Lenguaje de Expresión de Problemas y Algoritmos para el aprendizaje de la informática en niveles preuniversitarios"

Alumnas: Villarreal, Verónica
Lucero, Sandra

Directora: Gorga, Gladys
Madoz, Cristina

ÍNDICE

INTRODUCCIÓN

El presente trabajo se basa principalmente en las diversas observaciones realizadas con respecto al alto nivel de deserción en los primeros cursos de introducción a la programación en el nivel universitario [Inf94] [Inf95] [Inf02] [Inf03], los cuales pueden ser atribuidos a [Mad04]:

- la mala orientación previa o al desconocimiento del tipo de estudio al que se van a enfrentar los alumnos pre - universitarios,
- el rol simplemente informativo brindado generalmente por los niveles medios que no beneficia la capacidad de interpretar, analizar y sintetizar,
- la inclinación en el nivel medio a soluciones inmediatas o intuitivas, las normalmente llamadas soluciones "prueba-error" en contraposición al desarrollo inductivo donde se elaboran resultados parciales que conduzcan a la solución efectiva,

- la diversidad de niveles de conocimiento no solo respecto a la programación sino también a las metodologías para resolver problemas en general,

como consecuencia de lo expresado anteriormente se considera de fundamental importancia que existan mecanismos que posibiliten actividades con los alumnos interesados en la disciplina y que están cursando los últimos años de la escuela media.

El objetivo final es el desarrollo de un entorno específico que ayude y conduzca efectivamente a los estudiantes y docentes en el proceso de aprendizaje de la programación de manera más gráfica, diversa y comprensible. Esta adaptación intenta ayudar a superar los inconvenientes a los que se enfrentan los interesados en la disciplina, que haga atractivo e interesante el proceso de enseñanza-aprendizaje y que beneficie el autoaprendizaje y la autoevaluación por parte de los estudiantes, ayudándolos a evaluar su interés y competencia para realizar estudios vinculados a la especialidad y aproximándolos, en una etapa muy temprana en sus estudios, a algunos de los problemas de la disciplina informática. La idea justificadora es que este acercamiento pueda permitirle al alumno confirmar o desechar su interés en la resolución de problemas y, en caso afirmativo, le permita definir las capacidades que debe desarrollar para hacerlo eficientemente y así estimular el desarrollo de las habilidades para resolver problemas de manera autónoma y precisa, con la intención de disminuir la dificultad para alcanzar los niveles de abstracción necesarios y expresar rigurosamente los algoritmos.

Para cumplir con los objetivos planteados se realizaron investigaciones sobre diversos temas relacionados, análisis de la resolución de problemas solubles por computadoras, análisis de diferentes herramientas de expresión de algoritmos, análisis del Ambiente de Programación Visual Da Vinci, y finalmente implementación de LEPA – Nivel 1.

Este último punto concluyó con la adaptación de la herramienta actualmente en uso en el Curso de Ingreso a la Facultad de Informática y en el primer año de la carrera, con la intención de generar una extensión simple, accesible y de entorno amigable para así facilitar la expresión de problemas solubles por computadora junto con el diseño y desarrollo de algoritmos en los niveles preuniversitarios.

Luego de la investigación teórica realizada, numerosos autores consideran que la utilización de una herramienta informática con las características mencionadas la cual no actúe como elemento de dispersión en la elaboración y en la interpretación de algoritmos,

puede resultar eficaz como un recurso adicional tendiente a mejorar la información y preparación específica de los alumnos para facilitar las posibilidades de acceso y permanencia en la Universidad y por otro lado que ayude a definir la orientación vocacional del alumno en los últimos años de los estudios secundarios.

En otro sentido, también los docentes podrán utilizarlo como un recurso adicional en su capacitación, de modo de poder actuar como mediadores en el proceso de aprendizaje de los alumnos interesados en carreras informáticas.

El presente trabajo esta organizado de la siguiente manera, en el primer capítulo se presenta en forma detallada la etapa de análisis y resolución de problemas implementables por computadoras, para ello se comienza con la revisión del término, desde el punto de vista del uso coloquial en la lengua española, para concluir con una definición general de problema. Se presenta además una visión detallada sobre las etapas por las cuales se deben atravesar para resolver problemas. Se proporcionan definiciones de lenguaje, en particular de programación, indicando la necesidad de definir concretamente la sintaxis y semántica de los mismos, para que la implementación posea un único significado a nivel de computadora y así poder ejecutar instrucciones. Este capítulo finaliza presentando una breve reseña sobre la lógica de enunciados y su estrecha relación con la resolución de problemas.

El segundo capítulo muestra diferentes herramientas de expresión de algoritmos que se pueden utilizar para representar los problemas planteados en el primer capítulo, iniciándolo con un detalle sobre la teoría de autómatas y lenguajes formales (Lenguajes Abstractos). Luego se plantea una introducción a los lenguajes naturales describiendo como pueden ser incorporados en la expresión de problemas y algoritmos, y cuales son sus inconvenientes. Posteriormente se detallan las ventajas del uso de herramientas visuales basados en el hecho de que los humanos procesan, por su naturaleza, más fácil y rápidamente las imágenes que el texto, esperando que éstas permitan escribir soluciones pensadas por los alumnos para un problema dado y visualizando la ejecución de las mismas de modo de realizar una verificación experimental.

El aprendizaje de la programación estructurada tiene importancia no sólo desde de la adquisición de conocimiento, sino también desde el desarrollo de las capacidades de pensamiento. La visualización es importante debido a su fuerza expresiva: «una imagen vale más que mil palabras». Mapas, diagramas, señales de circulación, etc. son signos y señales omnipresentes en la vida cotidiana; los anuncios, la televisión y la Web también están

repletos de imágenes. En las Ciencias Informáticas resulta difícil concebir una actividad de ingeniería sin representaciones de los objetos que desarrolla. En este caso, el objetivo de la visualización es ayudar a la comprensión, diseño, análisis o uso. Debido a que la Visualización de Software es una de las áreas de las Ciencias de la Computación desde la cual se pueden hacer aportes interesantes relacionados con la enseñanza-aprendizaje de la programación, se plantea en este capítulo una introducción al tema.

En el último punto de este capítulo se desarrolla el tema de mapas conceptuales, herramienta de enseñanza – aprendizaje, el cual es una técnica utilizada en diferentes niveles educativos permitiendo la incorporación y asimilación de conceptos y proposiciones nuevas mediante aprendizaje significativo. El objetivo de este punto es presentar una técnica simultáneamente con una estrategia de aprendizaje, un método para captar lo más significativo de un tema y un recurso esquemático para representar un conjunto de significados conceptuales.

En el tercer capítulo se realiza un análisis exhaustivo sobre la herramienta actualmente en uso en el Curso de Ingreso de la Facultad de Informática y en el primer año de la carrera, denominada Visual Da Vinci, para realizar su adaptación al nivel medio.

Se concluye el cuarto capítulo con una explicación sobre los desarrollos propuestos en los diferentes niveles de aprendizaje, haciendo hincapié en la implementación de "Lepa" - Lenguaje de Expresión de Problemas y Algoritmos como un primer nivel preuniversitario.

Agradecimientos:

Queremos agradecer en forma conjunta a nuestras directoras de tesis, Gladys Gorga y Cristina Madoz, las cuales permitieron que lleguemos a buen puerto. Su guía fue vital para el desarrollo del presente trabajo, realmente cada vez que necesitamos su apoyo siempre estuvieron, aun sin necesidad de estarlo. A Armando De Giusti, que nos impulsó para que terminemos la carrera. A la presencia en el momento justo de Ariel Pasini, a la creatividad de Alfredo Lahitte.

En particular quiero agradecer a mi familia, que siempre estuvo en las buenas y en las malas incondicionalmente. Especialmente a mi abuela, que ha logrado que me convierta en la persona que soy y ha impulsado algo invalorable en mi, que nunca baje los brazos y que constantemente tenga fe. A los amigos que "nacieron" en el transcurso de la carrera y aun continúan, gracias, porque me permitieron vivir junto a ellos esta etapa única. A mis alumnos de Programación y de los cursos de ingreso de los que fui auxiliar, ellos hicieron revivir en mí las ganas de transmitir el poco o mucho conocimiento que tengo; parte de este trabajo se lo debo a ellos también. A Úrsula y a las "Graciela's", amigas incondicionales que siempre supieron que podía llegar. A Daniel, persona que transita por mi vida este último tiempo de facultad, gracias por escuchar, aconsejar, estar y tolerar mi trastorno "pre-tesis".

Sandra Lucero

Mi agradecimiento es para dos personitas muy especiales, mi marido y mi hijo, que pese a sentirse un poquito celosos me apoyaron siempre, y aunque no les fue fácil, cedieron un poquito de nuestro tiempo compartido para que pudiera cumplir una de mis metas mas importantes y pendientes, recibirme. A ellos y a mi familia, gracias a los cuales y por quienes soy los que soy.

Verónica Villarreal

CAPÍTULO I

ANÁLISIS Y RESOLUCIÓN DE PROBLEMAS

Los humanos se caracterizan, entre otras cosas, por la capacidad de hablar y expresarse mediante signos orales y escritos. El lenguaje hablado cotidiano es el principal medio de comunicación, teniendo una fuerte repercusión sobre los demás medios, dirigiendo y sirviendo de eje interpretativo. Por otra parte las matemáticas, como muchas otras disciplinas, crean su propio lenguaje, el cual comienza a construirse desde los primeros años de la matemática básica. El lenguaje matemático se caracteriza por su precisión en los significados, indispensable para la efectiva comunicación de las ideas. En particular, en el terreno de la educación informática, el término problema merece una reflexión sobre su significado, debido a que el análisis y la resolución de problemas posee sus bases en las matemáticas.

Para ello se comienza con la revisión del término, desde el punto de vista del uso coloquial en la lengua española, para concluir con una definición general de problema. Se presenta además una visión detallada sobre las etapas por las cuales se deben atravesar para resolver problemas. Se dan definiciones de lenguaje, en particular de programación,

indicando la necesidad de definir concretamente la sintaxis y semántica de los mismos, para que la implementación posea un único significado a nivel de computadora y así poder ejecutar instrucciones. Este capítulo concluye presentando una breve reseña sobre la lógica de enunciados y su estrecha relación con la resolución de problemas.

1.1 – Problemas del mundo real

1.1.1 - Definiciones

Según el significado que le atribuye el Diccionario de la Real Academia Española al término problema [Rae05]:

- (Del lat. Problema, y este del gr. προβλημα.) Cuestión que se trata de aclarar.
- Proposición o dificultad de solución dudosa.
- Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.
- Disgusto, preocupación. Ú.m. en pl. Mi hijo sólo da problemas.
- Mat. Proposición dirigida a averiguar el modo de obtener un resultado cuando ciertos datos son conocidos.
- Determinado. Mat. Aquel que no puede tener sino una solución, o más de una en un número fijo.
- Indeterminado. Mat. Aquel que puede tener indefinido número de soluciones.

Por otra parte el Diccionario Enciclopédico Santillana [Des91] expresa:

- Problema (del lat. problema, y éste del gr. problema, de proballo, lanzar hacia adelante) s.m.
- 1. Cuestión que se intenta resolver o en la que hay algo que averiguar, particularmente aquella en la que se conocen ciertos datos que hay que estudiar y manejar según unas reglas para obtener la respuesta o el resultado que se pide: Tengo que hacer un problema de matemáticas y otro de física.

- 2. Situación negativa o perjudicial que tiene difícil solución: el problema del paro, el problema económico, el problema del terrorismo.

- 3. Hecho o circunstancia que impide o dificulta que se logre un fin determinado: Con tantos problemas no vamos a poder acabar el trabajo. 4

- . Disgusto, preocupación: Sus hijos le crearon muchos problemas.

Como se puede observar, el término problema tiene una gran variedad de usos o acepciones coloquiales, entre los que se destacan:

"Cuestión que se trata de aclarar o resolver, existencia de una dificultad, un disgusto, …". Algunas de las acepciones se expresan en un contexto matemático, por ejemplo: "Proposición dirigida a averiguar el modo de obtener un resultado cuando ciertos datos son conocidos".

Desde el punto de vista matemático, el término problema involucra:

- Una proposición o enunciado.

- Unos conjunto de datos conocidos que hay que estudiar.

- Una acción que alguien o algunos sujetos deben averiguar.

- Una meta u objetivo, esto es, obtener un resultado.

- Un procedimiento para alcanzar el resultado.

- Unas reglas que deben seguirse para alcanzar la meta.

Por otro lado, asociados al término problema, aparecen otros, de los cuales se destacan los siguientes:

- Resolver: hallar la solución de un problema [Rae05]

- Solución: ..., acción y efecto de resolver una duda o dificultad, cada una de las cantidades que satisfacen las condiciones de un problema o de una ecuación,..." [Rae05]

- Solvente: que resuelve [Rae05]

Según estas definiciones, por solución o resolución se entiende la acción y efecto de resolver un problema y por solvente el sujeto que está inmerso en la tarea de resolverlo.

Entre los problemas y tareas que se presentan diariamente, algunos de ellos pueden ser resueltos mediante el uso de la capacidad intelectual y la habilidad manual de la persona, es decir, se pueden realizar de forma manual. Pero existe un gran número de problemas que pueden resultar tediosos si se persigue la convención que detallada en el párrafo anterior. Como solución a esto último, y gracias a la computadora, dicho trabajo se vuelve automático, se realiza con mayor rapidez, se reduce el número de errores en los cálculos, etc, esto debido a las características de dichas máquinas: rapidez, precisión y memoria.

Si se piensa en la forma que una persona indica a otra como resolver un problema, se observa habitualmente que utiliza el lenguaje cotidiano para realizar la explicación. El riesgo que esto involucra es demasiado grande debido a las ambigüedades que posee el lenguaje natural. Las personas que tienen cierta experiencia al respecto saben lo dificultoso que es comunicar correctamente el mensaje de manera que el receptor entienda exactamente lo que se quiere transmitir. Lamentablemente, con mucha frecuencia se malinterpretan las instrucciones y por lo tanto se ejecuta incorrectamente la solución, obteniéndose un resultado erróneo.

Al querer implementar una solución a un problema a través de una computadora, es fundamental entender exactamente lo que se quiere realizar, con que información se cuenta y que tipo de transformación se desea realizar sobre ella. Además se debe tener en cuenta que la solución no admitirá más de una interpretación para una misma acción.

Debido a que una máquina no posee la capacidad de decisión del ser humano para resolver situaciones no previstas, al producirse una situación no contemplada, se deberá abortar la tarea y recomenzar todo el procedimiento nuevamente.

A continuación se analiza en forma general las distintas etapas que deben seguirse para poder resolver un problema utilizando una computadora como herramienta.

1.1.2 – Etapas de la resolución de problemas

En el proceso de solución de problemas resolubles a través de una computadora se identifican, en forma general, tres secciones bien definidas (Fig. 1.1).

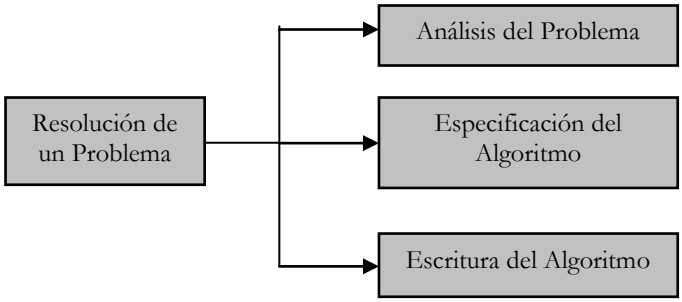


Figura 1.1. Etapas en la resolución de problemas

1.1.2.1 – Análisis del Problema

El objetivo de analizar un problema es el de poseer cierta comprensión de la naturaleza del mismo. El problema debe ser evaluado en su contexto. Como resultado de este análisis se tiene un modelo preciso del objetivo del problema y su ambiente. El análisis habrá concluido cuando los siguientes pasos se hayan completado en forma total (Fig. 1.2):

- definir el problema con total precisión
- especificar los datos de partida necesarios para la resolución del mismo (especificaciones de entrada)
- especificar la información que debe proporcionarse al resolverse (especificaciones de salida)

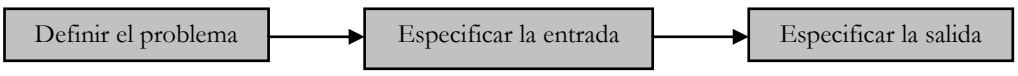


Figura 1.2. Tareas involucradas en el análisis de un problema

1.1.2.2 – Especificación del algoritmo

La siguiente etapa es la de especificar el algoritmo que expresará la solución al problema planteado.

La palabra algoritmo se cree que proviene del nombre de Al' Khowarizmi,

matemático del siglo IX, que escribió un tratado sobre métodos de álgebra, recopilando los pasos a seguir para multitud de problemas recogidos hasta entonces. Se ha usado este término como el de un "recetario" o colecciones de "pasos" para resolver problemas. Más específicamente, la definición que otorga el diccionario es la siguiente:

"Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema"

Adecuando el término a su importancia en ciencias de la computación basan la descripción de algoritmo en la de procedimiento, en el cual lo definen de la siguiente manera [Hop79]:

"Un procedimiento es una secuencia finita de instrucciones que pueden realizarse mecánicamente. Un procedimiento que siempre termina es un algoritmo".

Un algoritmo es un conjunto de instrucciones efectivo y explícito, para un procedimiento de cómputo (no necesariamente numérico) que puede usarse para encontrar la respuesta a cualquier pregunta entre las de una clase determinada [Ham81]

Planteado un problema se necesita un método que sea capaz de resolverlo. Esto es un algoritmo. Para hablar de resolución de problemas hacen falta los siguientes puntos:

- la existencia de un problema.
- la existencia de un medio para resolverlo.

En general, para resolver un problema es necesario que la información esté claramente especificada, que la solución pueda ser reconocida cuando el problema esté resuelto y que el problema no cambie cuando se va a resolver.

Como conclusión, un algoritmo es, en forma intuitiva, un conjunto de instrucciones o de especificaciones sobre un proceso para hacer una determinada tarea. Esa tarea generalmente es la solución de un problema de algún tipo. Se espera que un algoritmo tenga las siguientes propiedades:

- no debe ser ambiguo, esto es, que no permita mas de una interpretación
- cada instrucción del algoritmo debe tener significado unívoco

- debe completarse en tiempo finito, es decir, el algoritmo debe detenerse y debe informar de alguna manera, si ha logrado o no el objetivo deseado.

1.1.2.3 – Escritura del algoritmo

Una vez que el algoritmo está diseñado y representado, se continúa con la fase de escritura o programación del mismo. Esta etapa se puede descomponer en (Fig. 1.3):

- Codificación del algoritmo en un programa
- Ejecución del programa
- Verificación del programa

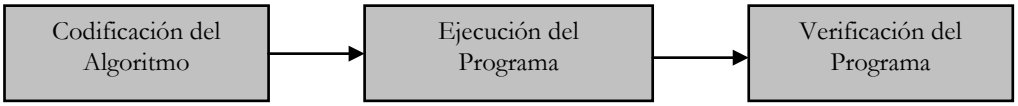


Figura 1.3. Tareas involucradas en la fase de escritura de un algoritmo

Un programa es una serie de instrucciones las cuales le indican a la computadora que ejecute una tarea determinada, con el objetivo de dirigir el funcionamiento de la máquina. Para ello se utiliza un lenguaje de programación, el cual es una notación formal para describir algoritmos a ser ejecutados en una computadora.

La fase de conversión de un algoritmo en instrucciones de un lenguaje de programación se denomina codificación.

Tras la codificación del programa, éste deberá ejecutarse en una computadora. El resultado de esta primera ejecución en general es incierto, ya que existe una alta posibilidad de que aparezcan errores tanto en la codificación como en el algoritmo. Por lo tanto, el paso siguiente consiste en comprobar el correcto funcionamiento del programa y en asegurarse, en la medida de lo posible, de la validez de los resultados proporcionados por la máquina.

Las aplicaciones reales y potenciales de la computación han alcanzado niveles inimaginables apenas veinte años atrás. A pesar de su uso extensivo, uno de los costos más alto no se da en la producción del software, sino en la corrección de errores que son

detectados posteriormente al desarrollo del sistema. En la actualidad, el método más usado para validar software es el "testing", que consiste en la simulación sobre casos de prueba representativos. No obstante, este método no garantiza la corrección del software analizado, por ser incompleto en la mayoría de los casos.

En las aplicaciones críticas, que tratan con vidas humanas y/o grandes inversiones económicas, la certeza de corrección es, en general, un criterio indispensable. De un software correcto se espera que resuelva un problema determinado por una especificación y que exista una justificación formal –matemática– de que el programa la satisface.

En los últimos años un gran esfuerzo de investigación se ha invertido en el desarrollo de métodos y herramientas para la especificación y el análisis de la corrección de sistemas. Sin embargo no hay un formalismo, una metodología o una herramienta claramente preferibles a otras en todas circunstancias.

Para el análisis de la corrección formal de sistemas se destacan dos importantes enfoques:

- Verificación de corrección. En este enfoque un sistema es considerado correcto cuando se prueba que toda ejecución posible satisface la especificación. Existen algunas técnicas bien conocidas que permiten recorrer, en ciertos casos, de manera exhaustiva el espacio de ejecuciones posibles y herramientas que las implementan.

- Demostración de corrección. En este caso se construye o deriva una prueba matemática de que el sistema satisface su especificación. Aquí las herramientas asisten al programador en el proceso de construcción de la prueba. Una de las principales características de los mismos es el carácter unificador de la teoría que implementan, en la cual pueden ser expresados programas, teoremas y pruebas de éstos. Otro punto destacable es que el usuario es guiado en forma interactiva por el sistema en el proceso de construcción de un programa o una prueba, siendo verificada inmediatamente la validez de cada paso del desarrollo. El principal objetivo de estos sistemas es convertirse en sofisticadas herramientas que asistan en la tarea del desarrollo incremental de programas correctos. Sin embargo, el marco conceptual necesario para desarrollar software verificado es de elevada complejidad y requiere cubrir muchos aspectos que en realidad escapan a la construcción de un asistente de pruebas. Estos sistemas

disponen de un lenguaje de especificación de orden superior, permiten hacer pruebas en lógica de alto orden y proveen definiciones de tipos inductivos y deductivos.

Además de constituir una de las áreas más importantes de la computación teórica, la semántica de lenguajes de programación es uno de los fundamentos matemáticos de la verificación de programas, la cual consiste en la demostración formal (es decir, por medio de herramientas matemáticas como la lógica y el álgebra) de que un programa dado cumple con la función para la cual fue diseñado. El objetivo es producir software confiable.

1.2 – Lenguajes de programación

1.2.1 - Definiciones

Informalmente un lenguaje es un conjunto de palabras o sentencias formadas sobre un alfabeto. Se entiende por alfabeto a un conjunto de símbolos, no vacío, de elementos. Según el significado que le otorga la [Rae05]:

(Del prov. lenguatge):

- Conjunto de sonidos articulados con que el hombre manifiesta lo que piensa o siente.
- Lengua (sistema de comunicación verbal).
- Manera de expresarse. Lenguaje culto, grosero, sencillo, técnico, forense, vulgar
- Estilo y modo de hablar y escribir de cada persona en particular.
- Uso del habla o facultad de hablar.
- Conjunto de señales que dan a entender algo. El lenguaje de los ojos, el de las flores.
- Inform. Conjunto de signos y reglas que permite la comunicación con un ordenador
- Lenguaje de alto nivel. Inform. lenguaje que facilita la comunicación con un

computador mediante signos convencionales cercanos a los de un lenguaje natural.

- Lenguaje ensamblador. Inform. lenguaje muy similar al de máquina, con pequeñas modificaciones mnemotécnicas que facilitan su uso. Es de nivel inmediatamente superior al de máquina.

- Lenguaje de máquina. Inform. Conjunto de instrucciones codificadas que una computadora puede interpretar y ejecutar directamente.

Observando las cuatro últimas definiciones se puede concluir que un lenguaje es un medio de comunicación el cual posee sintaxis (conjuntos de signos) y semántica (reglas), las cuales deben tener un significado unívoco para poder ser comprendidos por una computadora.

1.2.2 – Traducción e interpretación de un programa

Debido a que las computadoras solo ejecutan programas escritos en lenguajes de máquina, existen dos alternativas para la implementación [Ghe87]:

- **Interpretación**

Las acciones que implican sentencias del lenguaje se ejecutan directamente. Generalmente existe un subprograma para cada acción posible (escrito en lenguaje de máquina). Más precisamente un intérprete es un programa que ejecuta la siguiente secuencia:

- obtiene la sentencia siguiente

- determina las acciones que se deben ejecutar

- ejecuta las acciones

Lo anteriormente detallado tiene similitud al conjunto de acciones llevadas a cabo por un procesador tradicional, con lo que se puede considerar a la interpretación como una simulación en un ordenador anfitrión de una máquina de propósito especial, cuyo lenguaje de máquina es el lenguaje de alto nivel.

- **Traducción**

Dado un lenguaje escrito en su sintaxis original, la traducción es el proceso de trasladar el código al lenguaje de máquina antes de ser ejecutado. El proceso de traducción de un programa escrito en su sintaxis original es fundamental en toda implementación de un lenguaje de programación.

La traducción se puede dividir en dos partes principales: el análisis del programa fuente dado como entrada y la síntesis del programa objeto ejecutable. Dentro de cada una de estas partes hay a su vez divisiones, a continuación se muestra una figura que ilustra la estructura de un compilador típico:

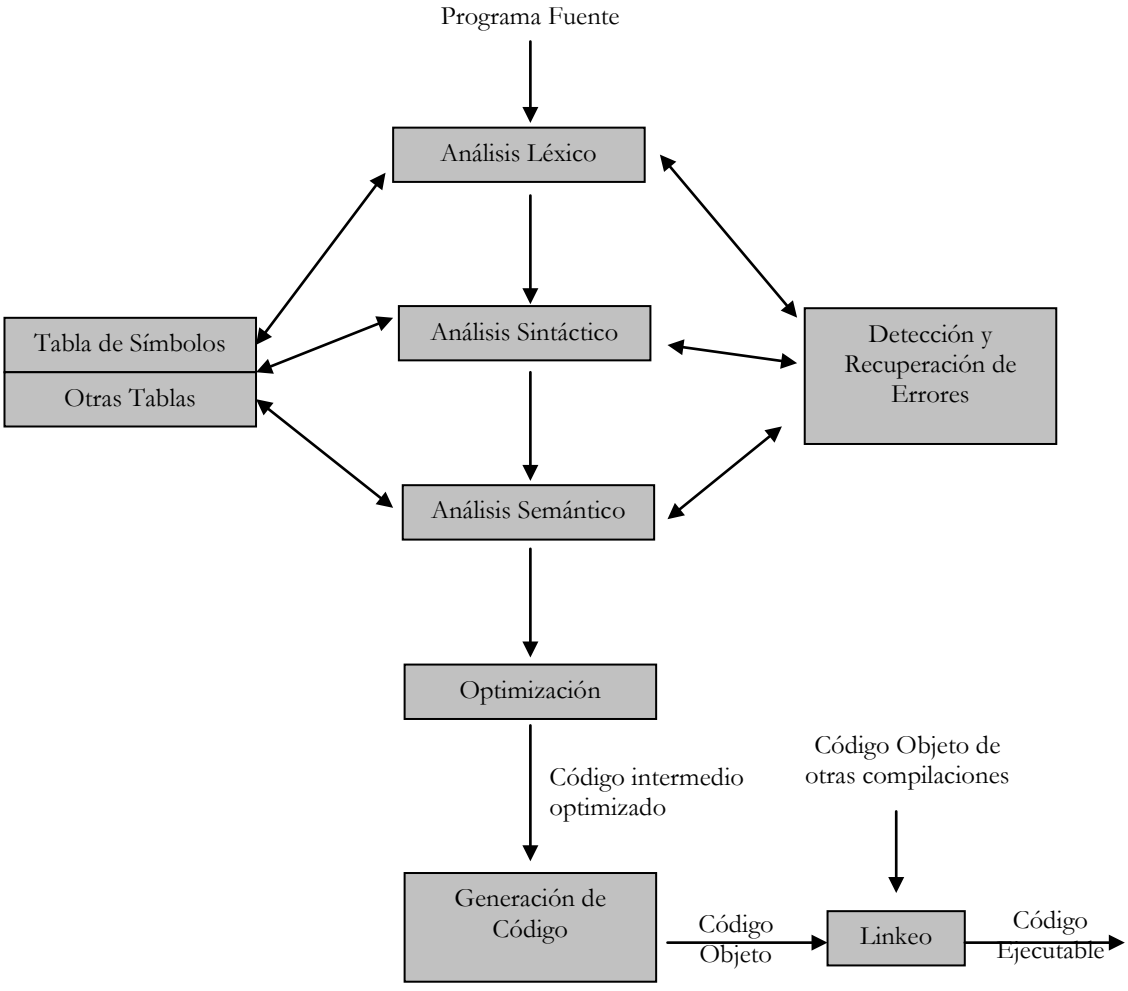


Figura 1.4. Estructura de un compilador

Para un traductor el programa fuente es una secuencia de cientos, miles o mas

caracteres no diferenciados. El análisis de la estructura de un programa se debe construir carácter a carácter.

- **Análisis Lexicográfico**

La fase básica del proceso de traducción es agrupar la secuencia de caracteres en sus componentes elementales: identificadores, delimitadores, símbolos de operadores, números, palabras claves, blancos, comentarios.

Esta fase se denomina análisis lexicográfico o scanner y los elementos básicos que resultan de este análisis se llaman tokens o items lexicográficos.

- **Análisis sintáctico**

Se denomina sintaxis de un lenguaje a la forma que tienen los elementos básicos del programa (identificadores, palabras claves, etc), expresiones, sentencias y unidades del programa.

Por ejemplo, la sintaxis (es decir como se escribe) de una sentencia if en C es:

if (<expr>) <sentencia>

El propósito principal de la sintaxis es proveer una notación para comunicación entre el programador y el procesador del lenguaje de programación.

A continuación se enumeran los criterios sintácticos generales que debe tener un lenguaje:

- Legibilidad: Un programa es legible en la medida que es autodocumentado.
- Facilidad para escribir programas: Se busca estructuras sintácticas concisas, regulares y poderosas.
- Facilidad de Traducción: Cuando más regular es la estructura de un programa, más simple es de traducir. Cuantas más construcciones sintácticas hay, más difícil es de traducir.
- Facilidad de Verificación: Prueba de programas o correctitud de programas.

- Falta de ambigüedad: La ambigüedad es un problema importante en todo diseño de lenguajes. Una definición de lenguaje debería proveer un único significado para toda construcción sintáctica. Una construcción ambigua permite dos o más interpretaciones diferentes.

1.2.3 – Sintaxis

Todo lenguaje de programación está formado por un conjunto de "símbolos básicos", que se agrupan para formar los elementos de un "vocabulario", de la misma manera que en la lengua española las letras se agrupan para formar las palabras. Los símbolos de los lenguajes de programación son los caracteres que forman el código, y a sus "palabras" se las llama tokens.

Las regla de un lenguaje indican cómo pueden o no agruparse los diferentes tokens. A estas reglas se las denominan reglas sintácticas o comúnmente sintaxis. Es frecuente que el vocabulario se defina implícitamente al definir las reglas sintácticas de un lenguaje. Al conjunto de reglas sintácticas de un lenguaje se la llama gramática. El conjunto de las reglas sintácticas del castellano es la gramática castellana.

Por último, un lenguaje suele ir acompañado de ciertas reglas que complementan a las reglas sintácticas, a este conjunto de reglas se las denominada semántica. En el lenguaje natural se utiliza la lógica y la memoria para saber si una frase tiene sentido. Por ejemplo, aunque en español la frase "El sediento bebió un vaso de tierra" es perfectamente válida a nivel gramatical, se detecta el error gracias al sentido común.

En conclusión, la sintaxis de un lenguaje de programación es el conjunto de reglas formales que especifican la estructura de los programas pertenecientes a dicho lenguaje. Como se mencionó en el párrafo anterior, un lenguaje debe ser especificado a través de una gramática. Se retoma este tema en el capítulo 3, el cual se refiere, entre otros temas, a la descripción de lenguajes formales y autómatas.

1.2.4 – Semántica y Análisis

La semántica de un lenguaje de programación es el conjunto de reglas que

especifican el significado de cualquier sentencia sintácticamente válida. Existen dos formas para describir las especificaciones semánticas de un lenguaje de programación: natural y formal.

La especificación natural de la semántica de un lenguaje de programación se basa en utilizar el lenguaje natural para especificar las características semánticas del lenguaje.

La especificación formal puede dividirse en dos grandes grupos:

- especificación por medio de gramáticas atribuidas o gramáticas con atributos.
- utilización de metalenguajes de especificación semántica.

La especificación natural de la semántica de un lenguaje de programación es describir mediante un lenguaje natural sus características semánticas.

El analizador semántico deberá comprobar las especificaciones semánticas relatadas en lenguaje natural y las especificaciones semánticas derivadas de la sintaxis del lenguaje de programación.

- Compatibilidad de tipos

La compatibilidad de tipos (type checking) es la principal especificación semántica derivada de la sintaxis del lenguaje.

- Rutinas semánticas

Las especificaciones semánticas descritas en lenguaje natural, no tienen reglas fijas para ser implementadas en el analizador semántico. Normalmente, lo que se hace es incorporar rutinas (procedimientos o funciones) denominadas semánticas, que comprueban las especificaciones descritas en lenguaje natural. Dependiendo de la complejidad de la especificación, unas veces estas rutinas semánticas serán verdaderos procedimientos o funciones, y en otros casos serán simplemente algunas sentencias intercaladas entre las específicas del análisis sintáctico.

La especificación formal de los aspectos semánticos de los lenguajes de programación se basa en la definición metamatemática, y no en la definición por medio del lenguaje natural. Es, por tanto, una especificación mucho más precisa, sin embargo requiere el uso de unas herramientas que pueden dividirse en dos grandes grupos:

- Gramáticas atribuidas o con atributos.
- Lenguajes de especificación semántica.

Ejemplo 1: llamada a funciones

En la llamada a subrutinas se debe comprobar que el número de argumentos coincida con el número de parámetros ficticios. Esto no lo puede describir la sintaxis del lenguaje, es función del analizador semántico dicha comprobación.

Finalmente, el análisis semántico de un procesador de lenguaje es la fase encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico.

El objetivo principal del analizador semántico de un procesador de lenguaje es asegurarse de que el programa analizado satisfaga las reglas requeridas por la especificación del lenguaje, para garantizar su correcta ejecución. El tipo y dimensión de análisis semántico requerido varía enormemente de un lenguaje a otro. En lenguajes interpretados como Lisp o Smalltalk casi no se lleva a cabo análisis semántico previo a su ejecución, mientras que en lenguajes como Ada, el analizador semántico deberá comprobar numerosas reglas que un programa fuente está obligado a satisfacer.

Se ve, pues, como el análisis semántico de un procesador de lenguaje no modela la semántica o comportamiento de los distintos programas construidos en el lenguaje de programación, sino que, haciendo uso de información parcial de su comportamiento, realiza todas las comprobaciones necesarias – no llevadas a cabo por el analizador sintáctico – para asegurarse de que el programa pertenece al lenguaje.

Las fases finales de la traducción tienen que ver con la construcción del programa ejecutable a partir de la salida producida por el analizador semántico. Esta fase involucra la generación de código y pueden también incluir optimización del programa generado. Si los subprogramas son traducidos en forma separada, o si se utilizan subprogramas de librerías, será necesaria una fase de enlaces (linkeos) para producir el programa ejecutable completo.

Ejemplo 2: Dado el siguiente ejemplo de código en C:

```
superficie = base * altura / 2;
```

La sintaxis del lenguaje C indica que las expresiones se pueden formar con un conjunto de operadores y un conjunto de elementos básicos. Entre los operadores, con

sintaxis binaria infija, se encuentran la asignación, el producto y la división. Entre los elementos básicos de una expresión existen los identificadores y las constantes enteras sin signo (entre otros).

Su semántica identifica que en el registro asociado al identificador superficie se le va a asociar el valor resultante del producto de los valores asociados a base y altura, divididos por dos (la superficie de un triángulo).

Finalmente, el análisis semántico del procesador de lenguaje, tras haber analizado correctamente que la sintaxis es válida, deberá comprobar que se satisfacen las siguientes condiciones:

- Que todos los identificadores que aparecen en la expresión hayan sido declarados en el ámbito actual, o en alguno de sus ámbitos (bloques1) previos.
- Que la subexpresión de la izquierda sea semánticamente válida, es decir, que sea un lvalue2.
- Que a los tipos de los identificadores base y altura se les pueda aplicar el operador de multiplicación. Un registro en C, por ejemplo, no sería válido.
- Deberá inferirse el tipo resultante de la multiplicación anterior. Al tipo inferido se le deberá poder aplicar el operador de dividir, con el tipo entero como multiplicando.
- Deberá inferirse el tipo resultante de la división y comprobarse si éste es compatible con el tipo de superficie para llevar a cabo la asignación. Como ejemplo, si superficie fuese entera y división real, no podría llevarse a cabo la asignación.

La fase de análisis semántico obtiene su nombre por requerir información relativa al significado del lenguaje.

1.3 – Relación entre la resolución de problemas y la lógica

La elaboración de algoritmos lógicos consiste en el planteo de forma esquemática de los pasos lógicos a seguir en la resolución y el análisis de cada problema, teniendo en

cuenta sus aspectos tanto conceptuales como de tipo práctico [Gar00].

Para ello se deben dar las bases lógicas que sustenten un buen diseño de la solución al problema planteado, entendiéndose por buen diseño al que, a partir de ciertos requerimientos de entrada, genere la salida desea.

En los puntos siguientes se desarrollan conceptos básicos de la lógica proposicional, necesarios en la resolución de problemas de baja y media complejidad que se enseñan en las primeras etapas de un curso tradicional de programación.

La lógica estudia la forma del razonamiento, es una disciplina que por medio de reglas y técnicas determina si un argumento es válido. Es ampliamente aplicada en la filosofía, la matemática, la informática, entre otras ciencias. En la filosofía para determinar si un razonamiento es válido o no, ya que una frase puede tener diferentes interpretaciones, sin embargo la lógica permite saber el significado correcto. En la ciencia matemática para demostrar teoremas e inferir resultados matemáticos que puedan ser aplicados en investigaciones. En la ciencia informática para desarrollar y verificar programas.

En general la lógica se aplica en la tarea diaria, ya que cualquier trabajo que se realiza tiene un procedimiento lógico, por el ejemplo; para ir de compras al supermercado una ama de casa tiene que realizar cierto procedimiento lógico que permita realizar dicha tarea. Si una persona desea pintar una pared, este trabajo tiene un procedimiento lógico, ya que no puede pintar si antes no prepara la pintura, o no debe pintar la parte baja de la pared si antes no pintó la parte alta porque se mancharía lo que ya tiene pintado, también dependiendo si es zurdo o derecho, él puede pintar de izquierda a derecha o de derecha a izquierda según el caso, todo esto es la aplicación de la lógica.

El objetivo fundamental de este tema es el de introducir, de manera intuitiva, los conceptos fundamentales de la lógica, y muy particularmente, el concepto de consecuencia, ya que la lógica puede ser caracterizada como el estudio de la consecuencia; o lo que es lo mismo, como el estudio de los razonamientos válidos o correctos.

1.3.1 - Definición

La lógica matemática es la disciplina que trata de métodos de razonamiento. En un nivel elemental, la lógica proporciona reglas y técnicas para determinar si es o no válido un

argumento dado. El razonamiento lógico se emplea en matemáticas para demostrar teoremas; en ciencias de la computación para verificar si son o no correctos los programas; en las ciencias física y naturales, para sacar conclusiones de experimentos; y en las ciencias sociales y en la vida cotidiana, para resolver una multitud de problemas. Ciertamente se usa en forma constante el razonamiento lógico para realizar cualquier actividad.

Para definir una Lógica se define un lenguaje artificial; con un alfabeto y unas reglas gramaticales de formación de fórmulas y se atribuye significado a las expresiones del lenguaje mediante interpretaciones semánticas. Dichas interpretaciones nos permiten decir, en algunos casos, que de ciertos conjuntos de fórmulas (que se toman como hipótesis) se siguen ciertas fórmulas. Es decir, que dichas fórmulas son consecuencia semántica de las hipótesis consideradas.

Lógica = Gramática + Semántica

En algunas ocasiones se puede definir un cálculo deductivo que permite mecanizar el proceso de extraer conclusiones a partir de hipótesis. Por supuesto, se desea que el cálculo sea una réplica mecanizable de dicho proceso; es decir, equivalente (los mismos resultados).

Semántica $\Leftrightarrow$ Cálculo

La Lógica es una herramienta que sirve para computar razonamientos, especialmente cuando el rigor y la precisión son imprescindibles. Esto sucede en matemáticas, filosofía, informática, etc. Pero un lenguaje lógico es también un objeto de estudio; se puede ver qué propiedades tiene el lenguaje; si el concepto de consecuencia se puede retener mediante las reglas de un cálculo, si hay cálculos más efectivos, incluso si existen algoritmos capaces de suplirlos.

1.3.2. - Lógica proposicional o de enunciados

Una proposición o enunciado es una oración que puede ser falsa o verdadera pero no ambas a la vez. La proposición es un elemento fundamental de la lógica matemática.

A continuación se muestran algunos ejemplos de proposiciones válidas y no válidas, y se explica porque algunos enunciados no son proposiciones. Las proposiciones se

indican por medio de una letra minúscula, dos puntos y la proposición propiamente dicha.

Ejemplo:

p: La tierra es plana.

q: -17 + 38 = 21

r: x > y-9

t: Hola ¿como estas?

w: Lava el coche por favor.

Los incisos p y q sabemos que pueden tomar un valor de falso o verdadero; por lo tanto son proposiciones válidas. El inciso r también es una proposición válida, aunque el valor de falso o verdadero depende del valor asignado a las variables x e y en determinado momento. Sin embargo los enunciados t y w no son válidos, ya que no pueden tomar un valor de falso o verdadero, uno de ellos es un saludo y el otro es una orden.

Existen conectores u operadores lógicos que permiten formar proposiciones compuestas (formadas por varias proposiciones). Los operadores o conectores básicos son:

- Operador and (^)

Se utiliza para unir dos o más proposiciones que se deben cumplir para que se pueda obtener un resultado verdadero:

Ejemplo.

Se tiene el siguiente enunciado "Voy al cine y tengo dinero para la entrada"

Aquí se identifican dos proposiciones atómicas:

p: Voy al cine

q: Tengo dinero para la entrada

De tal manera que la representación del enunciado anterior usando simbología lógica es como sigue:

P ^ q

Para que toda la proposición se cumpla debe pasar que las dos proposiciones atómicas sean verdaderas, en caso de que alguna no se cumpla, por ejemplo, si no tuviera dinero para la entrada no podría ir al cine. A partir de este simple razonamiento, se puede observar su tabla de verdad:

p	q	p ^ q
V	V	V
V	F	F
F	V	F
F	F	F

Donde

V = verdadero

F = falso

- Operador or (v)

Se utiliza para unir dos o más proposiciones en las cuales alguna o ambas pueden ser verdaderas para que el enunciado se cumpla.

Ejemplo:

Sea el siguiente enunciado: "Voy al cine o salgo de compras"

Aquí también se identifican dos proposiciones atómicas:

p: Voy al cine

q: Salgo de compras

p v q: Voy al cine o salgo de compras

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

Con este operador se obtiene un resultado verdadero cuando alguna de las proposiciones es verdadera. Se observa que basta con que una afirmación sea verdadera para que el enunciado completo se verifique. Siguiendo con el ejemplo anterior, si voy al cine pero no de compras basta para que una de los dos enunciados se cumpla, y por lo tanto la proposición completa será verdadera

- Operador Not (~)

Su función es negar la proposición. Esto significa que sí alguna proposición es verdadera y se le aplica el operador not se obtendrá su complemento o negación (falso).

Ejemplo:

Sea el siguiente enunciado: "Voy a estudiar Informática"

Con lo cual se ve que solo existe una proposición atómica:

p: Voy a estudiar Informática

donde ~ p se puede interpretar como:

"no es cierto que voy a estudiar Informática"

"No voy a estudiar Informática"

P	~ p
V	F
F	V

Siguiendo el ejemplo, si voy a estudiar Informática es verdadero, la negación de esta proposición será falsa, pues de hecho estoy afirmando que si lo voy a hacer.

- Operador condicionales (⇨)

Una proposición condicional, es aquella que está formada por dos proposiciones simples (o compuesta) p y q. El operador condicional impone cierta condición a cumplirse para poder verificar la consecuencia. La cual se indica de la siguiente manera:

p ⇨q Se lee "Si p entonces q", "q si p" (condición suficiente), "p solo si q" (condición necesaria),

El enunciado p se denomina antecedente mientras que q se llama consecuente de la implicación o condicional.

Ejemplo:

Sea la siguiente proposición: "Si apruebo el examen entonces te presto el libro"

La implicación está compuesta de las proposiciones:

p: apruebo el examen

q: te presto el libro

Con lo cual quedaría simbolizada como: p ⇨q

Lo que interesa conocer es la verdad o falsedad de la implicación, en relación a la verdad o falsedad de las proposiciones p y q. El enunciado puede pensarse como un compromiso, condicionado por p, y podemos asociar su verdad al cumplimiento del compromiso. Es evidente que si p es falso, es decir si no apruebo el examen, quedo liberado del compromiso y preste o no el apunte la implicación es verdadera.

Si p es verdadera, es decir si apruebo el examen, y no presto el libro, el compromiso no se cumple y la proposición es falsa. Si p y q son verdaderas, entonces la proposición es verdadera pues el compromiso se cumple. Mediante la tabla de verdad se pueden observar los resultados anteriores en forma gráfica.

P	q	p ⇨q
V	V	V
V	F	F
F	V	V
F	F	V

- Proposición bicondicional o doble implicación (⇔)

La doble implicación o bicondicional es la conjunción de dos implicaciones, es decir si tenemos p y q, el bicondicional será: (p ⇨ q ^ q ⇨ p). Ambos enunciados son necesarios para que la doble implicación se cumpla.

Sean p y q dos proposiciones entonces se puede indicar la proposición bicondicional de la siguiente manera:

p ⇔q Se lee "p si solo si q"

Esto significa que p es verdadera si y solo si q es también verdadera. O bien p es falsa si y solo si q también lo es.

Ejemplo: el enunciado siguiente es una proposición bicondicional

"Es buen estudiante, si y solo si; tiene promedio de diez"

Donde:

p: Es buen estudiante.

q: Tiene promedio de diez.

Deben pasar que tanto p como q sean verdaderas para que la doble implicación sea verdadera. En el caso de que ambas sean falsas, se está en la misma situación del conectivo implicación, no existe condición suficiente y necesaria para que el bicondicional no se cumpla, por lo tanto en este caso el resultado será verdadero.

p	q	p ⇔ q
V	V	V
V	F	F
F	V	F
F	F	V

CAPÍTULO II

HERRAMIENTAS DE EXPRESIÓN DE ALGORITMOS

En este capítulo se muestran diferentes herramientas de expresión de algoritmos que se pueden utilizar para representar los problemas implementables por computadoras.

Aunque no se debe hacer una distinción rotunda entre los aspectos prácticos y teóricos de la Informática, es cierto que existen materias que tienen un alto contenido formal, con desarrollos de tipo matemático, contrariamente con otros temas más cercanos a la resolución de problemas de tipo práctico. La Teoría de Autómatas y Lenguajes Formales (Lenguajes Abstractos) trata con las materias del primer tipo y los contenidos que se imparten constituyen el eje fundamental de diversas áreas de conocimiento encuadradas dentro de lo que se podría denominar como Informática Teórica. Por lo tanto, el primer tema a abordar dentro de este capítulo será la descripción de la Teoría de Autómatas y Lenguajes Formales.

Por otro lado, la interfaz en lenguaje natural es quizás, el sueño y el ideal de los usuarios inexpertos en diferentes ámbitos, inclusive fuera de la educación, ya que permite la interacción con la computadora o algún otro medio electrónico haciendo uso de un

lenguaje común o "natural". No se requieren habilidades especiales del usuario cuando se relaciona utilizando lenguaje natural. Es sabido que adquirir habilidades para escribir algoritmos no es una tarea trivial; resulta conveniente que los algoritmos se expresen en lenguaje natural y que se cuente con herramientas adecuadas que facilite su elaboración. Una introducción a los Lenguajes Naturales será el siguiente tema a tratar.

Se plantea también en este capítulo la Visualización de Software, la cual es una de las áreas de las Ciencias de la Computación desde la que se pueden hacer aportes sumamente interesantes relacionados con la enseñanza-aprendizaje de la programación, especialmente con respecto a la lectura comprensiva de algoritmos. Los humanos procesan, por su naturaleza, más fácil y rápidamente las imágenes que el texto. Es decir, adquieren mas información en menos tiempo descubriendo las relaciones gráficas de una imagen que leyendo las letras de un texto. La visualización tiene como meta transformar la información en una más significativa, a partir de la cual el observador humano pueda ganar en comprensión.

En el último punto de este capítulo se desarrolla el tema de Mapas Conceptuales, herramienta de enseñanza – aprendizaje. El objetivo de este punto es presentar una técnica simultáneamente con una estrategia de aprendizaje, un método para captar lo más significativo de un tema y un recurso esquemático para representar un conjunto de significados conceptuales.

2.1 - Lenguajes Abstractos

El estudio de los fundamentos teóricos de la Informática, conocidos como Informática Teórica, es un componente esencial del diseño curricular en las carreras afines. Para poder evolucionar, las ciencias de la computación dependen de que se construyan computadoras, programas y lenguajes que se comporten como sus diseñadores lo desearon. Surge aquí la necesidad de que el profesional en sistemas sea capaz de describirlos en forma precisa y de caracterizar correctamente su comportamiento [Sin].

Se concluye de lo anteriormente descrito que se requiere:

- especificar formalmente programas y lenguajes,
- desarrollar métodos que prueben la corrección de los mismos,

- encontrar formalismos semánticos que provean un sustento matemático para el desarrollo de las teorías básicas sobre el comportamiento de programas, y

- comprender el comportamiento de computadoras, redes de comunicación, etc, a fin de investigar su poder real y determinar los límites de las tareas que puede realizar.

Los puntos definidos precedentemente son áreas dentro de las ciencias de la informática. Para estudiarlas o para contribuir a su desarrollo, es preciso contar con conocimientos de tipo más abstracto y básico. Por ello se presenta en este capítulo una introducción a los modelos abstractos que sirven de fundamento teórico para los lenguajes de programación y máquinas reales.

2.1.1 - Definición de Lenguaje abstracto

Una consideración importante es la distinción entre los lenguajes abstractos o formales (Lenguajes de programación) y los lenguajes naturales (inglés, español, etc). Se puede decir que la diferencia radica en que los lenguajes formales obedecen a reglas preestablecidas y por lo tanto, se ajustan a ellas, no evolucionan y han sido creadas para un fin específico. Sin embargo, los lenguajes naturales existen "per se" y las reglas gramaticales que rigen su estructura han sido desarrolladas con posterioridad a esta última.

Los lingüistas han buscado largamente la manera de establecer formalmente la gramática de los lenguajes naturales como una forma de explicar la habilidad que tienen los humanos para interpretar sentencias que nunca antes hay escuchado. Si bien los estudios han fallado en su cometido de producir gramáticas para los lenguajes naturales, han brindado una cantidad considerable de conocimiento acerca de clases de lenguajes abstractos.

Un lenguaje abstracto es un conjunto de sentencias o secuencias, que satisfacen ciertas propiedades o reglas de construcción. Las sentencias son secuencias finitas de símbolos extraídos de un conjunto denominado alfabeto del lenguaje.

2.1.2 - Gramáticas

Son los formalismos los que pueden abrirnos la puerta del tratamiento informático cómodo y generalizado: dada una secuencia perteneciente a un idioma como el nuestro y un conjunto de propiedades que debe satisfacer, se las debe aplicar para verificar su gramaticalidad. Será necesario imponer una serie de restricciones a los formalismos, si queremos que exista un algoritmo o una secuencia de pasos, que nos garantice la finalización exitosa o no, de la operación en un tiempo finito.

Para llevar a cabo el objetivo de formalizar los lenguajes naturales, habrá que definir una gramática. Esta gramática será un conjunto de reglas que definirán si una secuencia arbitraria de símbolos es correcta. Decimos entonces, que una frase correcta pertenecerá al lenguaje.

La definición de lenguaje formal se basa solamente en sus estructuras sintácticas o gramaticales, ignorándose la estructura semántica.

En 1959, Noam Chomsky clasificó las gramáticas en cuatro tipos de lenguajes y esta clasificación es conocida como la jerarquía de Chomsky, en la cual cada lenguaje es descrito por el tipo de gramática generado. Estos lenguajes sirven como base para la clasificación de lenguajes de programación. Los cuatro tipos son: lenguajes regulares, lenguajes libres de contexto, lenguajes sensibles al contexto y recursivamente enumerables. Dichos lenguajes también se identifican como lenguajes de tipo 3, 2, 1 y 0 respectivamente.

Cada tipo de gramática añade restricciones al tipo inmediatamente superior y la jerarquía va de la más general a la más restrictiva (G3..G0). Cada una de estas gramáticas es capaz de generar un lenguaje. Un lenguaje L se denomina de tipo i (i = 0..3) si existe una gramática G del tipo i capaz de generar o describir ese lenguaje ($G0 \subseteq G1 \subseteq G2 \subseteq G3$).

La idea para aplicar una gramática es que se parte de una variable, llamada símbolo inicial, y se aplican repetidamente las reglas gramaticales, hasta que ya no haya variables en la palabra. En ese momento se dice que la palabra resultante es generada por la gramática, o en forma equivalente, que la palabra resultante es parte del lenguaje de esa gramática.

A continuación se describen dichas las gramáticas [Aut03]:

- **Gramáticas G3 de Tipo 3**

Se denominan gramáticas regulares. Los Lenguajes Regulares asociados a esta gramática son los más simples y restringidos dentro de la jerarquía de Chomsky.

Son de la forma A -> aB o bien A -> a, donde A y B son variables, y a es un carácter terminal.

Ejemplo.- Dadas las siguientes reglas de una gramática para generar el lenguaje de las palabras en { a, b } de longitud par terminadas en 'a':

1) S -> aA

2) S -> bA

3) A -> aB

4) A -> bB

5) A -> a

6) B -> aA

7) B -> bA

se puede generar la palabra 'ba' partiendo del símbolo inicial S y aplicando las reglas 2) y 5), como sigue:

S => bA (2) => "ba" (5)

Como 'ba' tiene solo constantes podemos concluir que la palabra ba es parte del lenguaje generado por la gramática dada.

- **Gramáticas G2 de Tipo 2**

Se conocen como gramáticas libres o independientes de contexto. En ellas una palabra mantiene el mismo significado prescindiendo del contexto en el que se encuentre.

Son de la forma α -> β donde tanto α como β son cadenas de símbolos en donde pueden aparecer tanto constantes como variables.

Ejemplo: Dadas las reglas de una gramática:

1) S -> aSb

2) S -> ab

la palabra "aaabbb" utilizando dicha gramática se puede obtener de la siguiente manera:

S => aSb (1) => aaSbb (1) => "aaabbb" (2)

- **Gramáticas G1 de Tipo 1**

Son las llamadas sensibles al contexto y que cumplen la regla: $\alpha 1 A \alpha 2 \rightarrow \alpha 1 B \alpha 2$, donde el símbolo A solo puede generar B cuando se encuentra rodeado por el "contexto" $\alpha 1 ... \alpha 2$.

Ejemplo: Se puede generar el lenguaje $\{a^n, b^n, c^n\}$ con la siguiente gramática:

1) S -> aBTc

2) T -> ABTc

3) T -> ABc

4) BA -> BX

5) BX -> Y X

6) Y X -> AX

7) AX -> AB

8) aA -> aa

9) aB -> ab

10) BB -> bb

Así, la palabra "aaabbbccc" se deriva de la siguiente manera:

S => aBTc (1) => aBABTcc (2) => aBABABccc (3) => aBXBXBccc (4) =>

⇨ aY XY XBccc (5) => aAXAXBccc (6) => aABABBccc (7) =>

⇨ aABXBBccc (4) => aAY XBBccc (5) => aAAXBBccc (6) =>

⇨ aAABBBccc (7) => aaABBBccc (8) => aaaBBBccc (8) =>

⇨ aaabBBccc (9) => aaabbBccc (10) => aaabbbccc (10)

- **Gramáticas G0 de Tipo 0**

Se denominan gramáticas no restringidas o gramáticas de estructuras de fases, es decir, no hay restricciones ni para el lado derecho ni para el lado izquierdo de las producciones. Esta clase de gramáticas formales tienen una tremenda generalidad.

El conjunto de los lenguaje de tipo 0 coincide con el de todos los lenguajes gramaticales posibles.

2.1.3 - Máquinas abstractas o autómatas

La teoría de los autómatas proviene del campo de la ingeniería eléctrica. Shannon publicó varios trabajos donde demostraba la aplicación de la lógica matemática a los circuitos combinatorios y secuenciales. Posteriormente, sus ideas se desarrollaron para dar lugar a la Teoría de Autómatas. Moore publicó el primer estudio riguroso sobre autómatas, a finales de los años 50, se comenzó a ver la utilidad de los autómatas en relación con los lenguajes [Sin].

Una máquina abstracta o autómata es un dispositivo capaz de recibir y transmitir información. Para realizar esta labor manipula cadenas de símbolos que se le presentan a la entrada produciendo otras tiras o cadenas de símbolos a su salida mediante un conjunto de estados que contienen información necesaria para poder deducir, dado un símbolo de entrada, cuál será el símbolo de salida en cada momento.

A continuación se muestra una breve referencia de los principales modelos de máquinas abstractas [Aut03]:

- **Máquinas de Estados Finitos**

El funcionamiento de los autómatas finitos (AF) consiste en ir pasando de un estado a otro, a medida que van recibiendo los caracteres de la palabra de entrada.

Los AF pueden ser visualizados como dispositivos con los siguientes componentes: una cabeza de lectura y un control (Fig. 2.1).

Los estados son el único medio de que disponen los AF para recordar los eventos que ocurren (por ejemplo, que caracteres se han leído hasta el momento); esto quiere decir que son máquinas de memoria limitada. Esta limitación de los AF los hace finalmente incapaces de distinguir las palabras aceptables de las no aceptables en ciertos lenguajes, más complicados que los lenguajes regulares como por ejemplo las palabras generadas por el lenguaje $\{a^n, b^n\}$.

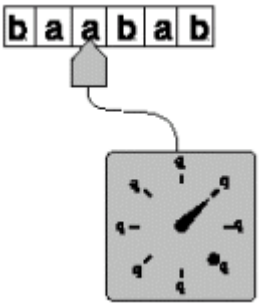


Figura 2.1. Máquina de Estados Finitos

- **Autómata de Pila**

Puesto que los autómatas finitos (AF) no son suficientemente poderosos para aceptar los Lenguajes Libres de Contexto (LLC), cabe preguntarnos qué tipo de autómata se necesitaría para aceptarlos. Una idea es agregar un almacenamiento auxiliar a los AF de manera que se incremente su poder de cálculo, que llamaremos pila, donde se podrán ir depositando, carácter por carácter, cadenas arbitrariamente grandes, como se aprecia en la figura:

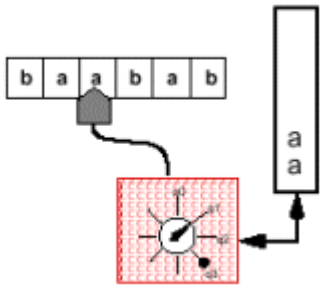


Figura 2.2. Autómata de Pila

Una idea que surge inmediatamente para resolver el problema de las MF con respecto a los lenguaje del tipo $\{a^n, b^n\}$, es la de utilizar la pila como "contador" para recordar la cantidad de a's que se consumen, y luego confrontar con la cantidad de b's.

- **Autómata Linealmente Acotado**

Los lenguajes sensibles al contexto son exactamente los lenguajes aceptados por los autómatas lineales acotados.

Pueden ser visualizadas como dispositivos con los siguientes componentes: una cinta de entrada; una cabeza de lectura/escritura y un control. Donde, la cinta no es infinita, está restringida a una porción de cinta con extremos acotados y además, puede ser modificada en tiempo de ejecución; la cabeza se mueve bidireccionalmente (izquierda y derecha), por lo que puede pasar repetidas veces sobre un mismo segmento de la cinta. (Fig. 3.1.)

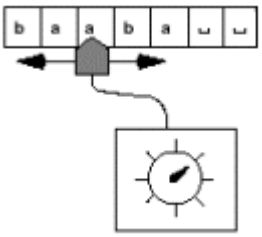


Figura 2.3. Autómata Linealmente acotado

- **Máquina de Tuning**

La primera máquina abstracta de procesamiento de información fue concebida por Turing. Esta máquina, denominada Maquina de Tuning (MT) puede ser vista como un modelo abstracto de un dispositivo físico que posee un número infinito de estados de existencia funcionalmente distinguibles, es un modelo abstracto de la ejecución de la computadora.

Una MT es similar a un Autómata Linealmente Acotado pero posee una cinta infinita (Fig. 10):

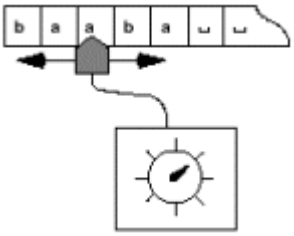


Figura 2.4. Máquina de Tunnig

2.1.4 - Lenguajes Abstractos vs. Autómatas

Existe una conexión entre los conceptos de autómatas y lenguajes abstractos, esta relación directa permite definir un isomorfismo entre ambos. Dado que las gramáticas proporcionan las reglas utilizadas en la generación de las cadenas de un lenguaje, se puede establecer una conexión entre la clase de lenguajes generados por ciertos tipos de gramáticas, y la clase de lenguajes reconocibles por ciertas máquinas. Entonces, se identifica a los lenguajes de tipo 0 con la clase de los lenguajes reconocidos por una Máquina de Turing; a los lenguajes de tipo 1 con los Autómatas Linealmente Acotados; a los lenguajes de tipo 2 con los Autómatas a Pila y a los lenguajes de tipo3 con los Autómatas Finitos.

Al igual que ocurre con las gramáticas, cada tipo de máquina abstracta añade restricciones al tipo de máquina de nivel superior.

Establecidas las reglas de una gramática, una cadena de símbolos pertenecerá al correspondiente lenguaje si tal cadena se ha formado respetando esas reglas. A partir de una gramática se puede construir una máquina reconocedora o aceptadora del lenguaje

generado por esa gramática, de forma tal que cuando reciba una entrada determinada, la máquina indicará si dicha cadena pertenece o no al lenguaje. Una máquina es capaz de reconocer un lenguaje L si es puede distinguir todas las sentencias pertenecientes a L y de no distinguir ninguna cadena no generada por L.

Así se pueden identificar las siguientes relaciones (Fig. 2.5) [Sin]:

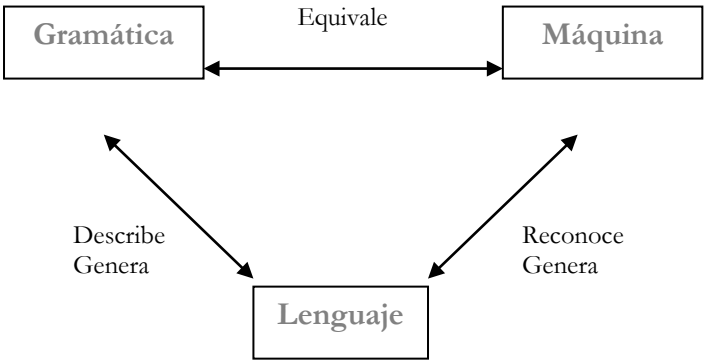


Figura 2.5. Relación entre Lenguaje, Gramática y Autómatas

Dado que una gramática describe formalmente un lenguaje y que este lenguaje puede ser reconocido por una determinada máquina abstracta o autómata, es posible establecer una jerarquía de máquinas abstractas y gramáticas, como se muestra a continuación (Fig. 2.6) [Sin]:

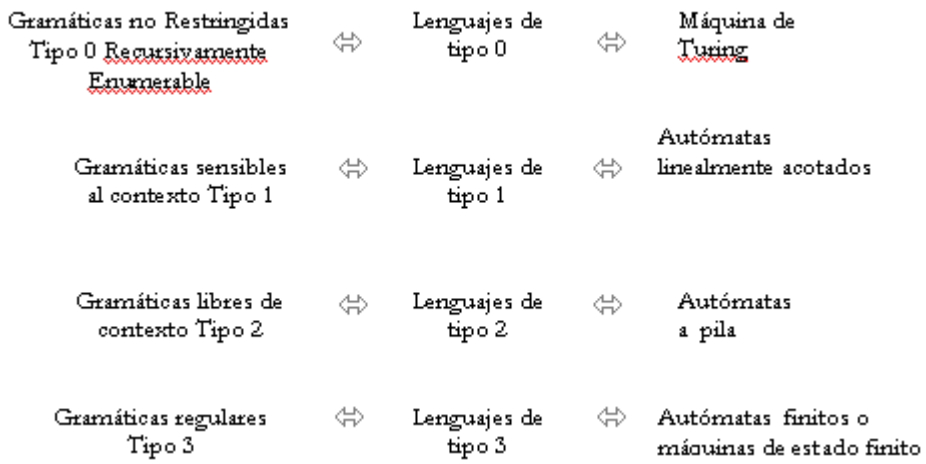


Figura 2.6. Jerarquía de Máquinas abstractas y Lenguajes

La teoría de gramáticas y máquinas teóricas tiene múltiples aplicaciones en diversas áreas de conocimiento.

Algunos ejemplos de las aplicaciones de los lenguajes regulares y los autómatas finitos son los problemas que implican el análisis de cadenas de caracteres como los analizadores léxicos que se utilizan en los compiladores de lenguajes de programación. También se utilizan en los editores de texto para buscar y reemplazar palabras que se equiparan con una expresión regular.

La sintaxis de los lenguajes de programación (Pascal, C, etc.) está descripta mediante gramáticas de Tipo 2 (libres de contexto) aunque también pueden contener algunos aspectos que requieran de una gramática sensible al contexto, con respecto a esta última, es utilizada también para desarrollar lenguajes naturales.

En cuanto a las MT, se corresponden con las gramáticas sin restricciones. La peculiaridad de este tipo de gramáticas es que pueden describir cualquier suceso computable. Un suceso computable es aquél que mediante un procedimiento es capaz de transformar unas determinadas entradas en las salidas requeridas. Las MT pueden usarse como se ha visto para reconocer lenguajes o para calcular funciones. Pero también para generar lenguajes.

Los lenguajes que pueden ser aceptados por una MT son los "recursivamente enumerables" (r.e.). El término enumerable proviene de que dichos lenguajes son aquellos cuyas cadenas pueden ser listadas (enumeradas) por una MT. Hay lenguajes r.e. para los cuales ninguna máquina de Turing que los acepte finaliza con todas las entradas. La subclase de los lenguajes r.e. que son aceptados por al menos una máquina de Turing que se detiene con toda cadena de entrada (independientemente de si la cadena es aceptada o no), se conoce por la clase de los lenguajes recursivos.

2.2 - Lenguajes naturales

El origen de los lenguajes naturales (LN) sucede de manera oral hace unos 100.000 años aproximadamente. Las primeras formas de expresión gráfica no son por cierto tan antiguas, pero, se las puede hacer remontar a las etapas del período paleolítico cuando comienza a verse el empleo de sistemas gráficos. Un paso decisivo en la evolución del homo sapiens fue la adquisición de un vínculo entre el pensamiento y símbolos materiales,

por primera vez el género humano establecía una relación simbólica entre operaciones mentales y símbolos exteriores deliberadamente realizados. De esta manera, para el homo sapiens, el lenguaje representó una ampliación de las facultades cognoscitivas, cuya función permitió expresar su pensamiento y comunicarse entre ellos. Esta función llevada a cabo, primeramente, por medio de señales y vocales (voz), culmina posteriormente con una escritura por signos escritos, para conformar en su totalidad el lenguaje natural.

El lenguaje natural permite designar las cosas reales y razonar acerca de ellas, así como también crear significados. Contrariamente a lo que ciertas teorías lingüísticas formales harían a creer, el lenguaje natural no fue fundamentado sobre una verdad racional a priori, sino que fue desarrollado y organizado a partir de la experiencia humana, en el mismo proceso en que la experiencia humana fue organizada. En su forma actual, los lenguajes naturales tienen un gran poder expresivo y pueden ser utilizados para analizar situaciones altamente complejas y conducir a razonamientos muy sutilmente. La riqueza de su componente semántico y su relación estrecha con los aspectos prácticos de los contextos en los cuales son usados, permite que lenguajes naturales tengan un gran poder expresivo y un valor invaluable como una herramienta para razonamientos profundos.

El Procesamiento del Lenguaje Natural (PLN) es una parte esencial de la Inteligencia Artificial (IA) que investiga y formula mecanismos computacionalmente efectivos que faciliten la interrelación hombre-máquina y permitan una comunicación mucho más fluida y menos rígida que los lenguajes formales y sistemas de menús utilizados tradicionalmente[Pro00].

La comprensión y reconocimiento del lenguaje natural es uno de los problemas más complejos a que se enfrenta la Inteligencia Artificial debido a la complejidad, irregularidad y diversidad del lenguaje humano y a los problemas filosóficos y psicológicos asociados al significado de frases, oraciones y textos en su conjunto.

La concepción principal que orienta el desarrollo de las tecnologías del habla y del lenguaje natural tiene por objetivo estratégico lograr que la interacción con las máquinas sea análoga a la que se ha desarrollado históricamente en la interacción entre las personas. En otras palabras, estratégicamente se plantea como objetivo lograr una relación hombre/máquina análoga a la relación hombre/hombre. En múltiples y diversas aplicaciones el reconocimiento del hablar y el lenguaje natural son de gran utilidad. En ocasiones se presentan problemas en el reconocimiento y comprensión de las expresiones orales y escritas. La escritura, los símbolos, las imágenes y la voz son los medios de

comunicación fundamentales en la interacción hombre/máquina. La experiencia practica acumulada, en el campo del Lenguaje Natural, ha señalado la conveniencia de orientar el reconocimiento del lenguaje natural más al análisis de frases, oraciones y textos en su conjunto, que al reconocimiento de palabras aisladas ya que por ejemplo, una persona que participa en un dialogo sabe como se pueden combinar las palabras para formar una oración, conoce los significados de las mismas, sabe cómo éstas afectan el significado global de la oración y poseen un conocimiento del mundo en general que permite participar de la conversación. En otras palabras, priorizar el reconocimiento del sistema en su conjunto, sobre el reconocimiento de sus partes integrantes.

Una meta fundamental de la IA, es la manipulación de lenguajes naturales usando herramientas de computación, en esta, los lenguajes de programación juegan un papel importante, ya que forman el enlace necesario entre los lenguajes naturales y su manipulación por una maquina.

La riqueza de sus componentes semánticos da a los lenguajes naturales su gran poder expresivo y su valor como una herramienta para razonamiento sutil. Por otro lado la sintaxis de un LN puede ser modelada fácilmente por un lenguaje formal, similar a los utilizados en las matemáticas y la lógica. Otra propiedad de los lenguajes naturales es la polisemántica, es decir la posibilidad de que una palabra en una oración tenga diversos significados.

En un primer resumen, los lenguajes naturales se caracterizan por las siguientes propiedades:

- Desarrollados por enriquecimiento progresivo antes de cualquier intento de formación de una teoría.
- La importancia de su carácter expresivo debido grandemente a la riqueza del componente semántico (polisemántica).
- Dificultad o imposibilidad de una formalización completa.

2.2.1 - El problema de la Ambigüedad

El problema principal del Lenguaje Natural se puede resumir en una palabra:

ambigüedad. Esta se produce cuando una expresión en LN posee más de una interpretación.

Un ejemplo típico en la literatura de procesamiento del LN para la ambigüedad, es a siguiente oración:

Juan vio un mono con un telescopio.

• Acepción 1: El mono tenía un telescopio y Juan lo vio.

• Acepción 2: Haciendo uso de un telescopio, Juan vio a un mono.

En apariencia este problema es demasiado sencillo, pero en realidad, es uno de los mas complicados y que más complicaciones ha dado para que el PLN pueda desarrollarse por completo, ya que al presentarse en todos los niveles del lenguaje, se tienen que desarrollar programas (en lenguaje formal) para solucionarlos en cada caso.

2.2.2 - Niveles de análisis de un PLN

Para continuar nuestro estudio de los lenguajes naturales, es necesario el que conozcamos los niveles del lenguaje, los cuales serán utilizados para la explicación del siguiente tema que es la Arquitectura de un sistema de PLN.

Los niveles de lenguaje que daremos a conocer son los siguientes:

- Nivel Fonológico: trata de cómo las palabras se relacionan con los sonidos que representan.

- Nivel morfológico-lexico: Transforma la secuencia de caracteres de entrada en una secuencia de unidades significativas haciendo uso del diccionario y reglas morfológicas. Trata de cómo las palabras se construyen a partir de unas unidades de significado mas pequeñas llamadas morfemas, por ejemplo: Rápida + Mente == Rápidamente.

- Nivel sintáctico: Analiza la secuencia de unidades léxicas y produce una representación de su estructura (árbol, red, etc.). Trata de cómo las palabras pueden unirse para formar oraciones, fijando el papel estructural que cada

palabra juega en la oración y que sintagmas son parte de otros sintagmas.

- Nivel semántico: A partir de la estructura generada por el proceso sintáctico genera otra estructura o forma lógica asociada que representa el significado o sentido de la sentencia. Trata del significado de las palabras y de cómo los significados se unen para dar significado a una oración, también se refiere al significado independiente del contexto, es decir de la oración aislada.

- Nivel contextual o función pragmática: Utiliza la forma lógica o estructura semántica de la fase anterior para desarrollar la interpretación final de la oración, en función de las circunstancias de contexto. Trata de cómo las oraciones se usan en distintas situaciones y de cómo el uso afecta al significado de las oraciones. Se suele reconocer un subnivel recursivo: discursivo, que trata de cómo el significado de una oración se ve afectado por las oraciones inmediatamente anteriores.

2.2.3 - Aplicaciones del PLN

El procesamiento en lenguaje natural es un área relativamente nueva, las investigaciones están orientadas a conseguir mejoras sustanciales en diversas áreas de análisis tales como: la resolución del sentido de la palabra; el desarrollo de técnicas robustas que permitan la identificación del idioma, reconocer y clasificar entidades y resolver anáforas o correferencias; el desarrollo de sistemas que sean capaces de agrupar e interrelacionar los documentos, generar automáticamente resúmenes de documentos o grupos de documentos, producir una representación conceptual de consultas en cualquier idioma deseado.

A continuación se listan algunas aplicaciones del Procesamiento del Lenguaje Natural:

- Traducción automática: se refiere más que nada a la traducción correcta de un lenguaje a otro, tomando en cuenta lo que se quiere expresar en cada oración, y no solo palabra por palabra. Una aproximación a este tipo de traductores es el Babylon.

- Recuperación de la información: en esta aplicación, un claro ejemplo seria el siguiente: Una persona llega a la computadora y le dice (en LN) que es lo que busca, esta busca y

le dice que es lo que tiene referente al tema.

- Extracción de Información y Resúmenes: Los nuevos programas, deben tener la capacidad de crear un resumen de un documento basándose en los datos proporcionados, realizando un análisis detallado del contenido y no solo la truncando las primeras líneas de los párrafos.

- Resolución cooperativa de problemas: La computadora debe tener la capacidad de cooperar con los humanos para la solución de problemas complejos, proporcionando datos e información, incluyendo también, la demanda de información por parte del ordenador al usuario, debiendo existir una excelente interactividad entre el usuario y el ordenador.

- Tutores inteligentes: La aplicación del PLN en este aspecto, viene siendo más académica, ya que se refiere a la enseñanza asistida por computadora, debiendo esta tener la capacidad de evaluar al educando y la capacidad de adaptarse a cada tipo de alumno.

- Reconocimiento de Voz: Esta es una aplicación del PLN que más éxito ha obtenido en la actualidad, ya que las computadoras de hoy ya tienen esta característica, el reconocimiento de voz puede tener dos posibles usos: para identificar al usuario o para procesar lo que el usuario dicte, existiendo ya programas comerciales, que son accesibles por la mayoría de los usuarios, ejemplo: ViaVoice.

2.2.4 - Arquitectura de un sistema de PLN

Como va la computadora a interpretar y analizar las oraciones que le sean proporcionadas:

- El usuario le expresa a la computadora que es lo que desea hacer.

- La computadora analiza las oraciones proporcionadas, en el sentido morfológico y sintáctico, es decir, si las frases contienen palabras compuestas por morfemas y si la estructura de las oraciones es correcta.

- El siguiente paso, es analizar las oraciones semánticamente, es decir saber cual es el significado de cada oración, y asignar el significado de estas a expresiones

lógicas (cierto o falso).

- Una vez realizado el paso anterior, ahora podemos hacer el análisis pragmático de la instrucción, es decir una vez analizadas las oraciones, ahora se analizan todas juntas, tomando en cuenta la situación de cada oración, analizando las oraciones anteriores, una vez realizado este paso, la computadora ya sabe que es lo que va a hacer, es decir, ya tiene la expresión final.

- Una vez obtenida la expresión final, el siguiente paso es la ejecución de esta, para obtener así el resultado y poder proporcionárselo al usuario.

2.2.5 - El PLN en los Sistemas Multimedia y Expertos

La pretensión de la informática de adaptarse al comportamiento natural de del usuario, ha llevado la incorporación de texto, imágenes y sonido (los llamados entornos multimedia) a las estaciones de trabajo y Pc´s actuales, al tiempo que éstos aumentan su capacidad.

Antes de continuar con el estudio del PLN en los sistemas multimedia, enumeremos las partes que pueden contener estos:

- Entornos de iconos
- Autopistas de información
- Ratón
- Programación interactiva
- Realidad Virtual
- Hipertexto
- Sonido

En si la multimedia es la unión del hipertexto con el sonido, estas uniones de imágenes, texto y sonidos necesitan una filosofía del conocimiento que fundamente su función interna dentro de la comunicación de conocimientos, o sea pasar a ser elementos

de la estructura de conocimiento y no solo datos. A esa filosofía la llamamos idea intuitiva de la comunicación sistema-usuario. Para deslindar, este problema, lo situamos en un contexto de enseñanza de conocimientos, es decir, Tutores Inteligentes (TI) en entornos multimedia. Los TI son un tipo de sistemas expertos con módulos especiales (alumno y pedagógico) y una reestructuración de los existentes (conocimiento, explicaciones, interfaz).

En si el modelo que debe seguir el TI es el siguiente:

Como es sabido un dialogo en lenguaje natural esta muy afectado por el conocimiento que un interlocutor tiene del otro y por el contexto o entorno donde el dialogo tiene lugar. Lo primero está aquí proporcionado por el modelo del usuario y, lo segundo, por el "paisaje de imágenes".

2.3 - Visualización

Es frecuente el uso de metáforas visuales en la descripción de los procesos cognitivos. Cuando algo es entendible se suele decir que se ve. Se trata de aclarar las ideas, de acomodarlas, de enfocar el pensamiento. Lo apropiado y usual de estas metáforas dan la pauta del nexo existente entre lo que se ve y lo que se piensa. La relación entre las acciones mentales y las percepciones externas no es accidental.

Una clase importante de estímulo externo son los gráficos en todas sus variantes. Son un medio valioso en el logro de dos objetivos relacionados pero diferentes al mismo tiempo:

- Su empleo en la comunicación de una idea (un dibujo vale más que mil palabras). Transmitir una idea requiere, por supuesto, tener una idea que transmitir.

- El uso de medios gráficos para crear o descubrir la idea en sí misma: Usar las propiedades especiales de la percepción visual para resolver problemas.

Se requiere un mayor entrenamiento específico para leer un libro que para contemplar y apreciar una pintura, aunque la forma de construcción de la imagen puede ser tanto o más compleja que la confección de un texto escrito. Sin embargo, la eficiencia comunicacional se mide desde la percepción del usuario y no desde la complejidad de las

herramientas utilizadas.

La representación visual le facilita al usuario la utilización de complejas inferencias perceptuales ya implementados en su corteza visual. El diseño adecuado de herramientas de visualización permite simplificar la búsqueda de información, mejorar las posibilidades de detección de patrones, aumentar los recursos accesibles por los usuarios, entre otros. [Car99].

Con el fin de satisfacer las necesidades de la persona que interactúa con las vistas resultantes de la visualización, todo lo informado a través de la misma debe tener en cuenta aspectos de la percepción y del conocimiento humano [Gri99]. Hay una variedad enorme de aportes sensitivos que pueden favorecer la formación de un cuadro mental. Con tal propósito, la visualización debe buscar estructuras, características, anomalías y relaciones entre los datos objeto de la visualización, presentar una visión global cuando se trata de conjuntos grandes y complejos de datos, y detectar las zonas de interés que merecen un análisis cualitativo.

2.3.1 - Visualización de software

La Visualización de información en general consiste en el uso de recursos gráficos, de animación y multimediales con importante interacción entre el usuario y la computadora. Específicamente, la Visualización de Software, que usa recursos similares a la de información, tiene como finalidad facilitar la comprensión y el uso efectivo de programas [Mor04].

La Visualización de Software comprende la Visualización de Algoritmos y la de Programas. La primera consiste en la visualización de abstracciones de alto nivel que describen el algoritmo, mientras que la segunda se refiere al código real de programa y a las estructuras de datos. Ambas pueden darse en forma estática o dinámica. La animación de algoritmos muestra la conducta del programa en ejecución, mientras que la visualización de código puede incluir algún tipo de mejoramiento de la impresión como indentado o estructura del programa en forma estática o destacando las líneas de código a medida que ellas están siendo ejecutadas, dinámicamente [Sta98].

Las visualizaciones de software, en general, tienen valor pedagógico en sí mismas ya que producen entusiasmo y son motivadoras para los estudiantes. Los docentes saben que

atraen su interés, lo que resulta de ayuda invalorable en el proceso de enseñanza – aprendizaje. El objetivo es apuntar a una mayor comprensión y a un uso efectivo del software.

2.3.2 - Lenguajes de Programación Visual

Un lenguaje de programación visual manipula información visual o soporta interacción visual, o permite la programación con expresiones visuales [Mye90].

Los ambientes de programación visual proveen elementos gráficos o icónicos que pueden ser manipulados por el usuario en forma interactiva de acuerdo a alguna gramática espacial específica para la construcción de programas [Gol90].

2.3.3 - Características de los Lenguajes de Programación Visual

Hay muchos argumentos a favor de la programación visual. Generalmente están centrados en el hecho de que los humanos procesan, por su naturaleza, más fácil y rápidamente las imágenes que el texto. Es decir, adquieren mas información en menos tiempo descubriendo las relaciones gráficas de una imagen que leyendo las letras de un texto [Rae85].

Algunos de estos argumentos son enumerados a continuación:

- Las imágenes permiten acceso no secuencial a cualquier contenido de interés para el aprendiz.
- Las imágenes permiten una visión global o detallada indistintamente, dependiendo de lo que espere encontrar el observador.
- El sistema sensorial humano esta diseñado para el procesamiento de imágenes en tiempo real. Por esa razón, las imágenes pueden ser accedidas y decodificadas mas rápidamente.
- El texto es unidimensional por naturaleza, mientras que las imágenes son multidimensionales.

- Las imágenes proveen un lenguaje mucho más rico a través de sus propiedades visuales. Como los colores, las formas, los tamaños. Esto conduce a una codificación de la información significativamente mas compacta que la del texto

- Las imágenes pueden capturar más fácilmente una idea abstracta.

- El texto es una secuencia de palabras y símbolos de puntuación. A su vez, cada palabra es una secuencia de letras. Y cada letra no es otra cosa que una pequeña imagen que representa sencillamente a un único símbolo de todo el texto.

A partir de numerosos estudios se ha concluido que un lenguaje de programación visual adecuado es potencialmente mejor que cualquier lenguaje textual.

Por otra parte los lenguajes visuales tienden a utilizar una notación relativamente dispersa. Esto puede significar que utilizan más espacio de la pantalla que los lenguajes textuales. Esto puede ser aliviado utilizando el concepto de abstracción procedural. Es decir, representando a un subdiagrama con un único símbolo simple, y tratándolo como a una "caja negra" [Cap97].

Muchos lenguajes visuales son interpretados y, en gran parte de los casos, el intérprete opera directamente sobre las representaciones de los componentes del diagrama. Obviamente, esto resulta en una eficiencia que deja mucho que desear. Además, muchos de ellos son latentemente tipados, lo cual significa que la verificación de que un operador reciba valores de tipos adecuados, sólo se puede realizar cuando dicho operador es aplicado, es decir, en tiempo de ejecución.

2.3.4 - Interfaz de usuario

El entorno de un lenguaje visual debe contemplar el nivel cognitivo, la capacidad de percepción, de respuesta, de abstracción, de los usuarios a los que va dirigido, como así también, la capacidad de interacción hombre-máquina, la velocidad de presentación, la estandarización de colores y sonidos, etc. Para ello se debe disponer de sistemas específicos, amigables con el usuario, que no actúen como elementos de dispersión en la elaboración y en la interpretación de la conducta del software propiamente dicho, de manera de no entorpecer su trabajo de abstracción [Teo99].

Usualmente se utilizan los términos diálogo (o interacción) hombre-computadora e interfaz del usuario en forma indistinta, pero no son dos conceptos análogos. El termino hombre-computadora hace referencia a la comunicación que existe entre el usuario y el sistema de computación mientras que la interfaz del usuario denota el medio por el cual tal comunicación se realiza.

El diseño de interfaces de usuario es una tarea que ha adquirido relevancia en el desarrollo de un sistema. La calidad de la interfaz de usuario puede ser uno de los motivos que conduzca al éxito o al fracaso.

La Interfaz de Usuario, en adelante IU, de un programa es un conjunto de elementos hardware y software de una computadora que presentan información al usuario y le permiten interactuar con la información y con la computadora. También se puede considerar parte de la IU la documentación (manuales, ayuda, referencia, tutoriales) que acompaña al hardware y al software.

Si la IU está bien diseñada, el usuario encontrará la respuesta que espera a su acción. Si no es así puede ser frustrante su operación, ya que el usuario habitualmente tiende a culparse a sí mismo por no saber usar el objeto.

El usuario tiene su visión personal del sistema, y espera que éste se comporte de una cierta forma. Se puede conocer el modelo del usuario estudiándolo, ya sea realizando tests de usabilidad, entrevistas, foros, o a través de una realimentación. Una interfaz debe facilitar el proceso de crear un modelo mental efectivo. Para ello son de gran utilidad las metáforas, que asocian un dominio nuevo a uno ya conocido por el usuario. Un ejemplo típico es la metáfora del escritorio, común a la mayoría de las interfaces gráficas actuales.

Una interfaz no es simplemente una "cara bonita"; esto puede impresionar a primera vista pero decepcionar a la larga. Lo importante es que el programa se adapte bien al modelo del usuario, cosa que se puede comprobar utilizando el programa más allá de la primera impresión.

- **Objetivos de la interfaz de usuario**

  - Flexibilidad de diálogos: debe haber dialogo sin autoritarismo por parte del sistema.

  - Facilidad de aprendizaje y utilización

- Confiabilidad
- Comportamiento del sistema transparente al usuario

➢ **Componentes de una interfaz**

La mayoría de los modelos existentes son descripciones estructurales que especifican la manera que la interfaz del usuario se relaciona con el sistema de aplicación, no son modelos descriptivos de la propia interacción hombre-computadora. Entre estos modelos se encuentra el de Sheeim, mas tarde modificado por Green (1985). Este modelo se muestra en la siguiente figura 2.7:

Figura 2.7. Modelo de Sheeim

La componente de presentación es la responsable de la visualización de la interfaz, especifica las pantallas que se presentaran al usuario y el estilo de interacción. Los colores, ubicaciones de la pantalla, figuras, menúes son conceptos relacionados con esta componente.

La componente de control de dialogo esta a cargo del manejo de la estructura, secuencia, eventos y del control de las técnicas de interacción provistas. Define la estructura, secuencia, el proceso del intercambio de información entre el usuario y la aplicación.

Finalmente, la componente denominada modelo de interfaz-aplicación es la representación de la aplicación desde el punto de vista de la interfaz. Contiene tanto una

visión de la aplicación desde la interfaz como una visión de la interfaz desde la aplicación. Abstrae por un lado, la forma en que el usuario consulta y provee información a la aplicación y, por otro, independiza las acciones de la aplicación de las implementaciones particulares. Especifica la relación entre los eventos de la interfaz y las funcionalidades de la aplicación. La comunicación entre ambas partes del software se realiza vía llamados a procedimientos y mediante el uso de estructuras de datos descriptas en un nivel de abstracción independiente de la implementación.

- **Principios de Nielsen**

Estos principios proveen una ayuda para localizar problemas en la UI y pueden aplicarse tanto a las interfaces orientadas a texto como a las orientadas a gráficos, ya que existe una correspondencias entre el lenguaje textual y el visual.

Los nueve principios básicos a seguir para el desarrollo de interfaces de usuario son los que se detallan a continuación [Nil]:

1) Utilizar diálogo simple y natural.
    - Se deben seguir las reglas usuales en el uso de mayúsculas
    - No abusar de las abreviaturas
    - Evitar errores de tipeo
    - No mostrar información innecesaria
    - Indicar de que forma se quiere que ingrese información el usuario
    - Agrupar información relacionada
    - No usar el orden más adecuado para la computadora debido a la conveniencia de datos internos.
    - No usar teclas de función predefinidas dispersas
    - Proporcionar ayuda general de la interface
2) Emplear el lenguaje del usuario

- Utilizar palabras y frases familiares al usuario
- No usar palabras extranjeras
- No usar vocablos de la jerga informática
- Usar alfabeto local
- No truncar datos de manera que el usuario no encuentre luego otro distinto al que ingreso
- No excederse en la información técnica
- Nivel adecuado de usuario, ni excesivo para usuarios experimentados ni pobre para novatos
- No requerir ingreso de información desde el punto de vista del sistema (Ej. : 'Ingrese dato')

3) Minimizar la experiencia de memoria del usuario

- Las tareas deben ser inducidas fácilmente por el sistema, proveer ayuda
- Evitar desorientación
- Evitarle al usuario operaciones innecesarias para realizar una operación o tarea específica.

4) Brindar consistencia

- No usar distintos términos para el mismo concepto
- No usar un mismo termino para distintos conceptos
- Mantener coherencia entre tipo y ubicación de errores

5) Proveer Feedback (retroalimentación)

- Informar el estado de la aplicación cuando los tiempos de respuesta son muy largos

- Informar proceso que esta en curso
- Informar de las siguientes alternativas
- Informar si la tarea se realizo exitosamente

6) Proveer salidas evidentes
   - Proveer la opción de cancelar en todos los entornos y debe ser fácil de ubicar
   - Permitir cancelar acciones en ejecución
   - Permitir modificar datos ingresados previamente
7) Proveer short-cuts
8) Proveer buenos mensajes de error
   - No informar códigos de errores en lugar de motivo del error
   - Los mensajes no deben intimidar al usuario
   - Los mensajes no deben ser muy vagos
   - Informar al usuario porque una entrada es insatisfactoria
   - Sugerir acciones a tomar
   - Evitar mensajes que no proveen información de interés
9) Prevención de errores
   - No discriminar mayúsculas y minúsculas en el ingreso de datos. Ej. : 'Si' y 'SI'
   - Distinguir entre el '1' (numero uno) y la 'l' (letra ele), y el '0'(cero) y la letra 'O'
   - No exigir paréntesis ni signos de puntuación
   - Diferenciar entre comando de entrada ('shift' + letra) y una tecla de función
   - No mantener el nivel de explicación demasiado general

- Mostrar posibles entradas para ser seleccionadas

Otros principios que se pueden agregar:

10) Documentación

- Facilitar exploración a través de caminos principales y secundarios que permitan profundizar en caso que se desee.

11) Minimalismo

- Permitir el uso inmediato productivo

2.4 - Mapas Conceptuales

Los Mapas Conceptuales (MC) iniciaron su desarrollo en el Departamento de Educación de la Universidad de Cornell, EUA, durante la década de los setenta y constituyen una respuesta a la teoría del aprendizaje significativo desarrollada por Ausubel. Sin embargo, se trata de una técnica desarrollada a mediados de la década de los setenta por Joseph Novak y sus colaboradores en la Universidad de Cornell, en los Estados Unidos. Ausubel nunca habló en su teoría de mapas conceptuales [PRO 1] [SIS 1].

La técnica de los MC es herramientas de enseñanza-aprendizaje que cada día se utiliza más en los diferentes niveles educativos. Involucra la asimilación de conceptos y proposiciones nuevas mediante la inclusión en estructuras cognitivas ya existentes significativas, que se conoce como aprendizaje significativo. Las nuevas ideas, conceptos y proposiciones pueden ser aprendidos en la medida en que otras ideas, conceptos o proposiciones relevantes estén adecuadamente claras y disponibles en la estructura cognitiva del individuo y funcionen como un punto de "anclaje" a las primeras.

Se define el MC como una técnica que representa, simultáneamente, una estrategia de aprendizaje, un método para captar lo más significativo de un tema y un recurso esquemático para representar un conjunto de significados conceptuales, son diagramas de significados, de relacionales significativas, incluidos en una estructura de proposiciones y que requieren una compresión acabada de los contenidos[PER99]. Los MC proporcionan un resumen esquemático de lo aprendido [ONT97].

El MC aparece como una herramienta de asociación, validación, interrelación, discriminación, descripción y ejemplificación de contenidos, con un alto poder de visualización.

Los Mapas Conceptuales constan de los siguientes elementos básicos que se relacionan gráficamente y forman cadenas semánticas, es decir, cadenas que poseen un significado:

- Los conceptos: También llamados nodos, hacen referencia a hechos, objetos, cualidades, características, etc., gramaticalmente los conceptos se pueden identificar como nombres, adjetivos y pronombres. Los conceptos son abstracciones mentales que el sujeto tiene de los objetos reales y que existen en su mundo cognitivo

- Las palabras-enlace: Son palabras que unen los conceptos y señalan los tipos de relación existentes entre ellos.

- Las proposiciones: Están constituidas por conceptos y palabras-enlace. Es la unidad semántica más pequeña que tiene valor de verdad.

2.4.1 – Características de los Mapas Conceptuales

- Jerarquización: Los conceptos están dispuestos por orden de importancia o de "inclusividad". Solo aparece una vez un mismo concepto. Las líneas de enlace pueden terminar en una flecha para indicar el concepto derivado.

- Selección: Son una síntesis o resumen que contienen lo más significativo de un tema. Se pueden elaborar sub-mapas: que vayan ampliando diferentes partes o subtemas del tema principal

- Impacto Visual: "Un buen mapa conceptual es conciso y muestra las relaciones entre las ideas principales de un modo simple y vistoso, aprovechando la notable capacidad humana para la representación visual." [NOV88].

Los MP apoyan los cuatro procesos básicos para la codificación de información:

- Selección: al elaborar un mapa conceptual, se tiene que seleccionar la

información que se va a emplear.

- Abstracción: se extraen los elementos más significativos para ubicarlos dentro del mapa.
- Interpretación: Para favorecer la comprensión, se hacen inferencias sobre las ideas que se tengan.
- Integración: Cuando se crea un nuevo esquema o se modifica uno existente.
- Y la recuperación de información, cuando se trata de comprender lo que trae el esquema.

2.4.2 - Tipos de Mapas Conceptuales

Los cinco tipos principales de MC se describen a continuación [KID 1]:

- **Mapa en forma de araña**

El mapa es estructurado de manera que el término que representa al tema principal es ubicado en el centro del gráfico y el resto de los conceptos llegan mediante la correspondiente flecha (Fig. 2.8).

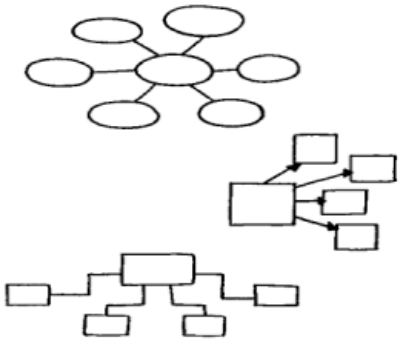


Figura 2.8. Mapa en forma de araña

- **Mapa Jerárquico**

Por refinamientos sucesivos, la información se representa en orden descendente de importancia o de acuerdo a niveles de generalización. El concepto más importante es situado en la parte superior del mapa o el mas general e inclusivo y el menos importante o mas especifico en la base(Fig. 2.9).

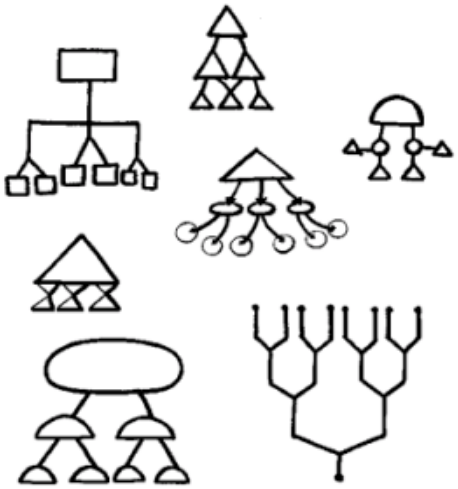


Figura 2.9. Mapa Jerárquico

- **Mapa Secuencial**

En este tipo de mapa los conceptos son colocados uno detrás del otro en forma lineal (Fig. 2.10).

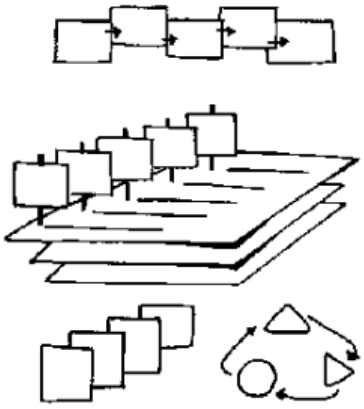


Figura 2.10. Mapa Secuencial

- **Mapa en sistema**

En este tipo de mapa la información se organiza también de forma secuencial pero

se le adicionan entradas y salidas que alimentan los diferentes conceptos incluidos en el mapa.

- **Mapas Hipermediales**

Es aquel que en cada nodo de la hipermedia contiene una colección de no más de siete conceptos relacionados entre sí por palabras-enlaces). Son el resultado de combinar MC con la tecnología hipermedial (sonido, imagen, animación, video, etc.)

Puesto que se produce más fácilmente un aprendizaje significativo cuando los nuevos conceptos o significados conceptuales se engloban bajo otros conceptos más amplios, más inclusivos, los mapas conceptuales jerárquicos son los mas recomendados

2.4.3 - Cómo construir un Mapa Conceptual

Para construir un MC se deben seguir las siguientes indicaciones [MAP 1]:

- Identificar los conceptos claves
- Ordenar los conceptos poniendo el (los) más general (es) en el tope del mapa y gradualmente colocar los demás hasta completar el mapa según el modelo de la diferenciación progresiva.
- Conectar los conceptos con líneas y rotular las líneas con una o más palabras claves que definan la relación entre los conceptos.
- Buscar relaciones horizontales y cruzadas.
- Ejemplos específicos pueden agregarse al mapa debajo de los conceptos correspondientes. En general, los ejemplos quedan en la parte inferior del mapa.
- En general, el primer intento de mapa tiene una simetría pobre y algunos conceptos o grupos de ellos están mal ubicados respecto a otros que están más estrechamente relacionados. Reconstruir el mapa es útil en ese caso.
- Quizás en ese punto ya se puede imaginar otras maneras de hacer el mapa. A

medida que cambia la comprensión de las relaciones entre los conceptos, el mapa también cambia. Es aconsejable reelaborar el mapa al menos una vez.

2.4.4 – Ventajas de los Mapas Conceptuales

Se puede concluir que los MC tienen una gran importancia en el aprendizaje ya que:

(a) Facilitan una rápida visualización de los contenidos de aprendizaje.

(b) Favorecen el recuerdo y el aprendizaje de manera organizada y jerarquizada.

(c) Permiten una rápida detección de los conceptos clave de un tema, así como de las relaciones entre los mismos, así como también permite una visión global de los mismos lo que permite una visión en conjunto entre ellos y al mismo tiempo resalta las particularidades.

(d) Favorece el desarrollo del pensamiento lógico.

(e) Los materiales elaborados utilizando MC facilitan el estudio independiente.

(f) Permiten que el alumno pueda explorar su conocimiento previo acerca de un nuevo tema, así como para la integración de la nueva información que ha aprendido.

(g) Organiza los conocimientos a partir de las principales relaciones entre los conceptos.

(h) Favorecen el trabajo colaborativo.

(i) Los mapas conceptuales proporcionan un resumen esquemático de todo lo que se ha aprendido.

(j) Los mapas conceptuales son una estrategia para organizar visualmente el conocimiento a aprender, pudiéndose aprovechar las facilidades que brindan los mismos en la computadora.

(k) Pueden ser utilizados para retomar los conceptos básicos y fundamentales de un contenido teórico

(l) Favorecen la autoevaluacion

(m) Permiten visualizar ordenamientos, niveles, jerarquías

(n) Facilitan la compresión

(o) Es un poderoso instrumento para que el alumno aprenda a escribir de forma ordenada.

Debe destacarse que los MC dirigen la atención, tanto del estudiante como del profesor, sobre el número de ideas importantes en las que deben concentrarse en cualquier tarea específica de aprendizaje. Provee a los profesores y alumnos de una forma rica para organizar y comunicar lo que se sabe.

Los mapas permiten que el profesor/a pueda utilizarlo para:

- Realizar una presentación inicial del tema o de la unidad
- Establecer unos límites en los conceptos y relaciones del tema
- Elaborar una visión global y completa al finalizar el desarrollo de la unidad.
- Evaluación y seguimiento del aprendizaje del alumno

Permiten a profesores y alumnos intercambiar sus puntos de vista sobre la validez de un vínculo proposicional determinado, o darse cuenta de las conexiones que faltan entre los conceptos y que sugieren la necesidad de un nuevo aprendizaje. Fomenta la negociación, al compartir y discutir significados

El alumno aprende con la utilización del mapa a tomar conciencia de sus conocimientos previos, a organizar la nueva información relacionándola con la de temas anteriores, y a elaborar resúmenes y síntesis diferenciando lo fundamental de lo accesorio; todo ello se convierte en una herramienta muy potente para facilitar el recuerdo de todo lo que debe conocer. Permite lograr un aprendizaje interrrelacionado, al no aislar los conceptos

Parece ser un buen método para estimular el aprendizaje significativo entre estudiantes con diferente preparación académica,

Diversos autores sugieren que los estudiantes que hacen o analizan MC, tendrán un

conocimiento base amplio y, por lo tanto, estarán más preparados para resolver problemas en comparación con aquellos estudiantes que han aprendido por memorización.

La inserción de los Mapas Conceptuales como herramienta importante para crear, compartir, organizar, sintetizar, almacenar y evaluar el conocimiento, contribuirá a que los estudiantes aprendan a aprender, aprendan haciendo, aprendan más y más eficientemente.

2.4.5 – Desventajas de los Mapas Conceptuales

En la construcción de un MP se debe evitar:

- Elaborar un esquema o diagrama de flujo en lugar de un mapa conceptual, en donde en lugar de presentar relaciones supraordenadas y combinatorias entre conceptos, se presentan meras secuencias lineales de acontecimientos.

- Relaciones entre conceptos excesivamente confusas. Es decir, con muchas líneas y palabras de enlace.

- Un mapa conceptual es más claro si está bien distribuido y presentado armónica y equilibradamente.

- Como el aprendizaje significativo implica necesariamente la atribución de significados idiosincráticos, no existe un mapa conceptual "correcto". Un profesor nunca debe representar a sus alumnos el mapa conceptual de cierto contenido sino un mapa conceptual para ese contenido de acuerdo con los significados que él atribuye a los conceptos y a las relaciones significativas entre ellos, exigir un mapa correcto, promoverá (como muchos otros recursos instruccionales) el aprendizaje mecánico en detrimento del aprendizaje significativo. De la misma manera, nunca se debe esperar que el alumno presente en una evaluación el mapa conceptual "correcto" de un cierto contenido. Eso no existe. Lo que el alumno presenta es su mapa y lo importante no es si ese mapa está correcto o no, sino si da evidencias de que el alumno está aprendiendo significativamente el contenido. Además, los MC no son autoexplicativos. Los mapas conceptuales deben ser explicados por quien los hace; al exponerlos, la persona externaliza significados.

2.4.5 - Herramientas disponibles

El uso de mapas conceptuales constituye una novedad para la visualización automática de programas. A continuación se detallan algunas herramientas:

Camp Tolls, desarrollada por el Institute for Human and Machine Cognition asociado a la West Florida University en USA.

Existe también una herramienta libre creada en la Universitat Pompeu Fabra de Barcelona por el profesor Cristófol Rovira que genera automáticamente el código necesario para incluirlo en formato XML usando el standard Topic Maps .

Otra herramienta es SVED (: Sistema de Visualización de Estructuras de Datos) orientado a las estructuras de Datos y que permite la animación de dichas estructuras mostrando el comportamiento de las mismas durante la ejecución de un programa ya creado.

CAPÍTULO III

ANÁLISIS DE VISUAL DA VINCI

El ambiente Visual Da Vinci (VDV) desarrollado en la Facultad de Informática de la Universidad Nacional de La Plata, ha sido utilizado sistemáticamente durante los últimos 10 años en los cursos preuniversitarios y en las primeras etapas del curso de Algorítmica y Programación de varias universidades del país. [DeG03].

La idea de emplear el VDV está fundada en el hecho de que se trata de una herramienta restringida en cuanto al tipo de datos, las estructuras de control y el conjunto de instrucciones, comparada con cualquier lenguaje de programación visual de los disponibles en el mercado, lo que evita la distracción propia de un número muy grande de posibilidades y disminuye los tiempos de aprendizaje del entorno, permitiendo a los que lo utilizan concentrarse en muy poco tiempo en el diseño de algoritmos y su expresión.

Visual Da Vinci tiene como base el desarrollo realizado por un grupo de investigadores del LIDI (Laboratorio de la Facultad Informática de la UNLP). En el mismo diseñaron las especificaciones de una máquina abstracta (el robot Lubo-I) junto con un lenguaje de programación asociado. El objetivo de esta investigación fue destinarla a la

enseñanza de los conceptos básicos la de programación estructurada [DeG88] [DeG89].

Esas especificaciones se están utilizando desde aquel entonces en el curso de ingreso a la carrera de Informática y en la materia curricular de primer año Programación de Computadoras de la UNLP.

Luego de haberse implementado, podría decirse, en forma abstracta, pues no existía entorno en el cual los alumnos pudieran verificar la validez de sus programas, surgió entonces la necesidad de desarrollar un ambiente que permita implementar esta máquina abstracta con su respectivo lenguaje de programación, al cual se lo denominó Visual Da Vinci [Cha97].

3.1 – Generalidades del robot Lubo-1

El robot puede realizar una serie de acciones esenciales sobre una ciudad de cien calles por cien avenidas. Una esquina esta delimitada por la intersección de una calle con una avenida, y se identifica por medio de un par ordenado (Av, Ca), donde Av y Ca son números en el rango 1..100.

Inicialmente, el robot es posicionado en la esquina determinada por el par (1, 1) orientado hacia el norte. Dicha posición puede ser modificada de forma tal que quede orientado hacia uno de los cuatro puntos cardinales.

Cada vez que el robot se mueve esto lo conduce de la esquina donde esta situado a la siguiente, según la dirección hacia la cual se encuentre orientado.

En cada esquina puede haber objetos - flores y papeles - , los cuales pueden ser tomados. Para ello el robot lleva consigo dos bolsas, una de flores y otra de papeles, en cada una de ellas puede tanto depositar (siempre que la bolsa de la cual quiera sacar un elemento no este vacía) o tomar (siempre que la esquina no este vacía) objetos.

En la ciudad puede haber obstáculos que impiden al robot continuar con su recorrido lineal. Los obstáculos pueden ser simples o barreras. Un obstáculo es simple si no hay otro en ninguna de las esquinas adyacentes. Un conjunto de obstáculos forman una barrera si están dispuestos en línea recta sobre esquinas consecutivas.

El lenguaje que se utiliza para la programación del robot es estructurado y cuenta

con las construcciones mínimas necesarias para el aprendizaje del paradigma de programación.

Todo programa escrito en dicho lenguaje está compuesto por tres partes: encabezado, declaraciones y cuerpo. El encabezado determina el nombre del programa a través de un identificador. La segunda parte está destinada a las declaraciones de subprogramas y variables. La declaración de una variable determina su nombre y su tipo. Los subprogramas tienen la misma estructura general que el programa salvo por los parámetros formales.

El cuerpo del programa principal, al igual que el de los subprogramas, es una secuencia de sentencias. Tales sentencias pueden ser primitivas, simples o compuestas. Las primitivas son aquellas directamente ejecutables por el robot. Las sentencias simples son las asignaciones y las invocaciones a subprogramas definidos por el usuario o del sistema. Las compuestas son las estructuras de control selección, iteración condicional e iteración incondicional.

Las expresiones son formadas a partir de valores, variables definidas por el usuario o del sistema, parámetros de entrada o parámetros de entrada/salida, conectados con operadores lógicos, aritméticos y relacionales.

3.2 – Características particulares de Visual Da Vinci

El ambiente de Visual Da Vinci esta integrado por ventanas dispuestas en forma aleatoria, las cuales se pueden mover libremente dentro y fuera del entorno, es decir, se puede tener a la vista el editor de código y, por ejemplo, un procesador de texto.

Visual Da Vinci esta compuesto por: la ventana principal, el editor de diagramas, el editor de código, la ciudad y el inspector de variables del sistema.

La ventana principal es la encargada de la funcionalidad general en el manejo de archivos, permitiendo crearlos, abrirlos, guardarlos, compilarlos y ejecutarlos.

La especificación visual de programas se realiza por medio de la confección de un diagrama del estilo del Nassi-Schneiderman [Nas73] [You93] en el editor de diagramas (Fig. 3.1).

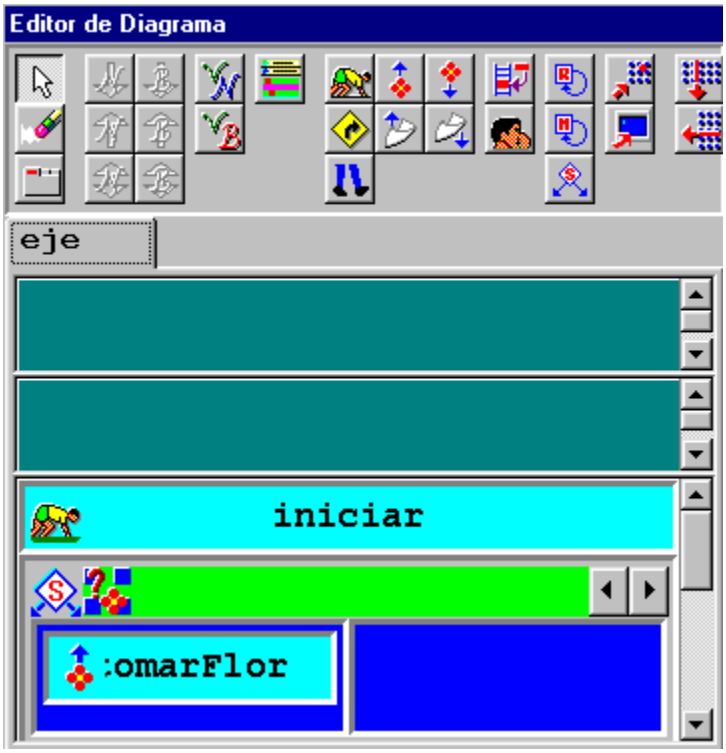


Fig. 3.1. Editor de Diagrama de Visual Da Vinci

Se puede observar una solapa – página – denominada programa, allí se ubicarán las instrucciones del programa principal. En el momento en que se desee agregar un proceso, el editor de diagrama agregará una nueva solapa con similar estructura a la página del programa.

Cada página está dividida en tres partes: una para la definición de los parámetros formales, otra para la declaración de las variables locales y la última para el cuerpo.

La confección del diagrama se realiza por medio de la inserción de los distintos elementos de un programa, seleccionables desde la barra de botones del editor.

Toda modificación efectuada en el diagrama es automáticamente traducida a código textual. Esa traducción se ve reflejada instantáneamente en la posición correspondiente del editor de código (Fig. 3.2)

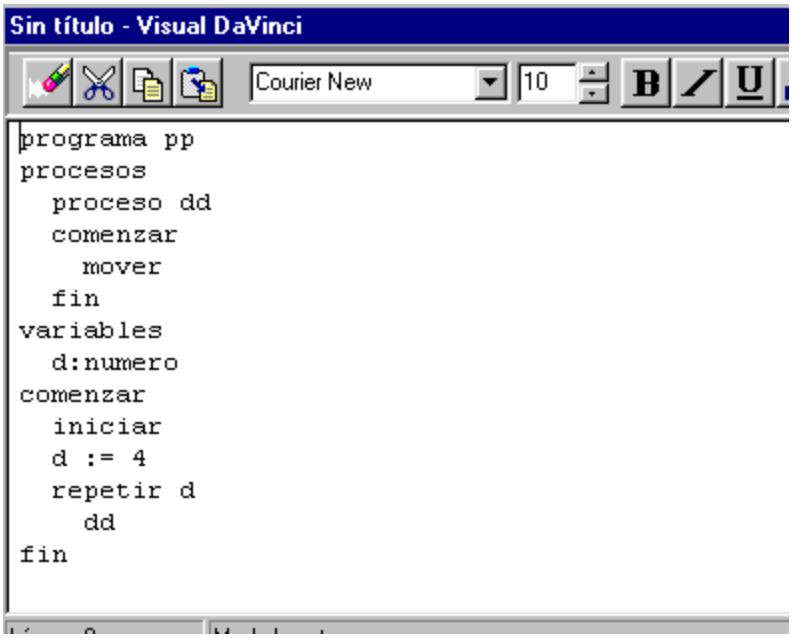


Fig. 3.2 Editor de Código de Visual Da Vinci

El editor de código es simple cuya única asistencia al usuario está relacionada con la indentación semiautomática de las líneas de código y el resaltado de la línea en ejecución durante el proceso de depuración.

El comportamiento del robot durante la ejecución de un programa, se puede ver en la ventana de la ciudad (Fig. 3.3).

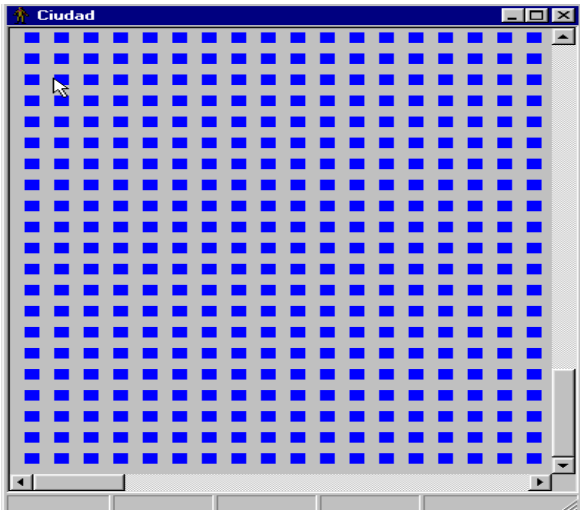


Fig. 3.3 Ciudad donde el Robot ejecuta acciones

Por último se observa la ventana de inspección de variables del sistema (Fig 3.4), la cual permite visualizar en ejecución la posición actual del Robot, la dirección en la cual se esta moviendo, y la cantidad de flores y papeles que tiene en sus bolsas.

Fig. 3.4 Inspector de variables del sistema

3.2.1 – Aspectos de implementación

Visual Da Vinci es específicamente un ambiente de desarrollo. En él un programa puede ser diseñado en forma visual a partir del editor de diagrama, en forma textual, desde el editor de código o en una combinación de ambas. En cualquier caso, primero se verifica la sintaxis, la cual genera código intermedio, para que finalmente se interprete para su ejecución.

La puntos mas sobresalientes de la implementación del lenguaje son los relacionados con la edición visual de diagramas, la traducción automática de diagramas a código, la verificación de código, la ejecución de programas y la visualización de sus efectos.

La verificación de sintaxis, así como la ejecución, se realiza directamente en base al

código textual. Por lo tanto, la eficiencia de los programas desarrollados no está en función de la condición visual del lenguaje, evitando así este problema inherente a los lenguajes visuales. De allí proviene gran parte de la importancia de la derivación automática de diagramas a código textual. Cada vez que se produce un cambio en el diagrama, ya sea una inserción, una eliminación o una modificación, se invoca al método Codificar de los objetos involucrados, el cual se encarga de reflejar esos acontecimientos en el texto del editor de código. Si bien ambas jerarquías, la de los objetos de verificación y ejecución, y la de los objetos de visualización y traducción, podrían haber sido implementadas como una sola, es preferible respetar la separación de los objetos visibles y los no visibles como lo hacen la mayoría de los lenguajes orientados a objetos. [Cha97].

3.2.2 – Ventajas del uso

Las herramientas visuales que emplean el uso de robots resulta muy atractivo y beneficioso para los alumnos pues los problemas pueden representarse utilizando imágenes que muestran las diferentes acciones para resolverlo junto con algunas líneas de texto. Esto lleva, en general que los alumnos tengan pocas dudas respecto de la especificación del problema. La animación – visualización proporciona una retroalimentación visual que permite medir la corrección de los algoritmos planteados. Los alumnos pueden darse cuenta donde el programa ha producido un error, observando simplemente el resultado de la ejecución.

Algunas ventajas observadas con el uso de la herramienta son [DeG03]:

- El alumno siente una gran motivación por resolver problemas concretos que se relacionan con la aplicación directa de los conceptos teóricos.
- Es muy natural pasar de un ambiente limitado (por la clase de problemas a tratar y las estructuras de datos que se manejan) a un lenguaje de programación real tipo Pascal.
- Resulta natural trabajar los conceptos de modularización y derivados de ellos los de datos locales y globales, así como argumentos y parámetros.
- El alumno quiere ir rápidamente a herramientas profesionales más complejas lo que dificultaría su maduración a partir de los conceptos

básicos. En este sentido VDV funciona como un puente que establece limites

3.3 – Análisis Funcional de la Herramienta

Surge la necesidad de encarar un trabajo de reingeniería que permite rediseñar la presentación de la herramienta e incluir funcionalidad.

En general los alumnos opinan que la herramienta los ayuda a entender el concepto de programa y la forma en que trabaja la computadora, rescatando la necesidad y la importancia de la precisión al plantear las instrucciones. Algunos comentarios negativos están dirigidos, básicamente, a las escasas facilidades que provee la herramienta para detectar errores.

Se pueden citar ciertos inconvenientes encontrados durante el uso de la herramienta [DeG03]:

- El alumno siente una atracción por resolver los problemas por "prueba y error" poniendo la verificación experimental antes que el análisis cuidadoso del que y el como.
- La clase de problemas a tratar no exige demasiada abstracción.
- El ambiente tiene defectos propios de un desarrollo de software (típicamente los errores son poco explicados).
- Sería deseable tener una herramienta de menor nivel para la educación pre-universitaria (en la Escuela Media) que vaya evolucionando. En este sentido VDV no tiene grados de flexibilidad.

El informe que a continuación se detalla describe los inconvenientes que se encontraron en Visual Da Vinci como resultado de un análisis exhaustivo de la interfaz utilizada y de la funcionalidad de la aplicación.

El criterio utilizado para dicho análisis se basa principalmente en facilitar al usuario el aprendizaje y el manejo del sistema de manera de que al utilizarlo no se encuentre con problemas que distraigan su atención.

3.3.1 – Análisis de la Interfaz

1. Al iniciarse la aplicación se muestra la imagen de presentación del VDV durante un lapso de tiempo considerablemente largo.

   Sería conveniente reducir el tiempo de exposición de dicha imagen.

2. Cuando el VDV está maximizado se conservan visibles, como fondo de pantalla, las aplicaciones anteriormente abiertas.

   Una alternativa para solucionar esto sería contar con un fondo de pantalla que cubra lo que se encuentra por debajo.

3. El menú principal así como la barra de herramientas pueden quedar ocultas debajo de las ventanas posteriormente activadas, dificultando su acceso inmediato.

   Una solución sería mantenerlos visibles permanentemente en la parte superior de la pantalla.

4. Luego de modificar un programa que ha sido compilado y ejecutado exitosamente, el botón de ejecutar permanece activo a pesar de que el programa no ha sido compilado nuevamente, por lo tanto cuando se presiona dicho botón se ejecuta el código de la compilación anterior.

5. Los datos numéricos pueden ser ingresados con la cantidad que se quiera de ceros no significativos a izquierda, y luego se siguen mostrando de la misma forma en que se ingresaron. Por ejemplo, el número 10 se podría ingresar como: 10 o 010 o 0010.

6. Para mantener la uniformidad, se debería indentar automáticamente todos los números ingresados ya sea a derecha o izquierda y completarlos o no con ceros no significativos.

7. El icono "?" en la barra de herramientas no cumple ninguna función.

8. Dentro la opción del menú "Opciones/Ciudad" aparecen las solapas para elegir Flores, Papeles u Obstáculos en el borde inferior de la ventana a diferencia de la

mayoría de las aplicaciones conocidas en el mercado.

9. Al depositar una flor y un papel en una misma esquina se visualiza en la ventana de la cuidad solo la imagen del papel en dicha esquina quedando oculta la flor

   Por otro lado, si deposito primero un papel y luego una flor aparece la flor con un color verde fosforescente de fondo.

   Una sugerencia sería utilizar otra imagen compuesta por un papel y una flor de manera de que con solo mirar la ciudad se pueda saber lo que deposita el robot en cada esquina.

10. Si la ejecución de un programa hace que el robot deposite una flor o un papel en una esquina y luego se detenga en ella al finalizar el programa, se refleja en la cuidad solo un circulo rojo que indica la posición del robot quedando oculta la flor o el papel.

11. El color de las flores es igual al color del círculo que indica la posición final del robot.

    Para acentuar la diferencia ente ambos podrían tener distintos colores.

12. Al seleccionar la opción del menú principal "Archivo/Presentación-Preliminar", aparece una ventana para configurar la impresora, y si se elige aceptar la configuración, no se visualiza ningún efecto.

13. La opción del menú principal "Archivo/Nuevo", blanquea el Editor de Código, pero el Editor de Diagramas y la Ciudad mantienen el contenido anterior.

14. El icono para salir de la aplicación en la barra de herramientas no refleja visualmente la acción que realiza.

    Por ejemplo podría reemplazarse por una puerta entreabierta.

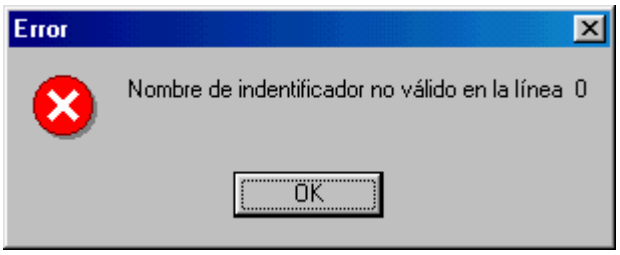
15. Al intentar salir de la aplicación sin haber guardado el programa, aparece una ventana con la leyenda: "Salvar cambios de Sin Título? ", si se contesta afirmativamente, se cierra la aplicación sin permitir salvar e ignorando dicha pregunta.

Nota: la pregunta se podría reescribir de la siguiente manera "Desea guardar los cambios realizados a (nombre del programa)?".

16. La aplicación contiene palabras en ingles tanto en el título de las ventanas como en los botones de aceptación ("Confirm", "Yes", "Cancel").

    Por ejemplo, la ventana que aparece como resultado de la ejecución de la sentencia "Informar", tiene como título la palabra en inglés "Information".

17. Cuando se intenta compilar un programa sin haberle asignado un nombre aparece el siguiente mensaje:



    Este no aporta ninguna información acerca de cual fue el error. Podría decir por ejemplo "El programa principal debe tener asignado un nombre".

18. Sería deseable que se pudiera iniciar la compilación por medio de una combinación de teclas rápidas o de una tecla función.

19. En la esquina superior de las ventanas no aparecen las opciones de cerrar/maximizar/minimizar/restaurar como en las aplicaciones con las que se trabaja habitualmente. De todos modos, con un doble click del mouse sobre la barra del título en la ventana se puede maximizar dichas ventanas.

20. La Ayuda del Visual Da Vinci explica la función de los botones "Entrar en" y "Pasar sobre" de la barra de herramientas, diciendo "... Ejecuta la instrucción actual del programa que se está ejecutando...". Podría cambiarse por una frase un poco más explicativa, por ejemplo: "... Ejecuta paso a paso las instrucciones del programa comenzando en la primer línea y continuando con las siguientes hasta finalizar dicho programa...".

21. Los botones para seleccionar las diferentes ventanas que se encuentran en la barra de herramientas no reflejan sus correspondientes acciones en la opción del menú principal "Ver", ya que no se tildan y destildan las ventanas en dicho menú al presionar los botones en la barra de herramientas. También ocurre la situación inversa, es decir, una modificación en el menú Ver no se refleja en los botones de la barra de herramientas.

22. Las instrucciones son representadas en el Editor de Diagramas con rectángulos que a pesar de que inicialmente se encuentran sin contenido, tienen dimensiones que podrían reducirse.

23. Presionando el icono del diagrama de flujo que permite ingresar el nombre del programa, surge una ventana donde se puede ingresarlo, pero el espacio donde se debe escribir el texto aparece en color azul y el cursor se encuentra posicionado en el último espacio.

24. Normalmente el espacio donde se ingresan los datos, directamente aparece en blanco, con el cursor posicionado en la primera posición habilitada.

25. El tipo de fuente elegido para ingresar el nombre del programa no se corresponde con los normalmente utilizados.

26. El titulo de la ventana de Editor de Código contiene el siguiente texto: "Sin titulo – Visual DaVinci", debería aparecer, al igual que en las demás, el nombre de la ventana a la que pertenece ("Editor de Código") y si se desea también el nombre del archivo con el que se encuentra guardado en disco (en el caso de que ya tenga asignado uno).

27. La ventana que informa el estado actual del robot y la cantidad de flores y papeles, se referencia como "Coordenadas" en la opción Ver del menú principal y como "Robot" en el título de la ventana.

28. El título de la ventana Robot debería ser más significativo, por ejemplo, podría llamarse "Resultado de la Ejecución del programa" o "Estado de la ejecución".

29. Si se modifica en el menú principal la opción "Opciones/Retardo" el número de milisegundos, pero se cancela la operación, y se vuelve a entrar en dicha ventana aparece igualmente el número que se ingresó en vez de aparecer

nuevamente inicializado en cero.

30. Cuando se selecciona la opción del menú principal "Archivo/Guardar-Como" y aún no se salvó el código generado por el diagrama, y se cancela la operación igualmente se elimina el contenido del Editor de Código.

3.3.2 – Análisis del Editor de Código:

1. Después de algún mensaje de error, el número de línea que se indica en el borde inferior del editor de código, no representa la posición donde se encuentra el cursor.

   Por ejemplo, luego del error "Run Time Error: No puede depositar una flor debido a que no hay ninguna en la bolsa de flores", el número de línea actual que aparece debajo de la ventana pasa de ser 8, que es donde realmente se encontraba el cursor, a ser 159.

2. Si se carga un programa desde el disco rígido (*.vdv), éste aparece en el Editor de Código, pero, si por alguna razón (equivocada o no) se realiza una acción en el editor de Diagramas, se borra automáticamente el contenido del editor de Código, a pesar de que se hubiese modificado. Sería conveniente preguntar si se desean guardar los cambios.

3. Solo está permitido tabular una instrucción a derecha hasta dos veces. Con la primera tabulación, el cursor queda ubicado al final de la instrucción y, aunque parezca extraño, igualmente con una segunda tabulación, la instrucción "anterior" al cursor se corre una tabulación. Por ejemplo ocurre esto con la instrucción "mientras".

4. Al intentar tabular un párrafo seleccionado, solo tabula la primer línea de dicho párrafo.

5. Si se indentó con tabulador, y se desea borrar a izquierda, se borra de a espacios en vez de a "tab" como sería deseable.

6. Las sentencias se escriben de forma diferentes, por ejemplo, depositarFlor y hayFlorEsquina se podrían escribir, depositarFlor y hayFlor o bien depositarFlorEsquina y hayFlorEsquina.

3.3.3 – Análisis del Editor de Diagrama

1. El pequeño espacio entre las instrucciones impide mover o insertar una instrucción entre otras dos. El puntero del mouse cambia de prohibido a permitido indicando si se puede mover o insertar una instrucción, solo si se acierta en la angosta línea divisoria.

2. Ya que un programa no compila si no se le coloca un nombre, sería conveniente que en la ayuda, se acentuara que un primer paso en la construcción de un diagrama es asignar nombre al módulo.

3. Solo se permite borrar de a una instrucción a la vez. Lo ideal sería poder borrar también mas de una al mismo tiempo, ya sea seleccionándolas y presionado el botón borrar o la inversa, conservar el botón "borrar" seleccionado e ir marcando las que se desean borrar.

4. Lo que sucede con el botón de borrar también ocurre con los demás botones, si se quiere poner más de un mover seguido, se debe hacerlo seleccionando el botón mover por cada vez que se desea insertarlo.

5. Las instrucciones que se encuentran dentro de la instrucción "REPETIR" y que eventualmente se mueven a otro lugar, desaparecen si se borra el cuerpo del repetir.

6. Una vez colocada una variable (constante o parámetro) en el cuerpo de una asignación, no se permite moverla ni insertar entre dos de ellas un operador.

7. Se permite eliminar la definición de una variable cuando aun es utilizada en el cuerpo del programa.

8. Cuando se suprime un parámetro en el cuerpo de un proceso, el lugar asignado en la invocación no desaparece ni se permite tampoco eliminarlo con el icono "borrar". Sería deseable que se refleje dicha acción de manera de que ocurra lo mismo que con el paso inverso, es decir, cuando se inserta un parámetro en el cuerpo de un proceso, también se incorpora en su invocación.

9. Si se desea borrar la invocación a un proceso, se muestra la ventana para

confirmar dentro del proceso, salvo que, después de seleccionar el icono borrar, se clickee en la fina línea blanca que se encuentra en la base del rectángulo que representa el llamado. En este último caso se lo borra directamente sin pedir confirmación.

10. No se permite eliminar el programa principal, por ejemplo si se quisiera comenzar uno de nuevo desde el principio.

11. Es posible permite llamar al programa principal con el mismo nombre de un proceso que existió y fue borrado pero cuya invocación aún no ha sido borrada.

12. Al eliminarse un proceso no se refleja la acción en el Editor de Código.

13. Se presenta dificultoso insertar una instrucción entre otras dos o al principio, ya que el mouse no cambia de estado indicando que se permite realizar dicha acción.

14. Se permiten sentencias de iteración que no poseen cuerpo.

15. La declaración de un proceso sin invocación no se refleja en el Editor de Código.

16. El texto para el icono "Nombre de módulo" debería ser "Nombre del programa" ya que la palabra "módulo" no es la que se utiliza la mayoría de las veces para referenciar justamente al programa.

17. Cuando se inserta una instrucción "Informar" y por ejemplo se selecciona agregar un valor numérico pero se cancela la acción, igualmente aparece el espacio que se le asignaría a dicha variable en el rectángulo correspondiente a "Informar".

3.3.4 – Análisis de los mensajes de error

Ciertos mensajes de error están escritos en inglés o con una mezcla de inglés y castellano sin sugerir acciones a tomar o posibles soluciones.

A continuación se listan algunos ejemplos:

a) "Access violation at adress 0105995D. Read of adress FFFFFFFF. "

   Cuando en el Editor de Código se mueve una instrucción (sombreándola con el mouse), a otro lugar.

b) "Instrucción esperada en línea 4".

   Cuando la instrucción Iniciar no se encuentra seguida a la instrucción comenzar.

c) "Run Time Error: No puede depositar una flor debido a que no hay ninguna en la bolsa de flores"

   Debería informar donde indicar cantidad de flores.

d) "´010´ is not valid integer value"

   Aparece cuando se coloca uno o más datos numéricos erróneos. No se informa dónde se ingresó mal y cuál fue el error. Y agrega un cero a izquierda que no se ingreso.

e) "´010´ is not valid integer value" o

f) "Invalid input value. Use escape key to abandon changes"

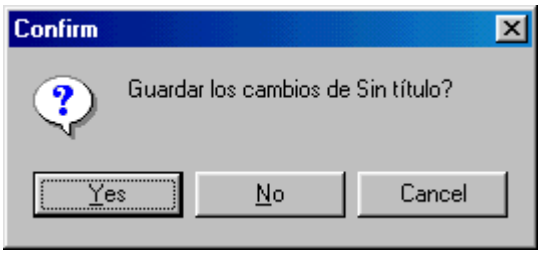
   Uno de estos errores (o uno u otro y al azar) aparece cuando en el menú Opciones/Retardo no se completa con ceros a izquierda los cinco dígitos.

g) "Access violation at adress 0101554E. Read of adress 20207120"

   Sucede en el Editor de Diagrama cuando primero se borra el cuerpo de un proceso, luego sin borrar la invocación de dicho proceso se intenta nombrar al programa principal.

   Nota: a pesar de haber dado el error, si se le da aceptar igualmente le coloca el nombre del proceso borrado y aún existiendo la invocación al proceso.

h) Si se intenta salir de la aplicación sin haber guardado los cambios aparece la siguiente ventana:



Donde el título y los botones de elección aparecen en inglés (“Confirm”, ”Yes”, ”Cancel”) y la pregunta está mal redactada, debería ser algo como “Desea guardar los cambios realizados a ...nombre del programa...?”.

i) “List index out of bounds”.

Ocurre cuando selecciono un nombre de proceso que se eliminó y que aparece en la lista de nombres permitiendo elegirse.

3.3.5 – Sugerencias

1. Cuando se selecciona un icono de alguna instrucción que se desea insertar en el cuerpo de un programa o proceso, sería deseable que la flecha del mouse su estado de manera de indicar en que lugar se puede colocar dicha instrucción.

2. La instrucción primitiva “Iniciar” posiciona al robot en la esquina (1, 1) orientado hacia el norte, e inicializa las variables del sistema. Ya que se debe colocar obligatoriamente un iniciar después del comenzar principal sería deseable que se coloque automáticamente como el comenzar-fin o que se incluya su código dentro del comenzar.

3. Sería deseable que se coloquen automáticamente, cuando se abre la ventana de código, las palabras claves programa/comenzar/fin.

4. Cuando se informa un error me dice el número de línea donde ocurrió. Convendría proveer una opción go-to para fácilmente ubicar la línea o simplemente posesionar el cursor en la línea que se generó el error.

5. En el nombre de variables por ejemplo podría estar permitido combinar letras y

números siempre que empiecen con una letra.

6. Cuando se intenta insertar una instrucción, se presiona un botón y luego se indica donde se debe insertar. Para colocar una variable dentro de una asignación por ejemplo se debe arrastrar dicha variable dentro de la asignación. Deberían seguir el mismo criterio en ambos casos permitiendo o bien arrastrar con el mouse o bien seleccionar e indicar donde insertar.

7. La cantidad de flores, papeles y obstáculos se especifica dentro de una opción del menú principal. Dentro del programa solo se puede usar esa cantidad. Se debería poder inicializar la cantidad dentro del programa ya que no es algo que deba estar separado y en el menú principal pues tiene relación directa con el programa en cuestión.

8. Sería ventajoso contar con algún símbolo que permita comentar solo una línea sin tener la necesidad de encerrarla entre llaves. Por ejemplo poniendo al principio de la línea doble guión "--".

9. Para el Editor de Diagrama, podría elegirse otra combinación de colores no solo por lo llamativo de algunos de ellos sino para distinguir mejor hasta donde abarcan las estructuras de control iterativas (repetir, mientras y si/sino).

10. Se podría guardar la ubicación de las ventanas que el usuario crea más conveniente de manera de que cuando ingrese nuevamente pueda elegirla sin necesidad de tener que volver a ubicarlas de nuevo. Otra alternativa sería arrancar con una ubicación standard donde se vean todas las ventanas y que el usuario luego la pueda cambiar si así lo desea.

11. Para que exista una línea similar a la de otros lenguajes que se verán en la facultad, sería deseable que en el Editor de Código las sentencias sigan el mismo patrón que por ejemplo en Objetos, que empiezan con minúscula y si es una palabra compuesta por más de una, que las internas comiencen con mayúscula. Ejemplo: depositarFlor, hayFlorEsquina, etc.

12. Agregar el manejo de string permitiría por ejemplo, poder mandar mensajes más significativos y no simplemente un valor de una variable. También se podría emitir un mensaje de terminación exitosa o mensajes intermedios para el seguimiento del programa.

13. Ya que la ventana de Coordenadas solo tiene la función de informar, no tiene sentido que se permita modificar número de avenida, calle, flores y papeles. Esto presta a confusión puesto que aparenta ser el lugar donde se coloca la cantidad de flores y papeles con la que se desea contar para la ejecución de un programa.

14. Ya que los comentarios en el código fuente no afectan la ejecución del programa, se deberían poder insertar en cualquier lugar sin considerarse reglas de indentación ni dar error por colocarlos antes que la instrucción "Iniciar".

15. Para ayudar al programador a saber si está escribiendo correctamente las sentencias, se podría incorporar en el Editor de Código una funcionalidad que cambie el color de las sentencias una vez que ésta ha sido escrita correctamente. El color dependería del subconjunto al que pertenece la sentencia: primitivas, sentencias simples y sentencias compuestas.

16. Mediante la incorporación de un Zoom, se podría lograr una herramienta que brinde "foco + contexto", es decir que interactúe con el usuario mostrando simultáneamente en pantalla un número razonable de elementos y manteniendo en todo momento el contexto de lo que se está mostrando.

CAPÍTULO IV

PROPUESTAS E IMPLEMENTACIONES

Es sabido que los programas son usados por usuarios con distintos niveles de conocimientos, desde principiantes hasta expertos. Es por ello que no existe una interfaz válida para todos los interesados y todas las tareas. Debe permitirse libertad al usuario para que elija el modo de interacción que más se adecue a sus objetivos en cada momento y además le permita incorporar conocimientos a su propio ritmo. Cada uno tiene un modelo mental propio de la interfaz, que contiene los conceptos y expectativas acerca de la misma, desarrollados a través de su experiencia.

4.1 – Propuestas

Los distintos tipos de usuarios, alumnos y docentes, y los diferentes grados de alfabetización informática permiten estructurar la presente propuesta en distintos niveles, un primer nivel dirigido a los alumnos y docentes de nivel preuniversitario, un segundo nivel para los alumnos de un primer curso de programación básica en carreras informáticas y finalmente un tercer nivel para alumnos de cursos avanzados de las carreras informáticas.

El primer nivel introduce los conceptos básicos de algoritmos y datos simples en la escuela media. En este sentido, alguna de las mejoras propuestas son sincronizar el editor de diagrama y el editor de código de forma tal que toda modificación realizada en el código se refleje automáticamente en diagrama, resaltar las palabras claves dentro del editor de código a medida que se está escribiendo con el objetivo de minimizar los errores de sintaxis, proveer de indentación semi – automática a fin de poder agrupar sentencias compuestas, elaborar mensajes de error en compilación mas claros y fácilmente identificable, generar pasos bien definidos para la verificación, distribución de objetos y visualización de programas, rediseñar la interfaz de usuario a fin de tener una visión global de todo el lenguaje.

El segundo nivel, similar al actual, para los cursos introductorios de algorítmica y programación de nivel universitario, se propone eliminar el editor de diagrama, incorporándole al de código mayor funcionalidad, entre otras, deshacer mas de una acción, permitir buscar/buscar reemplazar texto dentro de un programa, acceder directamente a una línea específica, proporcionar ayuda para palabras seleccionadas, disponer de combinación de teclas de acceso rápido. Por otro lado, se pretende aumentar la complejidad dotando a la herramienta de barreras y obstáculos para la ciudad, cuadras divididas en mas de un paso con la primitiva mover representando la ejecución de solo uno de ellos, ingreso de valores por programa (similar al Read de Pascal), múltiples editores de códigos, mapa estructural de programa.

El tercer nivel para iniciar a los alumnos en los conceptos de concurrencia utilizando el ambiente con múltiples robots. Se proyecta tener una cantidad arbitraria de robot, los cuales podrán circular por la ciudad, cooperando entre si para realizar una tarea específica, con distintos métodos de sincronización y comunicación entre ellos, proveer mecanismos para la ejecución de instrucciones en forma concurrente.

Asociado a esta propuesta también se propone el desarrollo de un entorno en la web para la orientación pre-universitaria no presencial, destinada a los alumnos y docentes de la escuela media. En este soporte se preveen demostraciones animadas utilizando la herramienta, incorporando ayudas on line, solución y visualización de problemas propuestos, etc. que complementan las actividades de aprendizaje y tutoría virtual que ofrece la Facultad de Informática de la UNLP para aquellos alumnos interesados en las carreras que ofrece esta institución.

4.2 – Elección y justificación

Los cursos introductorios a la programación estructurada se centran en la resolución de problemas de complejidad simple y su representación por medio de algoritmos, para ello se utilizan diferentes paradigmas y dentro de cada uno de ellos diferentes herramientas (desde expresiones formales, lenguaje natural o lenguajes de programación reales o simbólicos) [Fag00] [Gai98] [Kae01] [Pat81]

La enseñanza de la programación tiene como objetivo desarrollar la capacidad para la resolución de problemas y la expresión de las soluciones mediante alguna herramienta de expresión simbólica.

Este objetivo requiere que el alumno alcance alguna forma de representación mental del mismo, lo convierta en un modelo computable, analice alternativas de solución y decida la secuencia de ejecución más eficaz para cumplir una función específica del "programa- solución".

Se ha observado y concluido que los alumnos de cursos iniciales en programación en general presentan algunas características que conllevan dificultades en el aprendizaje y concreción de los objetivos definidos: [DeG03]

- falta de entrenamiento en pensar y expresar ideas.
- dificultad en la comprensión y asimilación de contenidos.
- dificultad de aprendizaje de temas básicos que impliquen algún grado de abstracción.
- falta de orientación vocacional adecuada.
- gran disparidad de conocimientos y formación previa.
- escasa valoración por el trabajo sistemático.
- cierta metodología de "prueba y error" asociada con el empleo de computadoras.

Normalmente estos alumnos no cuentan con conocimientos básicos de Informática y además poseen información escasa o parcial sobre sus alcances. Esto hace que la

formación inicial que poseen los alumnos sea muy diversa [Gon02].

Si se puede nivelar tempranamente estos obstáculos por medio de algún mecanismo, las posibilidades de éxito por parte del alumno tienen a aumentar considerablemente.

Por lo tanto, luego de realizar diferentes observaciones y haber leído varios autores que tratan el tema de la deserción en los primeros cursos de introducción a la programación en el nivel universitario, se considera de fundamental importancia que existan mecanismos que posibiliten actividades con los alumnos interesados en la disciplina y que están cursando los últimos años de la escuela media. Por lo tanto se propone entonces la realización de un "Lenguaje de Expresión de Problemas y Algoritmos para el aprendizaje de la Informática en niveles preuniversitarios" (LEPA), que tiene como punto de partida el lenguaje Visual da Vinci, el cual ha sido específicamente diseñado para la enseñanza de la programación estructurada en los cursos introductorios de las carreras de la Facultad de Informática de la UNLP.

Se espera que esta herramienta permita superar los inconvenientes mencionados en los diferentes capítulos analizados, que haga atractivo e interesante el proceso de enseñanza-aprendizaje y que beneficie el autoaprendizaje y la autoevaluación por parte de los estudiantes, ayudándolos a evaluar su interés y competencia para realizar estudios vinculados a la especialidad y aproximándolos, en una etapa muy temprana en sus estudios, a algunos de los problemas de la informática.

La principal justificación es que este acercamiento pueda permitirle al alumno confirmar o desechar su interés en la resolución de problemas y, en caso afirmativo, le permita definir las capacidades que debe desarrollar para hacerlo eficientemente y así estimular el desarrollo de las habilidades para resolver problemas de manera autónoma y precisa, con la intención de disminuir la dificultad para alcanzar los niveles de abstracción necesarios y expresar rigurosamente los algoritmos.

4.3 – Implementaciones

Tanto en el desarrollo de Lepa como en el diseño del entorno Web la principal meta fue la de concluir, mediante entornos amigables y sencillos de comprender, con ambientes que incrementaran la motivación, asistieran en el desarrollo de habilidades

analíticas, sirvieran de soporte a los docentes y ayudaran a los estudiantes a comprender conceptos básicos en la programación estructurada.

4.3.1 – Lepa. Características del ambiente

Inicialmente se comenzó, como se describió en el capítulo III, con un análisis exhaustivo de la herramienta actualmente en uso, Visual Da Vinci. Por lo tanto, el primer paso fue solucionar los inconvenientes encontrados.

Luego se rediseño la interfaz, para obtener una visión totalmente integrada del ambiente de desarrollo, ya que se considera que este punto es fundamental para que no actúe como elemento de dispersión, por lo tanto la distribución final quedó como se muestra en la siguiente figura:

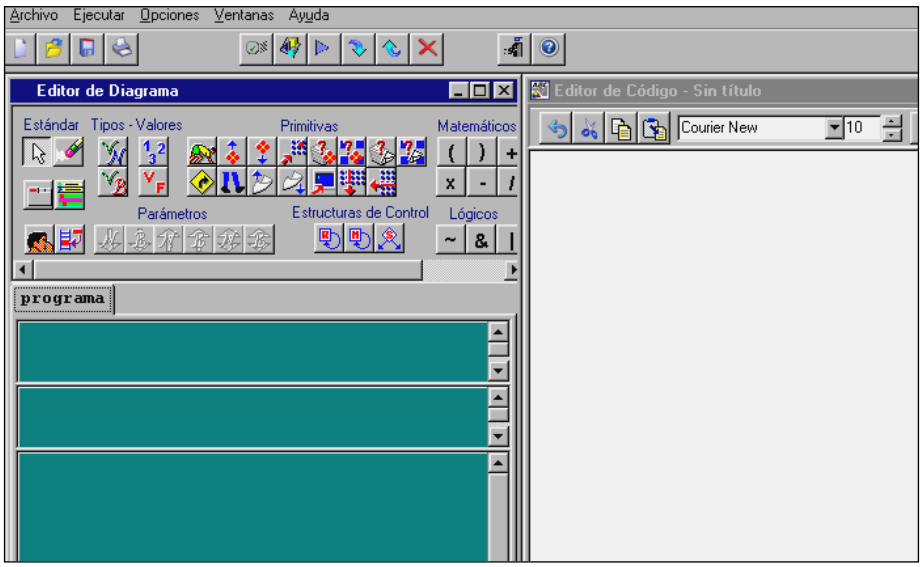


Figura 4.1. Entorno de desarrollo de Lepa

Se puede observar que existe una ventana principal o contenedora de todo el ambiente de desarrollo. Por un lado se encuentra el editor de diagrama, el cual permite, ir armando un programa a través de la selección de diferentes elementos. En el sector derecho se encuentra el editor de código, el cual posibilita el ingreso textual de un

programa.

Al iniciar la aplicación se desplegará una distribución como la que se muestra en la figura anterior. Se observa en la parte superior un menú principal toda la funcionalidad necesaria para trabajar con archivos vdv (extensión de los archivos creados en Da Vinci, los cuales siguen manteniendo la misma extensión en Lepa), compilar y ejecutar programas, tener diferentes vistas, distribuir elementos en la ciudad. Estas mismas opciones se encuentran en la barra de herramientas, situada por debajo del menú. Por otro lado se observa sobre el costado izquierdo el editor de código, el cual permite ir generando un programa a partir de la selección de diferentes elementos. Por último, al costado derecho se presenta el editor de código, en el cual se puede escribir el programa deseado.

Cabe destacar que a partir de un programa escrito en el editor de código y el cual compila esto es, esta correctamente escrito) se obtiene el diagrama correspondiente, y que a partir de la inserción de elementos en el editor de diagrama se va generando automáticamente el código textual.

A continuación se describen las funcionalidades proporcionadas en el menú principal:

Menú archivo:

En esta opción se observan las funcionalidades para trabajar con archivos, crear uno nuevo, abrir uno existente, guardar el archivo actual, guardar como (guarda el archivo actual posiblemente con otro nombre), imprimir el programa corriente, configurar la impresora, y por último salir de la aplicación.

Se debe notar que ciertas funciones poseen sobre su margen derecho alguna combinación de teclas, esto sirve para realizar con el teclado lo mismo que se hace desde el menú.

Menú Ejecutar:

En esta opción se muestran los pasos a seguir para poder ejecutar un programa.

Primero: mediante la sub-opción Compilar se verifica que el

programa este correctamente escrito, en caso de detectar error mostrará el correspondiente mensaje indicando el número de línea en el editor de código donde se ubica el mismo.

Segundo: mediante la sub-opción Ubicar Objetos, luego de una compilación exitosa, esta función distribuye los objetos a lo largo del perímetro especificado las flores y/o papeles en la opción Ciudad del menú Opciones.

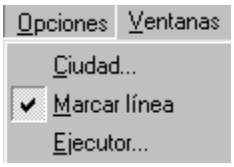
Tercero: desde la función Ejecutar se puede ver la ejecución del programa diseñado. Para ello se muestra la ciudad junto con el recorrido que realiza el robot.

Pasar sobre: ejecuta paso a paso las instrucciones del programa corriente, en caso de tener definido un proceso, si este se invoca desde algún punto del programa, la opción Pasar sobre no entrará en el código del mismo.

Entrar en: ejecuta paso a paso las instrucciones del programa corriente, en caso de tener definido un proceso, si este se invoca desde algún punto del programa, la opción Entrar en, a diferencia de Pasar sobre, entrará en el código del mismo.

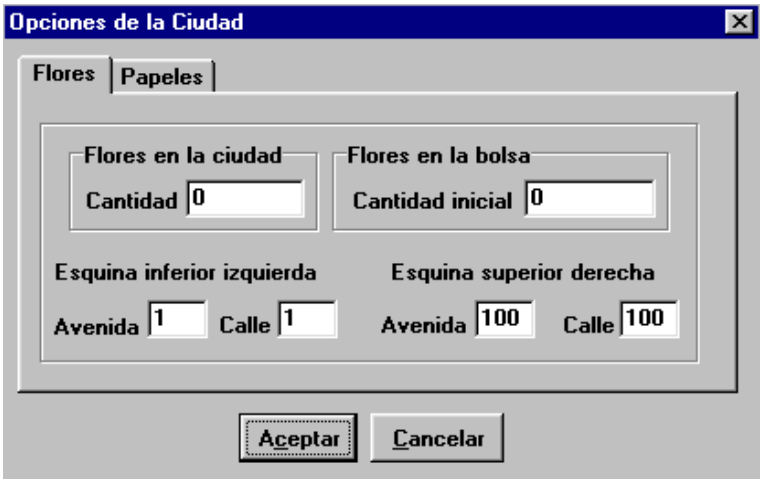
Abortar: finaliza la ejecución del programa corriente.

Menú Opciones:



En esta sección se encuentran opciones de inicialización de datos.

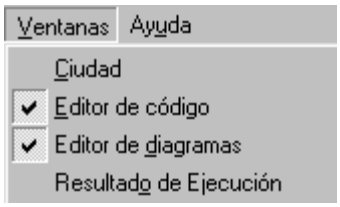
Ciudad: permite distribuir flores y/o papeles en el perímetro especificado, como así también inicializar las bolsas de flores y papeles que el robot lleva consigo en un rectángulo delimitado por sus esquinas inferior izquierda y superior derecha.



Marcar línea: permite indicarle al ejecutor si se desea marcar cada instrucción a medida que el programa se está ejecutando.

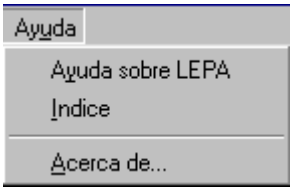
Ejecutor: Permite establecer un retardo en milisegundos entre la ejecución de una instrucción y la siguiente. Esto puede resultar útil cuando se desea seguir un algoritmo sin utilizar la ejecución paso a paso.

Menú Ventanas:



Este menú permite mostrar/ocultar las ventanas disponibles en el ambiente.

Menú Ayuda



A partir de esta opción se podrá visualizar la ayuda interactiva de Lepa. La misma contiene información sobre el lenguaje – sintaxis, semántica, reglas de programación – , el ambiente de desarrollo, y una guía indicativa de cómo se debe comenzar a trabajar.

4.3.2 – Diseño del entorno virtual

Las nuevas tecnologías de la información están revolucionando todos los terrenos en las cuales se aplican, y la enseñanza universitaria no puede ni debe permanecer ajena a los nuevos retos que se le plantean [San00].

Los profesores y alumnos cuentan con tecnologías cada vez más diversas, flexibles y sofisticadas, apareciendo en muchos casos como la base de los futuros sistemas educativos que traerán consigo la creación de nuevos entornos, tipos y facilidades para el aprendizaje.

La mayoría de los sistemas actuales, a través de los últimos desarrollos y avances tecnológicos, incluyen la posibilidad de trabajar con gráficos e incorporar otros medios adicionales (sonidos, imágenes, animaciones, hipertexto, etc.). Los elementos audiovisuales constituyen una herramienta muy poderosa, no solamente para el ocio y el consumo, sino dentro del trabajo y la educación, debido al gran poder didáctico y motivador que permiten incorporar a las aplicaciones que hacen uso de ellos.

Por otra parte, las redes informáticas ofrecen una perspectiva muy diferente a las computadoras aisladas. En principio rompen la separación tradicional de las aulas, abriéndolas al mundo. Permiten la comunicación entre las personas eliminando las barreras del espacio y el tiempo. El mayor potencial de las nuevas tecnologías de la información en la educación reside no solo en lo que aportarán a los métodos de enseñanza-aprendizaje actuales sino también en el modo de trabajo, de relacionarse y de percibir.

Con la digitalización de la información se está rompiendo con su tradicional estructura lineal. Al digitalizar la información ésta se hace discreta y, por lo tanto, compartible, manejable en partes y transportable. En nuevos soportes y espacios de la comunicación (pantallas y no papeles) se está abriendo una nueva era de la comunicación en la que una nueva palabra parece que puede curar los males del papel: interactividad. Ahora el receptor, no delante del papel sino de la pantalla, puede interferir el mensaje y decidir parcial o totalmente lo que va a recibir. La comunicación deja de ser unívoca y el mar de información en el que se esta inserto se puede hacer manejable en función de intereses e inquietudes. No hay obligación de seguir la linealidad narrativa del papel y su jerarquía causal, sino que, en principio, se podría establecer un itinerario propio. El término que simboliza estas posibilidades es el hipertexto.

Con las nuevas tecnologías de la comunicación, sea cual fuere el tipo de información (textual, sonora o imágenes), la estructura y organización ya no es lineal sino hipertextual. El discurso ya no se monta por jerarquías causales, sino por nodos referenciales. Esto permite, con las nuevas posibilidades de la multimedia, desarrollar uno de los factores esenciales de la enseñanza: el aprendizaje.

Como conclusión de la investigación y desarrollo implementado, se muestra el complemento de un entorno virtual, el cual permita encontrar un medio alternativo para obtener información sobre contenidos. El desarrollo de este entorno web tiene como objetivo cumplir la función de orientación pre-universitaria no presencial, destinada a los alumnos y docentes de la escuela media. En este soporte se preveen demostraciones animadas – interactivas utilizando la herramienta, que complementan las actividades de aprendizaje y tutoría.

Para la implementación de las páginas se utilizó HTML, JavaScript, Flash para armar el diseño de la interfaz. Las demostraciones fueron realizadas mediante la aplicación SnagIt 7.

4.3.2.1. Descripción del entorno

Cuando se inicia la aplicación se muestra una página de inicio como se la que se ve en la siguiente figura:

Se puede observar que en el borde izquierdo existe un menú de opciones, las cuales permiten interactuar con otras páginas.

Cada una de estas opciones contiene información sobre el lenguaje, su sintaxis y semántica y cuales son las reglas que hacen al estilo de programación. Además se detalla como comenzar a trabajar con el ambiente, finalizando con las demostraciones de algunos algoritmos.

Se puede observar que existe un acceso rápido a cada una de las opciones situado en el borde superior derecho.

La siguiente página detalla las características principales secciones que componen un programa escrito en Lepa, junto con las reglas que hacen al estilo de programación.

En la página "Cómo comenzar a trabajar en Lepa" se describen los pasos necesarios para implementar un algoritmo en el ambiente, asimismo se detalla la forma de ejecutar programas ya escritos y compilados

Finalmente, en la página "Animaciones y código", se presentan, a modo informativo, algunas animaciones de programas.

BIBLIOGRAFÍA

1. [Inf94] Informe de resultados de la Comisión de Enseñanza (90-95). Fac. Ciencias Exactas, UNLP.

2. [Inf95] Informe de resultados de las Universidades Nacionales. Secretaría de Políticas Universitarias. Ministerio de Educación de la Nación. 1995.

3. [Inf02] Informe de resultados de la Comisión de Ingreso (2002). Fac. de Informática de la UNLP.

4. [Inf03] Informe de resultados de la Comisión de Ingreso (2003). Fac. de Informática de la UNLP.

5. [Mad04] Madoz, C. Gorga G., Feierherd G., Depetris B. "Transformación de un curso presencial preuniversitario de Análisis y Expresión de Problemas a modalidad Semipresencial". Primer Congreso Virtual Latinoamericano de Educación a Distancia. 2004

6. [Rae] Real Academia Española http://www.rae.es

7. [Des91] Diccionario Enciclopédico Santillana (D.E.S., 1991)

8. [Ham81] Lógica para matemáticos. A. G. Hamilton Ed. Parainfo. 1981

9. [Gar00] GARCÍA RAFFI, L.M.; SÁNCHEZ-PÉREZ, J.V.; SÁNCHEZ-PÉREZ, E.A.; "Enseñanza combinada de matemáticas y física en las Escuelas de Ingeniería mediante la fijación de conceptos empleando algoritmos lógicos: aplicación a la resolución de problemas". Barcelona (España): En Actas del Primer Congreso Internacional "Docencia Universitaria e Innovación", 2000.

10. [Ghe87] "Programming Languages Concepts". Ghezzi, M. Jazayeri. John Wiley and Sons. 1987

11. [DeG88] "LUBO-1: Una Máquina Abstracta para un Primer Curso de Programación". A. De Giusti, C. Madoz, P. Pesado. Anales de XIV Conferencia Latinoamericana de Informática. 1988

12. [DeG89] "Abstract Machines in a first Course of Computer Science". A. De Giusti, L. Lanzarini, C. Madoz. Proceedings 11th International Symposium "Computer at the University". Zagreb. Yugoeslavia. 1989

13. [DeG03] De Giusti A., Madoz C., Gorga G., Feierherd G.,Depetris B. "Enfoques y herramientas en la enseñanza de un primer curso de computación". Cacic 2003

14. [Cha97] Champredonde, R. Visual Tool from programming teaching. ICIE . Bs. As. 1997. Tesina de grado. Facultad de Informática. UNLP.1997.

15. [You93] "Análisis Estructurado Moderno". Yourdon. Prentice Hall. 1993

16. [Nas73] "Flowchart Techniques for Structured Programming". I.Nassi, B.Shneiderman. ACM SIGPLAN Notices. Volumen 8. Número 8. 1973

17. [San] Sanz Valero, P J., Coltell Simón, O. y Marín Prados, R. Autoaprendizaje basado en multimedia en el ámbito de las enseñanzas técnicas universitarias: un caso práctico. Campus de Penyeta Roja. Departamento de Informática. Universitat Jaume I. 12071–Castellón. http://www.quadernsdigitals.net/articuloquaderns.asp?IdArticle=1636.

18. [Fag00] Fagin, B. Using Ada-Based Robotics to Teach Computer Science. Conference Proceddings The 5 th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education. ITiCSE 2000.

19. [Gai98] Gail, A. What Abilities Are Necessary for Success in Computer Science? ACM SIGCSE BULLETIN, Volume 30, Number 4. December 1998.

20. [Kae01] Karen R. "Teaching Programming from a Distance : Problems and a Proposed Solution." ACM SIGCSE BULLETIN, Volume 33, Number 4. December 2001.

21. [Pat81] Pattis, R. Karel the Robot : A Gentle Introduction to the Art of Programming. Wiley & Sons. 1981.

22. [Gon02] Gonzalez A. Hacia un perfil del ingresante de Informática. La articulación entre sus expectativas y la realidad. Seminario aspectos sociopolíticos universitarios. Carrera docente. UNLP. 2002.

23. [Tech] SnagIt 7.0 versión trial. http://www.techsmith.com

24. [Aut03] "Autómatas y Lenguajes", Ramón Brena , Tec de Monterrey, 2003

25. [Sin] "Sintaxis y Semántica del Lenguaje"

26. [Pro00] "Procesamiento Computacional del Lenguaje Natural", Cutberto Uriel Paredes Hernández,2000

27. [Mor04] "Aplicación de mapas conceptuales hipermediales en la visualización de programas", Norma Moroni - Perla Señas. Concept Maps: Theory, Methodology, Technology Proc. of the First Int. Conference on Concept Mapping. A. J. Cañas, J. D. Novak, F. M. González, Eds. Pamplona, Spain 2004

28. [Sve02] "SVED: Sistema de visualización de algoritmos", Norma Moroni – Perla Señas. Cacic 2002

29. [Gom04] "Diseño de Interfaces de Usuario Principios, Prototipos y Heurísticas para Evaluación", Leopoldo Sebastián M. Gómez (2004)

30. [Señ02] "Herramientas y estrategias visuales para sistemas de aprendizaje", Sergio Martig y Perla Señas. WICC 2002

31. [Gri99] Grinstein G – Levkowits H. "Perceptual Issues in Visualization", Springer-Verlag, 1999

32. [Sta98] Stasko, Domingue, Brown, Price. (1998). Software Visualization: Programming as a Multimedia Experience. MIT Press.

33. [Rae85] Raeder G. A. Survey of Current Graphical Programming Tecniques. IEEE Computer. Volumen 18. Nro 8. 1985

34. [Car99] Card, S. K., Mackinlay. J.D. y Shneiderman, B. Reading in Information Visualization: Using Vision to Think, Morgan-Kaufmann Publishers, San Meteo, CA, 1999, 16

35. [Nil]

36. [Teo99]

37. [Cap97]

38. [Gol90]

39. [Mye90]

40. [PRO 1]

41. [SIS 1]

42. [PER99]

43. [ONT97]

44. [NOV88]

45. [KID 1] si es una pagina solo tendria que quedar Kid

46. [MAP 1]

47.