisis lexocográfico, análisis sintáctico y generación de código, facilita la adaptación de la biblioteca a necesidades particulares.
- Realizar el analizador lexicográfico como una fase independiente facilita la realización del analizador sintáctico.
- De los analizadores sintácticos analizados, el LR brinda más facilidades para el proceso de generación de código intermedio. Además, exige el uso de una gramática con menos restricciones.
- Una forma interna, en la que predomine las características de la orientada a pila, hace más fácil el proceso de interpretación.
- El software libre es una solución para la realización de la biblioteca de clases puesto que ofrece facilidades de posible acceso al código.
- Java, como lenguaje de programación, que por sus características y popularidad es idóneo para la realización de la biblioteca de clases.
- De las metodologías estudiadas, XP puede ser empleada en el desarrollo de la biblioteca de clases.

Capítulo 2. Descripción de la biblioteca de clases para el reconocimiento y evaluación de expresiones.

Como respuesta al problema planteado en esta investigación, en este capítulo se describen las características de la construcción de la biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas, proceso que se realizó siguiendo las especificaciones de la metodología XP.

Se detallan las especificidades de las técnicas utilizadas en la construcción de cada una de las clases. Se presenta además un ejemplo práctico de aplicación de la biblioteca así como una valoración de esta por el criterio de expertos y un estudio de sostenibilidad con vistas a obtener una validación del producto.

Historias de usuarios.

Atendiendo al estudio de las principales técnicas para el reconocimiento y evaluación automático de cadenas realizado en el capítulo anterior, fueron identificados dos procesos fundamentales, analizar sintaxis e interpretar código, teniendo en cuenta que el resto de las fases analizadas en el capítulo anterior, tributan a la realización de alguna de estas. Por tanto, según la terminología de XP, estos constituyeron las dos historias de usuarios desarrolladas.

Historia de Usuario	
Número: 1	**Usuario:** Sistema Externo
Nombre historia: Analizar sintácticamente una expresión.	
Prioridad : Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 3	**Iteración asignada: 1**
Programador responsable: Programador	
Descripción:	

Recibe una expresión para verificar si pertenece al lenguaje generado por la gramática. Como resultado obtiene una representación interna de la expresión, con información referente a las variables correspondientes. Si detecta un error sintáctico emite un mensaje.
Observaciones: -

Historia de Usuario	
Número: 2	**Usuario:** Sistema Externo
Nombre historia: Evaluar expresión.	
Prioridad : Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 3	**Iteración asignada: 1**
Programador responsable: Programador	
Descripción: Recibe la representación interna de una expresión y la información de las variables que contiene. El usuario suministra valores a las variables. Seguidamente el sistema procede a obtener el resultado de la evaluación de la expresión en función de los valores suministrados.	
Observaciones: -	

Tareas para la historia 1.

Para analizar sintácticamente una expresión deben realizarse cuatro tareas fundamentales, realizar el análisis lexicográfico, aplicar el algoritmo LR para el reconocimiento de la sintaxis. Además, la generación de la representación interna de la expresión, proceso embebido en el análisis sintáctico de la

cadena, que incluye una representación de las variables con la información necesaria para el proceso de evaluación de la cadena.

En todas estas tareas interviene el uso de la tabla de símbolos, por ende su diseño y construcción es vital en este proceso. A continuación se describen las tareas mencionadas.

Tarea	
Número tarea: 1	**Número historia: 1**
Nombre tarea: Construir tabla de símbolos.	
Tipo de tarea : Desarrollo	**Puntos estimados: 2**
Programador responsable: Programador	
Descripción: En la tabla de símbolos será almacenada la información correspondiente a las variables y a las palabras claves definidas en el lenguaje.	

Tarea	
Número tarea: 2	**Número historia: 1**
Nombre tarea: Realizar análisis lexicográfico.	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Programador responsable: Programador	
Descripción: Determinar los componentes o partes válidas de la cadena a través de la lectura sucesiva de sus caracteres a partir de la definición de un autómata finito determinista para el lenguaje generado. Notifica al usuario en caso de error.	

Tarea	
Número tarea: 3	**Número historia: 1**
Nombre tarea: Analizar sintaxis	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Programador responsable: Programador	
Descripción: Utilizando al analizador lexicográfico, la tabla de transiciones y la tabla de símbolos, se determina si la cadena es sintácticamente correcta, en caso contrario se informa el error correspondiente.	

Tarea	
Número tarea: 4	**Número historia: 1**
Nombre tarea: Generar código	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Programador responsable: Programador	
Descripción: Se confecciona una representación intermedia de la cadena que se va analizando.	

Tareas de la historia 2.

La evaluación del código resultante del análisis sintáctico incluye también la realización de varias tareas. En esta historia, deben ser suministrados los valores a las variables detectadas y debe ser construido un intérprete de la representación interna obtenida que permita la obtención del resultado final de la expresión.

Tarea	
Número tarea: 1	**Número historia: 2**
Nombre tarea: Suministrar valores a las variables	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Programador responsable: Programador	
Descripción: Para cada una de las variables que aparecen en la cadena debe suministrarse un valor.	

Tarea	
Número tarea: 2	**Número historia: 2**
Nombre tarea: Calcular resultado	
Tipo de tarea : Desarrollo	**Puntos estimados: 3**
Programador responsable: Programador	
Descripción: Utilizando los valores de las variables, evaluar la expresión utilizando el intérprete que debe ser definido para obtener un resultado numérico. Notifica al usuario en caso de error.	

Diseño e implementación de las historias de usuarios.

Con el objetivo de apoyar la realización de la biblioteca y tener una representación visual del proceso de reconocimiento y evaluación de las expresiones, fueron construidos dos diagramas utilizando el Lenguaje de

Modelado Unificado (UML, por sus siglas en inglés)[39]. Un diagrama de secuencias, que muestra el flujo de información entre cada una de las fases del compilador, y un diagrama de clases, que contiene la estructura de las clases de la biblioteca. A continuación se muestran ambos diagramas.

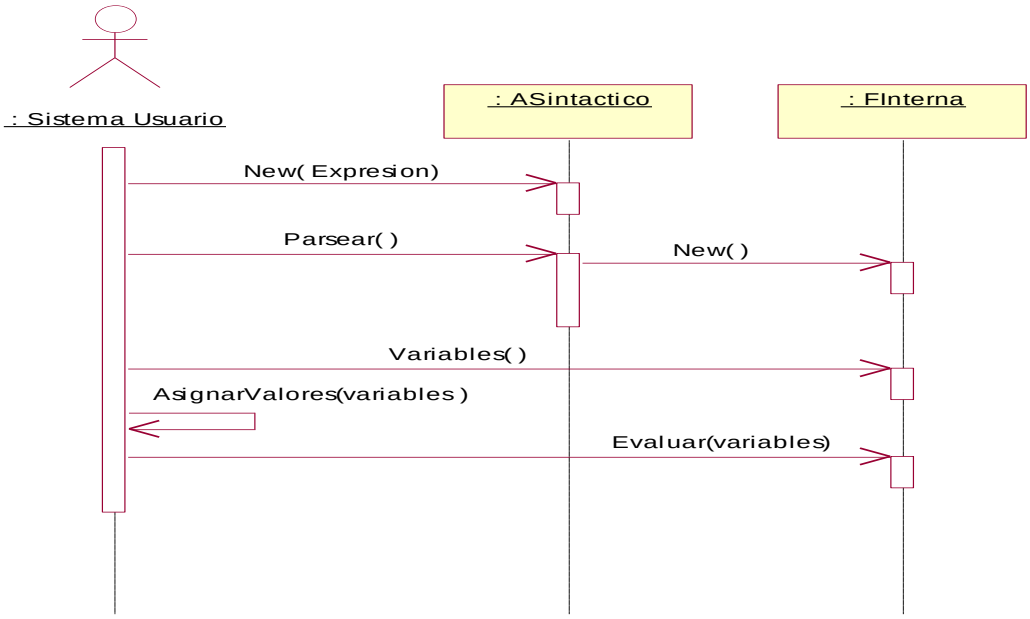


Fig. 2.1 Esquema general del trabajo de la biblioteca.

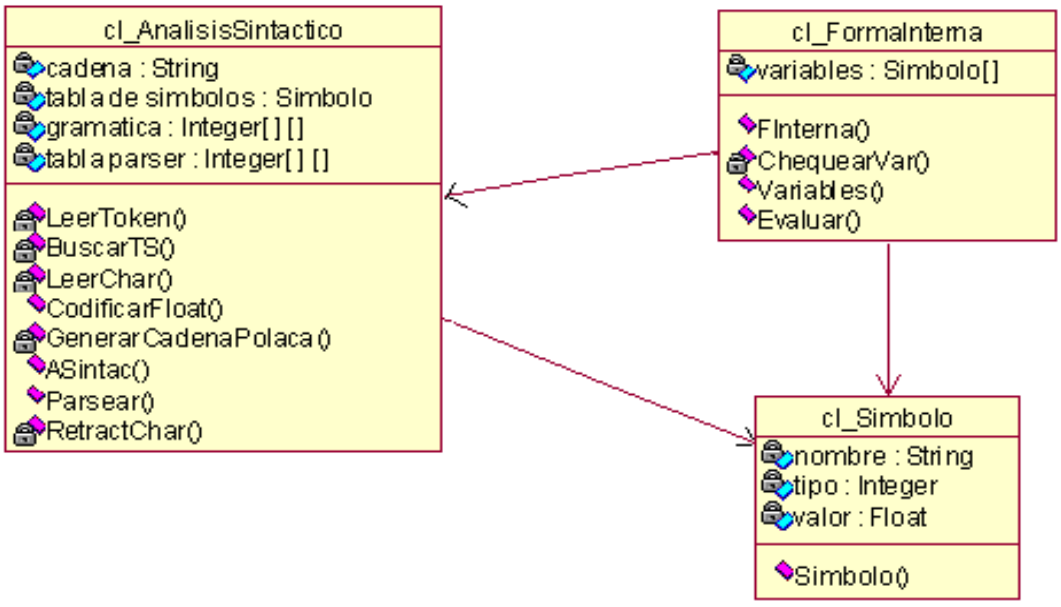


Fig 2.2 Diagrama de clases de la biblioteca.

[39] Rumbaugh, James. El lenguaje de modelado unificado.

Descripción de la clase ASintac.

Esta clase implementa la primera de las historias de usuario, encargada de realizar el análisis sintáctico de una expresión así como de crear una representación intermedia para su posterior evaluación. Cada uno de los métodos que se describen a continuación corresponden a la realización de las principales tareas trazadas con anterioridad para esta historia. Cada uno hace uso de otros métodos complementarios, necesarios para llevar a cabo cada responsabilidad.

Entre los atributos de esta clase están *cadena*, que alamcena la expresión a ser procesada.

Tabla de Símbolos, que al inicio del análisis contiene información sobre las palabras claves y luego almacena información sobre las variables.

Gramatica contiene la representación en un arreglo bidimensional de la gramática utilizada para generar el lenguaje. De acuerdo a la estructura de sus reglas clasifica entre las de tipo dependiente del contexto, no lineal a la derecha, y con reglas ambiguas. Ver anexo 4.

Tabla Parser contiene la tabla de transiciones, necesaria para la realización del analizador sintáctico, y fue construida manualmente a partir de la gramática del lenguaje.

Como consecuencia de la ambigüedad de la gramática, en el transcurso de la construcción de la tabla de transiciones se presentaron algunos conflictos del tipo *shift-reduce* lo que permitió afirmar que el analizador sintáctico elegido es del tipo LR[40].

Este tipo de conflictos estuvo relacionado con el tipo de asociatividad para algunos operadores, izquierda o derecha, situación que no puede ser resuelta solo a partir de las reglas de ninguna gramática[41]. En este caso, se decidió asociar siempre por la derecha, atendiendo al tipo de recursividad que presentan algunas de sus reglas, que sugieren la elección anterior. Ver tabla en anexo 5.

Como la tabla contiene dos entradas, una para los Tokens y otra para los estados, acceder a la acción a realizar implicó una codificación de los

[40] Parsons, Thomas. Introduction to compiler construction.
[41] Toro Bonilla, Niguel. Apuntes de compiladores.

elementos del analizador lexicográfico así como de las acciones. De esta forma, un número en la tabla negativo corresponde a un error sintáctico, un número entre 0 y 100 a la acción *shift,* y uno mayor que 100 a la acción *reduce.*

LeerToken()

Este método realiza la implementación del analizador lexicográfico. Previo a su construcción fue necesario la definición del alfabeto de lenguaje. Para ello se tuvo en cuenta que la representación para los operandos y operadores, así como la de los símbolos de agrupación, se correspondieran con la utilizada por el lenguaje java, atendiendo que el mayor número de usuarios de la biblioteca serán desarrolladores de aplicaciones en ese lenguaje. Ver anexo 2.

Con el objetivo de hacer más fácil la labor del analizador lexicográfico, se analizó que varios de los elementos válidos en el lenguaje brindaban la misma información sintáctica, por lo que no era necesario utilizar un símbolo o token para cada uno, sino que podían ser agrupados en determinados conjuntos atendiendo al criterio anterior. Esta agrupación garantizó la confección de una gramática con un menor número de producciones y por ende, un código más simplificado para el análisis.

De esta forma, el analizador lexicográfico construido devuelve en una estructura, también denominada token, dos tipos de información, el tipo o conjunto en el que se encuentra el elemento, y la subcadena correspondiente a él.

El primero de los dos valores es el más utilizado, y excepcionalmente el segundo, el cual es más necesario en el proceso de generación de la forma interna.

Como tokens válidos se identificaron los operadores aritméticos, los operadores relacionales, los operadores lógicos, los operadores aritméticos, las variables numéricas, las variables lógicas, las funciones y el paréntesis izquierdo y derecho. Además, se añadió el token especial $, que no pertenece al lenguaje definido, como indicador del final del procesamiento de esta.

En el caso de las variables, las palabras claves y las funciones, la estructura es la misma, una secuencia de letras, dígitos y el "_", que comienzan con una

letra. En algunos lenguajes se establecen restricciones sobre su longitud, fundamentalmente debido al espacio dedicado en memoria para guardar el nombre[42]. Este elemento no se tuvo en cuenta para la realización de este lenguaje, debido a que las cadenas son relativamente pequeñas y que la cantidad de variables o palabras claves es poco significativa.

Para diferenciar una variable lógica de una numérica, el usuario debe anteponer la letra l al nombre de esta. Esta distinción permitió eliminar ciertas ambigüedades en la construcción de la gramática, lo que facilitó la construcción de la tabla de transiciones.

A partir del autómata construido es imposible hacer la distinción entre una variable, de cualquier tipo y una función (ver anexo 3). Para ello se hizo necesario extender el uso de la tabla de símbolos a esta fase, de forma tal que en su contenido pudiera encontrarse la información necesaria para lograr la clasificación.

El analizador lexicográfico utiliza dos métodos auxiliares.

ReadChar(), el primero de ellos, garantiza la lectura de un caracter y posiciona el puntero en el próximo carácter a leer.

RetractChar() es utilizado para hacer retrocesos en la cadena. Esta situación se presenta cuando se ha procesado una secuencia de caracteres cuya estructura es válida para un token del lenguaje, pero a su vez esta es subcadena de un posible token, como en el caso de < y <=. Cuando se analiza el primero no puede concluirse con que es el operador relacional *menor*, sino que debe avanzarse en la cadena para analizar si forma parte del operador relacional *menor o igual*. En caso negativo, debe ser invocado este método para ubicar al puntero en el inicio del próximo *token* a analizar.

Por último, cabe señalar que cualquier modificación que decida hacerse al lenguaje, implicaría cambios mínimos solo en este método, lo cual es posible gracias a que fue concebido como una fase independiente.

[42] Katrib, Miguel. Lenguajes de programación y técnicas de compilación.

Parsear()

Este método es el más importante de la clase *Asintac*, debido a que es su responsabilidad la realización del análisis sintáctico de la cadena así como la generación de la forma interna.

Para ello hace uso de los demás métodos y de todos los atributos de la clase. Opera aplicando el método de análisis sintáctico LR y realiza además algunas verificaciones del tipo semánticas, que no son posibles realizar por el analizador sintáctico.

La principal información semántica analizada estuvo relacionada con las variables. Posterior a la realización del análisis sintáctico, elimina de la tabla de símbolos la información irrelevante para el proceso de evaluación de la cadena. En este caso todo lo referente a las palabras claves del lenguaje entre las que se incluyen las funciones.

Opera realizando las acciones que se especifican en la tabla de transiciones y la información que obtiene del analizador lexicográfico hasta procesar la cadena o detectar algún error. Utiliza además información referente a la estructura de las reglas de la gramática y dos pilas en las que almacena información producto de las acciones *shift* indicadas en la tabla, o extracciones provenientes de las acciones *reduce*.

GenerarCadenaPolaca()

Este método construye la forma interna para su posterior evaluación por un intérprete. Para su almacenamiento se utilizó como estructura de datos una pila de enteros.

La generación de esta forma interna estuvo asociada con la regla de la gramática que especifica la acción *reduce* de la tabla de transiciones, teniendo en cuenta que en el momeoto de la reducción ya fue procesada la parte derecha de esta.

Analizando la similitud en cuanto a estructura que tienen muchas reglas lo que está en correspondencia con la estructura del código que generan, a continuación se describe este proceso con las más significativas.

- <Expresión lógica> → <Expresión lógica> OR <Expresión lógica>: En este punto del análisis de la cadena ya se generó la forma interna para la expresión lógica de la izquierda y de la derecha del operador de disyunción. Queda entonces generar la forma interna para esta operación.
- <A3> → fun (Expresión aritmética): Cuando se indica reducir la parte derecha de esta regla es porque ya se ha generado la forma interna para la expresión aritmética. Solo queda generar el código para la función indicada. La información del tipo de la función no aparece reflejada en la regla, con el objetivo de lograr producciones más compactas. Esta información, del tipo semántica, puede ser encontrada en la tabla de símbolos.
- <A3> → num: En este caso se necesita generar la forma interna para un número, tal como lo indica la regla.

  Como la forma interna solo contendrá elementos enteros fue necesario descomponer los números reales en parte entera y decimal. Para ello fue construido el método *CofificarFloat()* que descompone un número real en parte entera y decimal.

  Pudo pensarse en almacenar todos los números en la tabla de símbolos, así un número, independientemente de la cantidad de veces que estuviera en la cadena de entrada, estaría almacenado una sola vez, pero esta alternativa obliga a realizar más búsquedas, considerando además que es bastante improbable que un número apareciera repetido varias veces en una expresión.

  <A3>→ var: En el caso de las variables, pueden almacenar valores enteros, reales o lógicos, en último caso 1 o 0. Esta información decidió almacenarse fuera de la memoria destinada a la forma interna, de manera tal que en el lugar correspondiente a su valor se encontrara un apuntador a su entrada en la tabla de símbolos. De esta forma, se facilita el proceso de cambio en cuanto a la estructura de las variables, posibilitando a los desarrolladores operar sobre otros tipos de datos con la consecuente redefinición de los operadores de la expresión.

Esta solución implicó el costo de extender el uso de esta hasta la fase de ejecución. No obstante, se tuvo en cuenta que una variable con distintos tipos implicaría contar con un operador para procesar cada tipo de variable y de mecanismos adicionales para evitar la confusión de un dato con un operando en la forma interna.

De esta manera, pudiera pensarse en la posibilidad de redefinir el tipo de las variables a tipos más complejos, sin necesidad de modificar la estructura de la forma interna lo cual haría más extensible el uso de la biblioteca.

Descripción de la clase FInterna

La construcción de esta clase fue mucho más sencilla que la anterior y representa la implementación de la segunda historia, evaluar cadena. Por ende, su responsabilidad radica en procesar la información obtenida en la realización del análisis sintáctico, y obtener un resultado numérico, producto de este proceso.

Cuenta con un único atributo, *variable*, en el que serán almacenados los nombres de las variables obtenidas en el proceso de análisis con sus correspondientes valores.

Variables()

Es un método que permite la publicación de los valores del atributo *variable* de la clase. Este método es necesario porque en este punto de la ejecución o evaluación de la cadena, no se tiene inofrmación de los valores que tomarán las variables, para la posterior obtención de un resultado. Esta información deberá ser sumisnistrada de alguna forma por el *sistema usuario.*

ChequearVar()

Es un método construido para chequear que se cuenta con la información necesaria para el proceso de evaluación de la cadena. En este caso, el valor de las variables. Notifica al sistema usuario en caso de un error.

Evaluar()

Constituye el método principal de esta clase. Tiene como responsabilidad la ejecución de la forma interna y contiene, por ende al intérprete.

Para llevar a cabo la interpretación de la forma interna, se hizo necesario contar con un *contador de programa* (cp), para informar en qué punto se está en la evaluación. Esto presupone un análisis riguroso pues pudiera introducir el problema de confundir un operador y un dato, tarea que fue delegada a cada uno de los operadores de manera que en todo momento, cp esté situado en el próximo operador a ejecutar.

La ejecución de la forma interna almacena en una pila los resultados intermedios de las operaciones. De esta forma, como consecuencia de la notación posfija, para cualquier operación los operandos ya han sido procesados y se encuentran en la estructura de datos antes mencionada. En ocasiones, estos resultados intermedios fueron almacenados en una variable evitando la necesidad de operar sobre la pila, y de esta forma la invocación de las subrutinas correspondientes. A continuación se presenta las características del código objeto:

Código objeto

- Operando1 or Operando2:

  R(Operando1), R(Operando2), OR

  OR: operador que toma los valores de sus operandos de la pila de ejecución y hace la operación disyunción lógica entre ellos, los elimina de la pila y apila el resultado. La operación se realiza de la siguiente forma:

  resultado = (operando1 + operando2) - (operando1 * operando2)

- Operando1 and Operando2:

  R(Operando1), R(Operando2), AND

  AND: operador que toma dos valores de la pila de ejecución y hace la operación conjunción lógica entre ellos, elimina de la pila los operandos y apila el resultado. La operación se realiza de la siguiente forma:

  resultado = operando1 * operando2

- Not Operando

  R(Operando), NOT

  NOT: operador que toma el valor que está en el tope de la pila de ejecución y hace la operación lógica de negación, sobrescribe el operando que está en el tope con el resultado de la operación. La operación se realiza de la siguiente forma:

  resultado = resto de la división entre (operando +1) y 2.

- Operando1 < Operando2

  R(Operando1), R(Operando2), MENOR

  MENOR: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 1 si el operando que estaba en el tope es mayor que el otro.

- Operando1 < Operando2

  R(Operando1), R(Operando2), MAYOR

  MAYOR: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 0 si el operando que estaba en el tope es mayor que el otro.

- Operando1 >= Operando2

  R(Operando1), R(Operando2), MAYORIGUAL

  MAYORIGUAL: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 0 si el operando que estaba en el tope es mayor que el otro.

- Operando1 <=Operando2

  R(Operando1), R(Operando2), MENORIGUAL

  MENORIGUAL: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 1 si el operando que estaba en el tope es mayor que el otro.

- Operando1 < > Operando2

  R(Operando1), R(Operando2), DISTINTO

  DISTINTO: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 1 los operandos son diferentes.

- Operando1 = = Operando2

  R(Operando1), R(Operando2), IGUAL

IGUAL: operador que toma dos valores de la pila de ejecución y compara sus valores, elimina de la pila los operandos y un 1 los operandos son iguales.

- Variable lógica

  VL, dirección en la tabla de variables de esa variable.

- VL: operador que toma el valor que aparece a continuación, es decir, la dirección que se especifica, y busca el valor de la variable lógica en la tabla de símbolos. Apila ese valor en la pila de ejecución.

- Operando1 + Operando2

  R(Operando1), R(Operando2), SUM

  SUM: operador que toma dos valores de la pila de ejecución y los adiciona, elimina de la pila los operandos y apila el resultado.

- Operando1 - Operando2

  R(Operando1), R(Operando2), RES

  RES: operador que toma dos valores de la pila de ejecución y le resta el que está en el tope del otro operando, los elimina de la pila y apila el resultado.

- Operando1 < Operando2

  R(Operando1), R(Operando2), MULT

  MULT: operador que toma dos valores de la pila de ejecución y los multiplica, elimina de la pila los operandos y apila el resultado.

- Operando1 / Operando2

  R(Operando1), R(Operando2), MENOR

  DIV: operador que toma dos valores de la pila de ejecución y divide por el que está en el tope al otro operando, los elimina de la pila y apila el resultado.

- Operando1 ^ Operando2

  R(Operando1), R(Operando2), EXP

  EXP: operador que toma dos valores de la pila de ejecución y eleva uno de los operandos por el que está en el tope, los elimina de la pila y apila el resultado.

- - Operando

  R(Operando),MIN

MIN: operador de dos argumentos que forman un número. Multiplica ese valor por -1 y apila el resultado en la pila de ejecución. Los argumentos se encuentran a continuación de él en la forma interna.

- *sin* Operando

  R(Operando), SEN

  SEN: operador que toma el valor que está en el tope de la pila de ejecución y calcula su seno, sobrescribe el operando que está en el tope con el resultado de la operación.

- cos Operando

  R(Operando), COS

  COS: operador que toma el valor que está en el tope de la pila de ejecución y calcula su coseno, sobrescribe el operando que está en el tope con el resultado de la operación.

- tan Operando

  R(Operando), TAN

  TAN: operador que toma el valor que está en el tope de la pila de ejecución y calcula su tangente, sobrescribe el operando que está en el tope con el resultado de la operación.

- *cotan* Operando

  R(Operando), COTAN

  COTAN: operador que toma el valor que está en el tope de la pila de ejecución y calcula su cotangente, sobrescribe el operando que está en el tope con el resultado de la operación.

- *arctan* Operando

  R(Operando), ARCTAN

  ARCTAN: operador que toma el valor que está en el tope de la pila de ejecución y calcula su arcotangente, sobrescribe el operando que está en el tope con el resultado de la operación.

- *arccot* Operando

  R(Operando), ARCCOT

  ARCCOT: operador que toma el valor que está en el tope de la pila de ejecución y calcula su arcocotangente, sobrescribe el operando que está en el tope con el resultado de la operación.

- *arcos* Operando

  R(Operando), ARCOS

  ARCOS: operador que toma el valor que está en el tope de la pila de ejecución y calcula su arcocoseno, sobrescribe el operando que está en el tope con el resultado de la operación.

- *arcsen* Operando

  R(Operando), ARCSEN

  ARCSEN: operador que toma el valor que está en el tope de la pila de ejecución y calcula su arcoseno, sobrescribe el operando que está en el tope con el resultado de la operación.

- Variable numérica

  VN, dirección en la tabla de variables de esa variable.

  VN: operador que toma el valor que aparece a continuación, es decir, la dirección que se especifica, y busca el valor de la variable lógica en la tabla de símbolos. Apila ese valor en la pila de ejecución.

- Número

  NUM, parte entera, parte real.

  NUM: operador de dos argumentos que forman un número. Forma a partir de ellos un número real y lo apila en la pila de ejecución.

Laboratorio Químico Central de la Empresa de Níquel "Cmdte Ernesto Che Guevara". Una aplicación práctica de la biblioteca de clases.

La Empresa de Níquel "Comandante Ernesto Che Guevara" está enclavada en el municipio holguinero de Moa. Perteneciente al Ministerio de la Industria Básica (MINBAS), tiene como principal objetivo la producción de níquel y cobalto para su exportación hacia el mercado internacional.

El Grupo de Producción de Software (GPROSOFT) de la Universidad de Holguín "Oscar Lucero Moya" desarrolló e implantó en la Empresa de Níquel "Comandante Ernesto Che Guevara" de Moa, el Portal Corporativo CheNET,

cuyo núcleo de funcionamiento es la Gestión de Procesos, destinado al control de la actividad productiva de la Empresa[43].

En el control de las actividades de la Empresa se miden un gran número de indicadores[44]. El análisis de los resultados de estos indicadores se realiza a partir de la evaluación de la expresión matemática que lo describe, resultados que aportan un conjunto de información a los directivos que les permite tomar determinadas decisiones.

Estos indicadores suelen depender unos de los otros, así como de algunas otras variables adicionales, que incluyen tanto factores, como muestras de mineral que se analizan dentro de la propia entidad. De aquí se deriva que se maneje una gran cantidad de fórmulas que se complementan para obtener resultados intermedios y finales necesarios para garantizar el control productivo.

En ocasiones, por diversas causas, puede variar el modo de cálculo de un determinado indicador, o sea, variar la fórmula que se utiliza para determinarlo. De igual modo y con mayor frecuencia que el caso anterior, puede surgir uno nuevo y con ello una nueva fórmula asociada al mismo.

En tales circunstancias, el Portal Corporativo CheNET desarrollado se concibió para que en estos casos se acceda directamente al código y se reprogramen las consultas destinadas a captar los valores necesarios para los cálculos. Esta solución, aunque efectiva, implica una disponibilidad de programadores, lo cual en ocasiones no es posible y consecuentemente, hace más lenta la adaptación del sistema al cambio.

Ante tal situación, se propuso a los desarrolladores del Portal la opción de contar con una biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas, como un módulo de apoyo destinado al control dinámico de las fórmulas, como una herramienta que permita el trabajo dinámico con fórmulas.

En la empresa existen tres tipos de procesos fundamentales: los productivos, destinados a la ejecución y control de las actividades de producción; los

---

[43] Menéndez Mora, Raúl. CheNET: Portal corporativo de la empresa de níquel "Comandante Ernesto Che Guevara".

[44] Pérez Campaña, Marisol. Modelo y procedimientos para organizaciones comericalizadoras.

procesos de apoyo, que persiguen ayudar al desarrollo de los procesos productivos, en cuanto a la gestión de insumos fundamentales respecta y los procesos de gestión de venta, que están encaminados a la comercialización de los productos finales.

El ciclo productivo, como el más importante, es realizado a través de los siguientes subprocesos:

1. Extracción del mineral.
2. Preparación del mineral.
3. Reducción del mineral.
4. Lixiviación y lavado del mineral.
5. Separación del cobalto contenido en el mineral.
6. Recuperación de Amoníaco, uno de los insumos fundamentales de los procesos químicos.
7. Calcinación y Sínter del mineral para obtener los productos finales de la Fábrica.

Entre ellos existe una dependencia muy estrecha, y uno da paso a la realización de otro hasta lograr finalmente la obtención de los productos finales de la fábrica para su posterior comercialización.

Como en todo proceso productivo, la calidad es un factor prioritario al que se dedica una gran cantidad de esfuerzo. De esta forma, cada uno de los subprocesos está constantemente sometido a pruebas con vistas a determinar el nivel de calidad en un momento. Con tal propósito, existe en la Empresa un laboratorio denominado Laboratorio Químico Central (LQC) donde son realizados análisis físico-químicos del mineral, en cada una de sus fases.

El resultado de un análisis, o la combinación de varios de estos, permite medir ciertos indicadores, con vistas a obtener una información parcial de la calidad del proceso productivo, resultado del que dependerá la toma de decisiones acertadas que garanticen los resultados planificados al inicio del proceso.

Existen dos tipos esenciales de reporte: metalúrgico y operativo. Estos se generan con frecuencias distintas: cada 2, 4, 6, 12 y 24 horas, así como una especial que refiere el resumen del mes hasta el momento en que se solicita.

Como puede constatarse, esta operación se realiza con mucha frecuencia y es facilitada considerablemente por los servicios que brinda el Portal, atendiendo a la rapidez con que permite acceder a esta información y a la forma en que es presentada. Ello implica que cualquier imposibilidad para usar estos servicios implicaría el uso de variantes alternativas menos eficientes, lo que repercutiría en la rapidez y calidad con que se obtiene la información.

El Portal, actualmente, brinda estos resultados en un reporte, pero no permite incluir de manera automática otro indicador ni modificar uno existente. Esta funcionalidad pudo haberse implementado con antelación, pero se analizó que no tenía sentido prever el cambio de la configuración de un reporte de forma automática, sin que pudiera gestionarse la fórmula para un nuevo indicador, o la modificación de alguno de los existentes de ese mismo modo.

Con la utilización de la biblioteca de clases para el reconocimiento y la evaluación de expresiones, se crea la posibilidad de extender el subsistema de Gestión de Reportes, de manera que sea posible añadir, modificar y ocultar indicadores de forma automatizada. De esta forma, se dota al Subsistema de Gestión de Reportes y por extensión al Sistema de Gestión de Procesos y al Portal en sentido general, de una herramienta más para evolucionar y poder enfrentar cambios tecnológicos y/o de los procesos productivos.

Una vez analizada esta situación se propuso la inclusión en el sistema de los siguientes requerimientos funcionales:

R1. Aprobar actualización de indicador de reporte.

R2. Actualizar indicador de reporte.

R3. Visualizar indicador de reporte.

R4. Actualizar propuesta de fórmula.

R5. Visualizar propuesta de fórmula.

R6. Evaluar propuesta de fórmula.

R7. Enviar mensaje.

R8. Actualizar fórmula de cálculo.

R9. Validar fórmula de cálculo.

R10. Aprobar actualización de condición de cálculo.

R11. Actualizar condición de cálculo.

R12. Validar condición de cálculo.

R13. Visualizar condición de cálculo por indicador de reporte.

R14. Visualizar fórmula de cálculo por indicador de reporte.

R15. Visualizar variable.

Diagrama de casos de uso del subsistema Fórmula LQC.

Para la realización de este subsitema se identificaron 15 casos de uso, que al igual que los requerimientos funcionales fueron sometidos a la valoración de los desarrolladores del Portal. Se tuvo en cuenta la seguridad del sistema, por lo que fue necesario el estudio de los diferentes niveles de acceso a este submódulo. Ver detalles en anexo 7.

Entre los actores del sistema que interactuarían con este sistema se encuentran:

- *TecnólogoLQC:* Representa al Tecnólogo de Laboratorio, encargado de realizar la propuesta para la actualización de una fórmula.
- *TecnólogoCM:* Representa al Tecnólogo de Contabilidad Metalúrgica, encargado de evaluar la actualización de una fórmula en caso de que considere necesario. Determina y aprueba la actualización de indicadores de reportes.
- *AdminGTI:* Representa al miembro del Grupo de Tecnología de la Información que se encargará de actualizar las fórmulas.
- *Visualizador:* Representa a los diferentes usuarios que visualizarán fórmulas, condiciones de cálculo, indicadores, factores y muestras.
- *Usuario:* Representa a todos los usuarios del sistema.
- *Destinatario:* Representa a los diferentes usuarios a los que se les enviará un correo electrónico de notificación.
- *Administrador:* Representa al administrador de seguridad del sistema.

Modelo de casos de uso del subsistema Laboratorio Químico Central.

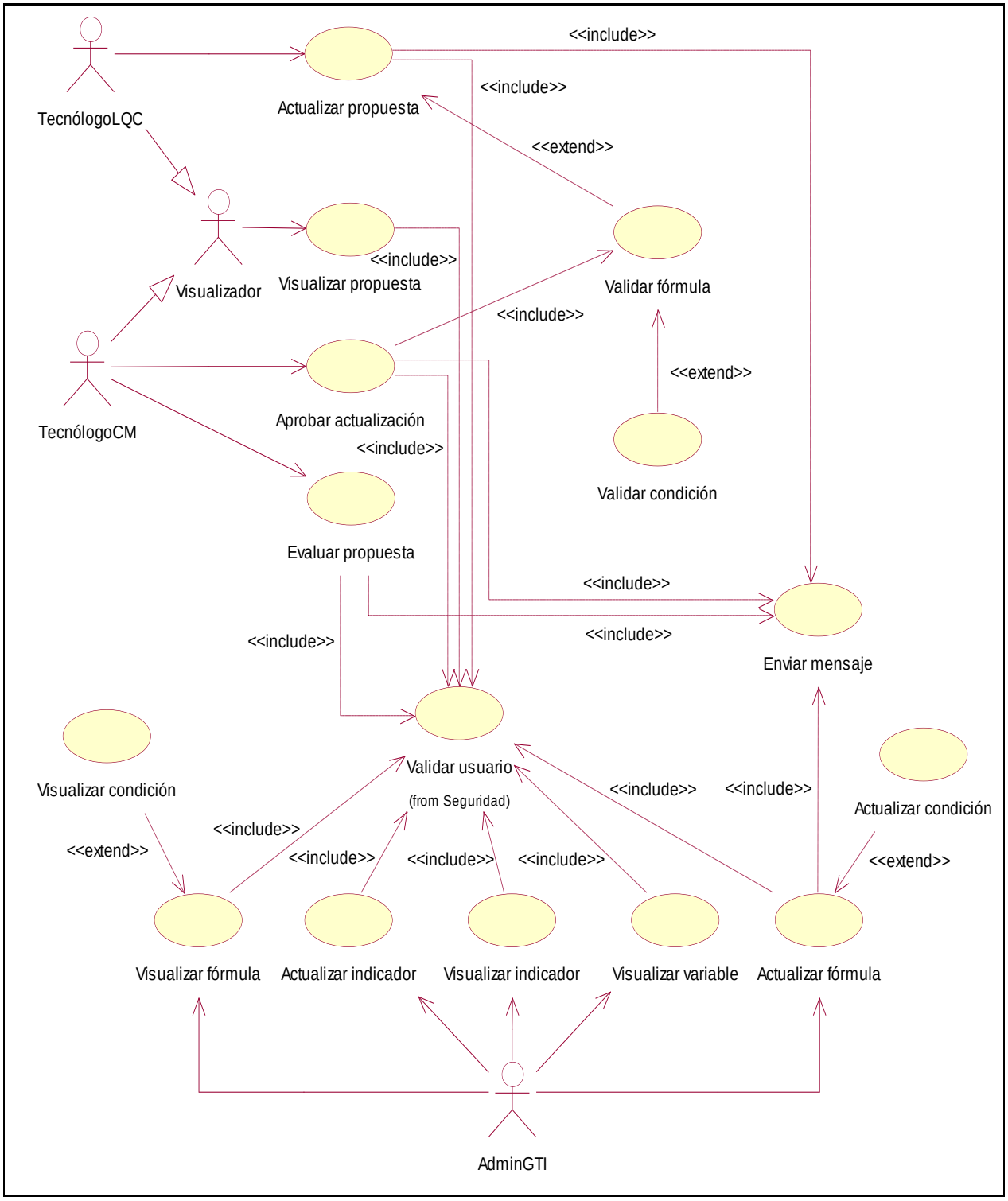


Fig. 2.3 Diagrama de casos de uso del sistema.

Impacto Económico–Social del producto informático creado.

Desde hace varias décadas, la informática ha estado formando parte indisoluble de la vida de los seres humanos. De ahí que su influencia en muchos de los comportamientos y procederes sea cada vez más significativa, tanto a nivel de individuo como de la sociedad misma. Ello presupone que en todo momento de la creación de un producto informático deba hacerse un estudio de los efectos que traerá aparejado su uso, atendiendo a su impacto desde el punto de vista económico, ambiental, tecnológico así como sociopolítico.

Este proceso no termina nunca, teniendo en cuenta que el entorno no es estático y que constantemente surgen amenazas contra la vida de los sistemas, a las que este deberá enfrentar de la mejor manera posible.

Los sistemas informáticos que requieran la modificación de las expresiones aritméticas, relacionales o lógicas, definidas de forma estática, necesitan modificar el código de la aplicación para reflejar algún cambio en la estructura de estas, lo cual pudiera ser muy costoso y además, constituye una solución un tanto engorrosa, como se planteó en el epígrafe anterior.

Con el uso de la biblioteca en una aplicación de este tipo, no solo se logra un sistema más adaptable al cambio sino que se disminuye la cantidad de horas dedicadas al desarrollo de la aplicación, debido a que el desarrollador se ahorra la necesidad de modificar el código constantemente o de crear un módulo para reconocer la sintaxis de una expresión así como su evaluación, procesos que como se ha señalado con anterioridad, son bastante complejos y engorrosos. De esta forma, se logra una mejor adaptación del sistema a las necesidades de los usuarios.

Cabe destacar, que la biblioteca no generará ingresos de forma directa, pero sí la posibilidad de crear sistemas de una mayor calidad, capaces de gestionar información que antes les era imposible utilizar. De esta manera, el desarrollador podrá centrarse en los requerimientos propios del sistema.

Para el uso de la biblioteca, el equipo de desarrollo no tendrá que mejorar el equipamiento que dispone ya que la misma se incorporaría a su ambiente de

desarrollo (IDE, Integrated Development Enviroment) como una librería más del mismo.

Al disminuir la cantidad de horas-máquina en el desarrollo de una aplicación, el consumo de electricidad disminuye. Las herramientas técnicas utilizadas para el desarrollo de la biblioteca son distribuidas como software libre o de manera gratuita, por lo que no se necesitó invertir en software para su construcción y de igual manera, los usuarios no tendrán que invertir en el pago de licencias para su uso.

Con la implantación del sistema no se afecta la plantilla de los equipos de desarrollo, sino que se facilita el trabajo de las personas involucradas en la creación de software, aspecto importante en las tendencias de desempleo que devienen con la implantación de numerosos sistemas informáticos.

La utilización de la biblioteca no ocasiona daños al medio ambiente. La estructura de las clases y las técnicas utilizadas para su construcción, permiten su fácil utilización e inclusión de nuevas funcionalidades.

Valoración del producto por criterio de expertos.

Todo producto informático debe ser objeto de ciertas pruebas que aseveren su importancia en el entorno en el cual será aplicado. En este sentido, la biblioteca de clases que se propone en esta investigación, fue sometida a una valoración por criterio de expertos, con vistas a obtener opiniones más representativas sobre las facilidades de construcción y adaptación que puede brindar a los programadores en la construcción de sistemas informáticos que requieran el procesamiento y evaluación de expresiones del tipo aritméticas, relacionales y lógicas.

De esta forma, auxiliándose del análisis realizado en el epígrafe anterior, se confeccionó una encuesta para determinar cuán útil sería el uso de la biblioteca de clases en el Portal Corporativo CheNet de la Fábrica de Níquel antes mencionada. Para ello se indicó clasificar las funcionalidades, atendiendo a la relevancia de la biblioteca para lograr su inclusión en el sistema, en totalmente de acuerdo (TA), muy de acuerdo (MA), de acuerdo (A), parcialmente de acuerdo (PA) y en desacuerdo (D). Ver anexo 1.

Para su aplicación fueron seleccionados 30 programadores con un promedio de 6 años de experiencia en el desarrollo de productos informáticos con las características antes señaladas, teniendo en cuenta su grado de experticia[45].

El procesamiento de la encuesta arrojó los siguientes resultados: (ver detalles en anexo 6):

CONCLUSIONES GENERALES					
Aspectos	TA	MA	A	PA	D
A 1	Si	-	-	-	-
A 2	-	Si	-	-	-
A 3	Si	-	-	-	-
A 4	Si	-	-	-	-

Ello constituye una evidencia de que la hipótesis de la que se partió es válida.

Conclusiones del capítulo

- La manera en que se estructuró la biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas facilita su uso y adaptación.
- A través de su uso en el Portal Corporativo CheNet se demostró que la biblioteca de clases es una herramienta que puede facilitar el desarrollo de sistemas informáticos que trabajen con expresiones aritméticas, relacionales y lógicas.
- Se demostró que el producto desarrollado es sostenible desde las dimensiones administrativa, socio-humanista, ambiental y tecnológica.

[45] Rodríguez Expósito, Félix; Concepción García, Rita. El Método Delphy Para el Procesamiento de los Resultados de Encuestas a Expertos o Usuarios en Estudios de Mercado y en la Investigación Educacional

Conclusiones

Como resultado de la investigación se arribó a las siguientes conclusiones:

1. El procesamiento y evaluación de expresiones están presentes en el proceso de toma de decisiones que tiene lugar en las empresas, influyendo directamente en sus resultados y niveles de competitividad. Debido a las ventajas que las NTIC brindan a la forma en que los directivos gestionan la información, contar con herramientas automatizadas que faciliten estas tareas es fundamental.
2. Los lenguajes de desarrollo actuales carecen de mecanismos eficientes que faciliten el procesamiento y evaluación de expresiones aritméticas, relacionales y lógicas. Además, su realización requiere del uso de técnicas de programación de una elevada complejidad.
3. La inclusión en los lenguajes de programación de herramientas para el procesamiento y evaluación de expresiones aritméticas, relacionales y lógicas facilita a los desarrolladores la realización de sistemas informáticos para la toma de decisiones en un menor tiempo.
4. La biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas presentada en este trabajo facilita el desarrollo de aplicaciones informáticas de ayuda a la toma de decisiones, permitiendo una mejor adaptación a los cambios del entorno así como una forma más eficiente de procesar la información.

Recomendaciones

1. Fomentar el uso de la biblioteca en la realización de sistemas informáticos que requieran del análisis y la evaluación de con expresiones aritméticas, relacionales y lógicas.
2. Identificar nuevas áreas que requieran del procesamiento y evaluación de expresiones aritméticas, relacionales y lógicas.
3. Incluir en la biblioteca nuevas funcionalidades que permitan una adaptación más fácil a necesidades particulares de los desarrolladores.
4. Continuar la divulgación de las experiencias y resultados obtenidos en el trabajo de investigación.
5. Continuar el perfeccionamiento de la biblioteca a través del estudio de experiencias similares y criterios de expertos.

Bibliografía

- Aaby, Anthony A. Compiler Construction using Flex and Bison.[documento en línea]ftp://serverinfo.uho.edu.cu/Docs/Programación/Teoría de Compiladores/Compiler Construction using FLex and Bison/Compiler.rar [consultado, 13 de abr. 2006]
- ADESA. [Documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/ ADESA/ADESA.rar [consultado: 3 de sept. 2006]
- ADOOSI. [Documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/ ADOOSI/ADOOSI.rar [consultado: 4 de sept. 2006]
- Agile development in Small and Medium Size Proyects. [Documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/agile/agile development in small and medium size proyects.rar [consultado: 12 de sep. 2006]
- Aguilera, María del Mar. Traductores, Compiladores e Intérpretes. [documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Programación/Teoría de Compiladores/Traductores.rar. [Consultado: 20 de mar. 2006]
- Ceballos Carmona, Miguel Angel. Compiladores. [Documento en línea]. http://www,monografias.com/trabajos11/compil/compil.shtml [consultado: 20 de mar. 2006]
- Concepción, Rita. Procedimiento para la valoración de sostenibilidad de un PI. [documento digital]
- Conejo, Ernesto. Compiladores. [Documento en línea] http://www.lcc.uma.es/docencia/ETSIInf/pl/pl1.html [consultado: 7 de abr. 2006]
- Conejo, Ricardo. Funcionamiento de un analizador lexicográfico. [documento en línea] http://www.lcc.uma.es/docencia/ETSIInf/pl/apuntes/pl1/lecc21.html [consultado: 20 de mar. 2006]
- Cueva, Lovelle, Juan M. Conceptos básicos de procesadores de lenguajes. [documento en línea] http://di002.edv.uniovi.es/procesadores/apuntes/10_conceptos.pdf [consultado: 20 de mar. 2006]
- Cueva, Lovelle, Juan M. Análisis léxico en procesadores de lenguajes. [documento en línea]

ftp://serverinfo.uho.edu.cu/Docs/Programación/Teoría de Compiladores/Compilers.rar [consultado: 23 de mar. 2006]

- Design Patterns Java WorkBook. [documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Programación/Java/Design Patterns Java WorkBook.rar [consultado: 15 de feb.2006]
- Estrategia para el uso del Software Libre en Cuba (Nov-2002). Disponible en http://www.linux.cu/ [Fecha de consulta 3 de abril de 2006]
- Free Software Foundation Web Site. [documento en línea] http://www.fsf.org [consultado 30 de mar. 2006].
- García Carballeira, Félix. Arquitectura de la Máquina Virtual Java [documneto en línea] http://www.revista.unam.mx. [consultado: 3 de abr. 2006]
- JACOBSON, Ivar. "El Proceso Unificado de Desarrollo de Software".[documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/Pressman/ Pressman 5ta ed.rar [consultado: 10 de sep. 2006]
- Java Developer's Reference. [documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Programación/Java/Java Developer's Reference.rar [consultado: 15 de feb.2006]
- Katrib, Miguel. *Lenguajes de programación y técnicas de compilación*. La Habana: Editorial Pueblo y Educación, 1889.
- Lee, James. Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP. [document en línea] ftp://serverinfo.uho.edu.cu/Docs/Articulos varios/opensource.rar [consultado: 13 de feb. 2006].
- Lindholm T., Yellin F. The JavaTM Virtual Machine Specification. 2nd Edition.[documento en línea] http://java.sun.com/docs/books/vmspec/2nd edition/html/VMSpecTOC.doc.html. [consultado: 30 de may. 2006].
- Mas i Hernández, Jordi. Software Libre: Técnicamente viable, económicamente sostenible y económicamente justo. [documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Programación/Java/adobe.rar

[consultado: 30 de may. 2006]

- Metodología OMT. [documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Ingenieríadesoftware/Metodologías/MetodologíaOMT.rar [consultado: 20 de sep, 2006]
- Métodos Heterodoxos en Desarrollo de Software. [Documento digital]
- Menéndez Mora, Raúl. CheNET: Portal corporativo de la empresa de níquel "Comandante Ernesto Che Guevara". [documento digital]
- Modelación de la investigación científica. [documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Modelación de la investigacion/Modelación de la investigación científica.pdf [Consultado: 14 de sep, 2006]
- Parsons, Thomas. Introduction to Compiler Construction.1991[document digital]
- Planning Extreme Programming. [Documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/Extreme Programming/ Planning Extreme Programming.rar [consultado: 13 de sep. 2006]
- Pérez Campaña, Marisol. Modelo y procedimientos para organizaciones comericalizadoras. [documento digital].
- Pocalles, Josep. ¿Cuáles son las ventajas del software libre para las PYMES? [documento en línea] http://winred.com/EP/articulos/tecnologia/a1886.html [consultado: 6 abril 2005].
- Portal de Proyectos Open Source. [documento en línea] http://sourceforge.net [consultado: 10 de ene. 2006].
- Rodríguez Expósito, Félix; Concepción García, Rita. El Método Delphy Para el Procesamiento de los Resultados de Encuestas a Expertos o Usuarios en Estudios de Mercado y en la Investigación Educacional [documento digital]
- Riels, Hanne. Semantics with applications. [documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Programación/Teoría de Compiladores/Semantics with applications/wiley.rar [consultado: 15 de abr. 2006]
- Rumbaugh, James. Guía de la notación del UML. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/ UML/UML Biblia.rar [consultado: 17 de sep. 2006]

- Rumbaugh, James. El lenguaje de modelado unificado. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/ UML/El lenguaje unificado de modelado.rar [consultado: 17 de sep. 2006]
- RUP Small. [Documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/ Rational/RUPSmall.rar [consultado: 14 de sep. 2006]
- Thinking in Java. [documento en línea] ftp://serverinfo.uho.edu.cu/Docs/Programación/Java/Thinking in java.rar [consultado: 15 de feb.2006]
- Sierra Lombardía, Virginia. Metodología de la investigación científica. [documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Metodología de la investigacion/Folleto de met invest.rar [Consultado: 13 de sep, 2006]
- Toro Bonilla, Miguel. Apuntes de compiladores. [documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Programación/Teoría de Compiladores/Apuntes de compiladores.rar. [Consultado: 24 mar. 2006]
- Wake, Willliam. Extreme Programming Explained. [Documento en línea]. ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/Extreme Programming/ Extreme Programming Explainned.rar [consultado: 12 de sep. 2006]
- Wake, Willliam. Extreme Programming Explored. [Documento en línea].
- ftp://serverinfo.uho.edu.cu/Docs/Ingeniería de software/Extreme Programming/ Extreme Programming Explored.rar [consultado: 12 de sep. 2006]

Anexos

Anexo 1. Resultados de la aplicación del método de expertos para validar los resultados de la investigación.

Fuente: Elaboración propia.

A: _____

Como parte de la realización de una tesis de maestría en Matemática Aplicada e Informática para la Administración, se propone el uso de una biblioteca de clases para el reconocimiento y evaluación de expresiones aritméticas, relacionales y lógicas con el objetivo de facilitar el desarrollo y adaptación de aplicaciones informáticas que requieran esta funcionalidad.

En tal sentido, se adjunta a este documento un ejemplo práctico de su aplicación en el proceso de generación de reportes de indicadores del Portal CheNet, perteneciente a la Empresa del Níquel Ernesto Ché Guevara.

Por favor, clasifique su posición con respecto a cada uno de los aspectos que se relacionan en la tabla siguiente en:

- **TA:** Totalmente de acuerdo.
- **MA:** Muy de acuerdo.
- **A:** De Acuerdo.
- **PA:** Parcialmente de acuerdo.
- **D:** En desacuerdo.

No	Aspectos	TA	MA	A	PA	D
1	Con el uso de la biblioteca de clases en sistemas informáticos que operen con fórmulas matemáticas propensas al cambio, se aumenta la capacidad de estos para adaptarse a las variaciones que les impone su entorno.					
2	Prever la modificación de fórmulas en sistemas informáticos que operen con definiciones dinámicas de estas, a través de cambios en su código, es una solución que atenta contra la rapidez					

	para dar respuesta a cambios en su entorno.					
3	Para un programador, el desarrollo de las funcionalidades que brinda la biblioteca es una tarea compleja.					
4	Con la biblioteca de clases se disminuye el tiempo de desarrollo de sistemas informáticos que requieran el procesamiento y evaluación dinámico de expresiones.					

Anexo 2. Alfabeto del lenguaje para expresiones aritméticas, relacionales y lógicas.

En el lenguaje, las variables tienen la estructura de un identificador, una secuencia de letras, dígitos o el carácter "_" que comienza con una letra.

Para la representación de los operadores aritméticos se utilizó la notación clásica:

+ : operación de adición.

- : operación de sustracción.

* : operación de multiplicación.

/ : operación de división.

Los operadores relacionales, por su parte, fueron representados de la siguiente forma:

= = : denota igualdad.

!= : denota desigualdad.

< : menor.

> : mayor.

<= : menor o igual.

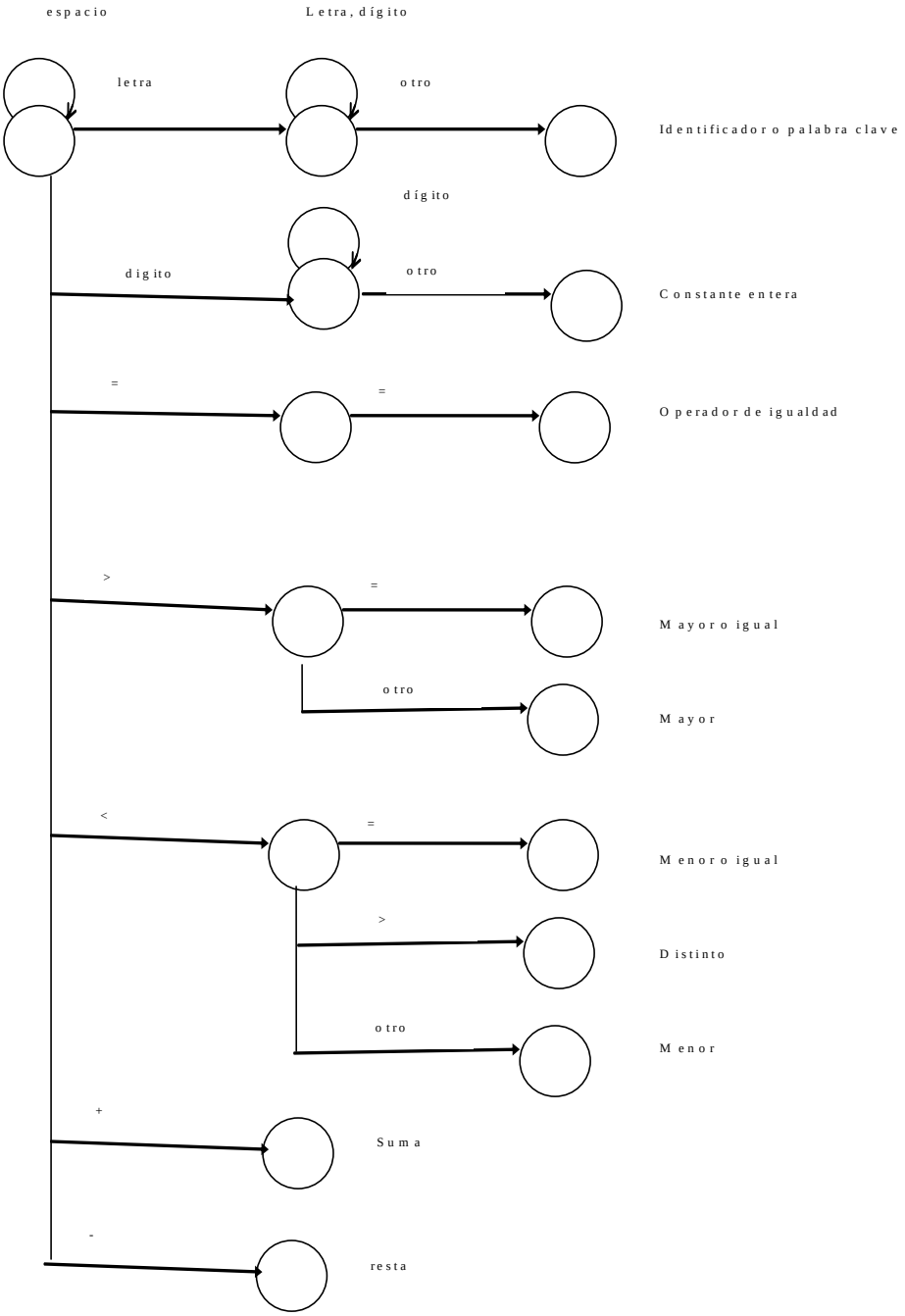
>= : mayor o igual

Los operadores lógicos serán denotados con los símbolos AND, OR y NOT que denotan la conjunción, disyunción y negación respectivamente.

Para los símbolos de agrupación fueron utilizados los paréntesis

- Símbolos de agrupación. (,)
- Funciones.
    - Aritméticas:
    - Trigonométricas:

Anexo 3. Autómata finito.(Aquí va un diagrama de estados UML)



Anexo4. Gramática del lenguaje.

<Expresión> → <Expresión aritmética> |<Expresión lógica>

<Expresión lógica> → <Expresión lógica> OR <Expresión lógica> |<L1>

<L1> → <L1> AND <L1> |<L2>

<L2> → NOT <L2> |(<Expresión lógica>) | vl | < Expresión relacional >

<Expresión relacional> → <Expresión aritmética> OPR <Expresión aritmética>

<Expresión aritmética> → <Expresión aritmética> + <Expresión aritmética> |

<Expresión aritmética> - <Expresión aritmética> |

<A1>

<A1> → <A1> * <A1> | <A1> / <A1> |A2

<A2> → <A2> ^ <A2> |<A3>

<A3> → FUN (Expresión aritmética) | vn | - vn | número | - número

Anexo 5. Tabla de acciones.

NO	OR	AND	NOT	(	)	VL	OPR	+	-	*	/	^	FUN	VN	NUM	$	E	L	L1	L2	A	A1	A2	A3
0			S14	S7		S15			S9				S6	S8	S10		1	11	12	13	2	3	4	5
1																ACC								
2							S18	S16	S17							R2								
3	R14	R14			R14		R14	R14	R14	S19	S20					R14								
4	R17	R17			R17		R17	R17	R17	R17	R17	S21				R17								
5	R19	R19			R19		R19	R19	R19	R19	R19	R19				R19								
6				S22																				
7			S14	S7		S15			S9				S6	S8	S10			23	12	13	24	3	4	5
8	R22	R22			R22		R22	R22	R22	R22	R22	R22				R22								
9														S25	S26									
10	R24	R24			R24		R24	R24	R24	R24	R24	R24				R24								
11	S27															R3								
12	R5	S28			R5											R5								
13	R7	R7			R7											R7								
14			S14	S7		S15			S9				S6	S8	S10					29	41	3	4	5
15	R10	R10			R10											R10								
16				S31					S9				S6	S8	S10						30	3	4	5

17				S31					S9				S6	S8	S10						32	3	4	5
18				S31					S9				S6	S8	S10						33	3	4	5
19				S31					S9				S6	S8	S10							34	4	5
20				S31					S9				S6	S8	S10							35	4	5
21				S31					S9				S6	S8	S10								36	5
22				S31					S9				S6	S8	S10						37	3	4	5
23	S27				S38																			
24					S39		S18	S16	S17															
25	R23	R23			R23		R23	R23	R23	R23	R23	R23				R23								
26	R25	R25			R25		R25	R25	R25	R25	R25	R25				R25								
27			S14	S7		S15			S9				S6	S8	S10			40	12	13	41	3	4	5
28			S14	S7		S15			S9				S6	S8	S10				42	13	41	3	4	5
29	R8	R8			R8											R8								
30	R12	R12			R12		R12	S16	S17							R12								
31				S7					S9				S6	S8	S10						43	3	4	5
32	R13	R13			R13		R13	S16	S17							R13								
33	R11	R11			R11			S16	S17							R11								
34	R15	R15			R15		R15	R15	R15	S19	S20					R15								
35	R16	R16			R16		R16	R16	R16	S19	S20					R16								
36	R18	R18			R18		R18	R18	R18	R18	R18	S21				R18								

37					S44			S16	S17															
38	R9	R9			R9											R9								
39	R21	R21			R21		R21	R21	R21	R21	R21	R21				R21								
40	S27				R4											R4								
41							S18	S16	S17															
42	R6	S28			R6											R6								
43					S39			S16	S17															
44	R20	R20			R20		R20	R20	R20	R20	R20	R20				R20								

Anexo 6. Procesamiento de las encuestas aplicadas a los expertos.

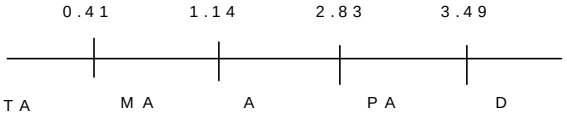
Expertos	*Aspectos*			
	A 1	**A 2**	**A 3**	**A 4**
E 1	T A	T A	T A	T A
E 2	T A	T A	T A	T A
E 3	T A	T A	T A	T A
E 4	T A	T A	T A	T A
E 5	T A	T A	M A	T A
E 6	T A	T A	T A	T A
E 7	T A	M A	A	T A
E 8	M A	M A	M A	T A
E 9	T A	P A	A	T A
E 10	T A	T A	T A	T A
E 11	T A	T A	A	M A
E 12	T A	P A	M A	T A
E 13	M A	M A	A	M A
E 14	T A	P A	T A	M A
E 15	T A	T A	M A	T A
E 16	T A	P A	A	T A
E 17	M A	M A	M A	T A
E 18	T A	T A	T A	T A
E 19	T A	T A	T A	T A
E 20	T A	P A	A	T A
E 21	T A	T A	T A	T A
E 22	T A	T A	T A	T A
E 23	M A	A	M A	T A
E 24	T A	T A	T A	T A
E 25	M A	M A	M A	T A
E 26	A	A	A	A
E 27	T A	P A	A	T A
E 28	T A	T A	T A	T A
E 29	A	M A	T A	T A
E 30	M A	M A	M A	T A

TABLA DE FRECUENCIA ABSOLUTA						
Aspectos	**TA**	**MA**	**A**	**PA**	**D**	**TOTAL**
A 1	22	6	2	0	0	30
A 2	15	7	2	6	0	30
A 3	14	8	8	0	0	30
A 4	26	3	1	0	0	30

TABLA DE FRECUENCIA ABSOLUTA ACUMULADA					
Aspectos	TA	MA	A	PA	D
A 1	22	28	30	30	30
A 2	15	22	24	30	30
A 3	14	22	30	30	30
A 4	26	29	30	30	30

TABLA DEL INVERSO DE LA FRECUENCIA ABSOLUTA ACUNULADA				
Aspectos	TA	MA	A	PA
A 1	0.7333	0.9333	1	1
A 2	0.5	0.7333	0.8	1
A 3	0.4667	0.7333	1	1
A 4	0.8667	0.9667	1	1

TABLA DE DEREMINACIÓN DE LOS PUNTOS DE CORTES							
Aspectos	TA	MA	A	PA	Suma	Promedio	N - Prom.
A 1	0.62	1.5	3.49	3.49	9.1	2.28	-0.31
A 2	0	0.62	0.84	3.49	4.95	1.24	0.73
A 3	-0.08	0.62	3.49	3.49	7.52	1.88	0.09
A 4	1.11	1.83	3.49	3.49	9.92	2.48	-0.51
Suma	1.65	4.57	11.31	13.96	31.49		
Punto de corte	0.41	1.14	2.83	3.49	7.87	1.97	



Anexo 7. Desarrollo de los casos de uso del sistema

1. Descripción textual del caso de uso **Actualizar propuesta**.

Nombre del Caso de Uso	Actualizar propuesta
Actores	TecnólogoLQC (inicia)
Propósito	Insertar, modificar o eliminar propuesta de fórmula para un indicador de reporte.
Resumen	El caso de uso inicia cuando el TecnólogoLQC decide insertar, modificar o eliminar una propuesta de fórmula de cálculo. Para insertar debe especificar la fórmula, el indicador de reporte a que está sujeta y la condición de cálculo en caso de que exista. En cualquier caso debe justificar la causa. En caso de modificación o eliminación debe existir la propuesta. En caso de inserción o modificación se debe validar la fórmula. En cualquiera de los casos se envía un mensaje de notificación al TecnólogoCM. El caso de uso finaliza cuando el TecnólogoLQC pasa a otras opciones o sale del sistema.
Requerimientos	R.10, R.2, R.13 (los casos de uso Validar usuario y Enviar mensaje son incluidos de Actualizar propuesta)
Precondiciones	El usuario debe estar logueado como TecnólogoLQC. Para modificar o eliminar tiene que existir la propuesta.
Poscondiciones	Se actualizó la propuesta.

Tabla 3. Descripción textual del caso de uso Actualizar propuesta.

2. Descripción textual del caso de uso **Visualizar propuesta**.

Nombre del Caso de Uso	Visualizar propuesta
Actores	Visualizador (inicia)
Propósito	Visualizar propuesta de fórmula de cálculo.
Resumen	El caso de uso inicia cuando el Visualizador decide ver las propuestas de fórmula y/o condición de cálculo. El caso de uso finaliza cuando el Visualizador pasa a otras opciones o sale del

	sistema.
Requerimientos	R11, R2 (el caso de uso Validar usuario es incluido de Visualizar propuesta)
Precondiciones	El usuario debe estar logueado como Visualizador.
Poscondiciones	Se visualizó la propuesta de fórmula.

Tabla 9. Descripción textual del caso de uso Visualizar propuesta.

3. Descripción textual del caso de uso **Validar fórmula**.

Nombre del Caso de Uso	Validar fórmula
Actores	—
Propósito	Validar si la expresión insertada o modificada constituye una fórmula.
Resumen	El caso de uso inicia cuando se decide insertar o modificar una fórmula. Se valida la fórmula. Si la fórmula tiene asociada una condición de cálculo, se debe validar la condición de cálculo. Así finaliza el caso de uso.
Requerimientos	R15, R18 (el caso de uso Validar condición es incluido de Validar fórmula)
Precondiciones	Se debe haber aprobado una actualización o insertado o modificado una propuesta.
Poscondiciones	Se evaluó la fórmula.

Tabla 10. Descripción textual del caso de uso Validar fórmula.

4. Descripción textual del caso de uso **Ordenar actualización**.

Nombre del Caso de Uso	Ordenar actualización
Actores	TecnólogoCM (inicia)
Propósito	Ordenar inserción, modificación o eliminación de indicador de

	reporte, fórmula o condición de cálculo.
Resumen	El caso de uso inicia cuando el TecnólogoCM decide ordenar una actualización de un indicador de reporte, una fórmula o una condición de cálculo. Para el caso de la inserción y modificación de una fórmula hay que validar la misma. En caso de la inserción o modificación de una condición hay que validar la misma. En cualquiera de los casos se envía un mensaje a AdminGTI. El caso de uso finaliza cuando el TecnólogoCM presiona el botón para confirmar la operación.
Requerimientos	R5, R16, R2, R13, R15, R18 (los casos de uso Validar usuario y Enviar mensaje son incluidos de Ordenar actualización y los casos de uso Validar fórmula y Validar condición son extendidos de Ordenar actualización)
Precondiciones	El usuario debe estar logueado como TecnólogoCM. Para modificar o eliminar un indicador de reporte, fórmula o condición, que existir.
Poscondiciones	Se ordenó la actualización del indicador de reporte o la condición de cálculo.

Tabla 11. Descripción textual del caso de uso Ordenar actualización.

5. Descripción textual del caso de uso **Evaluar propuesta**.

Nombre del Caso de Uso	Evaluar propuesta
Actores	TecnólogoCM (inicia)
Propósito	Aprobar o rechazar propuesta de inserción inserción, modificación o eliminación de fórmula.
Resumen	El caso de uso inicia cuando el TecnólogoCM decide evaluar una propuesta de fórmula para aprobarla o rechazarla. Si se aprueba se le envía un mensaje de confirmación al TecnólogoLQC y a AdminGTI y si no se le envía un mensaje de rechazo al TecnólogoLQC. El caso de uso finaliza cuando el

	TecnólogoCM sale del sistema o pasa a otras opciones.
Requerimientos	R12, R2, R13 (los casos de uso Validar usuario y Enviar mensaje son incluidos de Evaluar propuesta)
Precondiciones	El usuario debe estar logueado como TecnólogoCM. Para evaluar una propuesta tiene que existir y no haber sido evaluada con anterioridad.
Poscondiciones	Se evaluó la propuesta.

Tabla 12. Descripción textual del caso de uso Evaluar propuesta.

6. Descripción textual del caso de uso **Visualizar fórmula**.

Nombre del Caso de Uso	Visualizar fórmula
Actores	AdminGTI (inicia)
Propósito	Visualizar las fórmulas de cálculo propias de un indicador de reporte.
Resumen	El caso de uso inicia cuando AdminGTI decide visualizar una fórmula de cálculo específica de un indicador de reporte. Si la fórmula tiene condiciones entonces se debe visualizar también. El caso de uso finaliza cuando AdminGTI pasa a otras opciones o sale del sistema.
Requerimientos	R12, R19, R2 (el caso de uso Visualizar condición es extendido de Visualizar fórmula y el caso de uso Validar usuario es incluido de Visualizar fórmula)
Precondiciones	El usuario debe estar logueado como AdminGTI. Para visualizar una fórmula tienen que existir y el indicador de reporte correspondiente.
Poscondiciones	

Tabla 13. Descripción textual del caso de uso Visualizar fórmula.

7. Descripción textual del caso de uso **Actualizar indicador**.

Nombre del Caso de Uso	Actualizar indicador
Actores	AdminGTI (inicia)
Propósito	Insertar, modificar o eliminar indicador de reporte.
Resumen	El caso de uso inicia cuando AdminGTI decide insertar, modificar o eliminar un indicador de reporte. Para modificar o eliminar el indicador de reporte tiene que existir. El caso de uso finaliza cuando AdminGTI sale del sistema o pasa a otras opciones.
Requerimientos	R6, R2 (el caso de uso Validar usuario es incluido de Actualizar propuesta)
Precondiciones	El usuario debe estar logueado como AdminGTI. Para modificar o eliminar un indicador de reporte, este tiene que existir.
Poscondiciones	Se actualizó el indicador de reporte.

Tabla 14. Descripción textual del caso de uso Actualizar indicador.

8. Descripción textual del caso de uso **Actualizar fórmula**.

Nombre del Caso de Uso	Actualizar fórmula
Actores	AdminGTI (inicia)
Propósito	Insertar, modificar y eliminar una fórmula de cálculo.
Resumen	El caso de uso inicia cuando AdminGTI decide insertar, modificar o eliminar una fórmula de cálculo. En caso de modificación o eliminación debe existir la fórmula. Si la fórmula está sujeta a una condición de cálculo hay que actualizar la misma. El caso de uso finaliza cuando AdminGTI presiona el botón para confirmar la operación.
Requerimientos	R14, R2 (el caso de uso Validar usuario es incluido de

	Actualizar propuesta)
Precondiciones	El usuario debe estar logueado como AdminGTI. Para modificar o eliminar tiene que existir la fórmula.
Poscondiciones	Se actualizó la fórmula.

Tabla 15. Descripción textual del caso de uso Actualizar fórmula.

9. Descripción textual del caso de uso **Visualizar indicador**.

Nombre del Caso de Uso	Visualizar indicador
Actores	AdminGTI (inicia)
Propósito	Visualizar indicadores primario, de equipo y de reporte.
Resumen	El caso de uso inicia cuando AdminGTI decide visualizar un indicador primario, de equipo o de reporte. Para visualizar indicador primario por planta se debe seleccionar la planta. Para visualizar indicador por equipo se debe seleccionar el equipo. Para visualizar indicador por reporte se debe seleccionar el reporte. El caso de uso finaliza cuando AdminGTI pasa a otras opciones o sale del sistema.
Requerimientos	R7, R21, R22, R23, R24, R2 (el caso de uso Validar usuario es incluido de Visualizar indicador)
Precondiciones	El usuario debe estar logueado como AdminGTI. Para visualizar indicador primario por planta debe existir la planta. Para visualizar indicador por equipo debe existir el equipo. Para visualizar indicador por reporte se existir el reporte.
Poscondiciones	Se visualizó el indicador.

Tabla 16. Descripción textual del caso de uso Visualizar indicador.

10. Descripción textual del caso de uso **Visualizar variable**.

Nombre del Caso de	Visualizar variable

Uso	
Actores	AdminGTI (inicia)
Propósito	Visualizar muestras y factores que se emplean en las fórmulas de cálculo.
Resumen	El caso de uso inicia cuando el AdminGTI decide visualizar la(s) variable(s) de una fórmula. El caso de uso finaliza cuando el AdminGTI pasa a otras opciones o sale del sistema.
Requerimientos	R25, R26, R27, R28, R2 (el caso de uso Validar usuario es incluido de Visualizar variable)
Precondiciones	El usuario debe estar logueado como AdminGTI. Para visualizar las variables de una fórmula, esta debe existir.
Poscondiciones	Se visualizó la variable.

Tabla 17. Descripción textual del caso de uso Visualizar variable.