Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

LIDA/REC para el análisis estadístico y visualización de datos

Tesis que presenta

Armando Palafox Martínez

para obtener el Grado de

Maestro en Ciencias

en Computación

Director de la Tesis

Dr. Sergio Víctor Chapa Vergara

México, D. F. **Noviembre, 2013**

Resumen

La programación visual es la parte de la computación la cual consiste la utilización de representaciones gráficas en el proceso de programación, estas representaciones deben de ser significativas y no meramente decorativas. La programación visual es estimulada por las habilidades no verbales que poseemos. En el proceso de la programación hay muchos aspectos, como son los lenguajes y los entornos para trabajar con estos, la visualización de los datos y de los resultados de la ejecución de algún programa, el diseño de la aplicación; la programación visual se puede aplicar a todos estos aspectos.

Los lenguajes visuales son un paradigma para expresar sistemas de cómputo. Ofrecen la posibilidad de una manipulación directa de objetos computacionales, en un caso particular para resolver el problema de visualización de datos. Un programa visual escrito en un lenguaje visual dado, consiste en un arreglo espacial de íconos. Un ícono generalizado es un objeto con una representación dual de una parte lógica (el significado) y una parte física (la imagen). Cuando los íconos son metáforas adecuadas para objetos computacionales, el significado de la sentencia visual es como la que el humano supone y depende de cuánto las construcciones mentales corresponden a las construcciones del lenguaje visual.

El objetivo de esta tesis es crear una versión de LIDA/REC para el análisis estadístico y visualización de datos a través funciones estadísticas, estas funciones son representadas por íconos que podrán tener entradas y salidas con lo que se conectarán con otros íconos de diferentes procesos. El flujograma completo es modelado con LIDA (Lenguaje Iconográfico para el Desarrollo de Aplicaciones), cuyo ambiente es visual, y este generará un lenguaje intermedio llamado REC (Regular Expression Compiler) en una versión estadístico, y los resultados que se arrojen van a poder ser descritos mediante gráficas, como son los histogramas, de pastel, de barras, entre otras.

Se desarrolló el compilador de REC para realizar operaciones del lenguaje estadístico R con lo que se generan las gráficas mencionadas anteriormente y los valores estadísticos según la selección del usuario, así como también la interfaz entre el sistema R y la herramienta visual. Todo este proceso será transparente para el usuario lo cual le facilitará la realización de sus tareas de análisis.

Abstract

Visual programming is part of the computer which is the use of graphic representations in the programming process, these representations are to be meaningful and not merely decorative. Visual programming is stimulated by nonverbal skills we possess. In the process of programming many aspects, such as languages and environments for working with these, display data and the results of the execution of a program, the application design; visual programming can be applied all these aspects.

The visual languages are a paradigm for expressing computational systems. They offer the possibility of direct manipulation of computational objects in a particular case to solve the problem of data visualization. A visual program written in a given visual language, is a spatial arrangement of icons. A generalized icon is an object with a dual representation of a logical part (the meaning) and a physical part (the image). When the icons are metaphors suitable for computational objects, the meaning of the visual sentence is like the human involved and depends on how the mental constructs correspond to visual language constructs.

The aim of this thesis is to implement a version of LIDA/REC for statistical analysis and data visualization through statistical functions, these functions are represented by icons that may have inputs and outputs that will connect with other icons of different processes. The complete flowchart is modeled with LIDA (Iconographic Language Application Development), whose environment is visual, and it will generate an intermediate language called REC (Regular Expression Compiler) in a statistical version, and the results will be thrown to be described using graphs such as histograms, pie, bar, among others.

REC compiler was developed to perform statistical language R thus generated above graphs and statistical values according to user's selection, as well as the interface between the R and the visual tool. This whole process is transparent to the user which will facilitate the realization of their analysis tasks.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo financiero que me otorgó, sin el cual no hubiera sido posible cursar la maestría.

Agradezco a mi asesor, el Dr. Sergio Víctor Chapa Vergara, por su trato tan agradable, por todos los conocimientos que compartió conmigo y por su apoyo durante el desarrollo de mi tesis. También a los revisores de mi tesis, el Dr. Amilcar Meneses Viveros y a la Dra. Xiaoou Li Zhang, por el tiempo dedicado para enriquecer mi tesis.

A todos mis compañeros de generación por apoyarme cuando alguna materia se me complicaba, también por los momentos divertidos que pasamos.

A todos los profesores del CINVESTAV-IPN por transmitirme sus conocimientos, a Sofia por su amabilidad, atenciones y por echarme la mano en incontables ocasiones con los trámites que tuve que hacer.

Agradezco también a mis padres, por estar siempre conmigo, por su paciencia y apoyo.

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde hace mucho tiempo se han usado elementos gráficos como medio de comunicación [1], empezando a introducirse en la tecnología de la computación gráfica como una de las más prometedoras en el campo. Sin embargo, con claridad, se puede decir que: no es sino en los 90's cuando los lenguajes y ambientes visuales toman su lugar y derecho en la computación, inician su presencia con la aparición de la revista *Journal of Visual Languajes and Computing* [2].

La computación desde sus principios ha hecho uso de diagramas y herramientas gráficas para el diseño y la programación. Los diagramas de flujo asociados a la programación y los diagramas de Nassi-Schneiderman llamadas cartas de estructuras. A mediados de los ochenta y principios de los noventa, es cuando de una manera más definitiva surgen los lenguajes gráficos, para incursionar en diferentes temas de la computación: ingeniería de software, programación, procesamiento de datos, y en bases de datos, uno de los más significativos Query-by-Example y los diagramas Entidad-Vínculo de Chen. En ese momento los sistemas y lenguajes visuales respondían ya a la tecnología, pantallas y monitores de computadora, el avance en la tecnología referente al campo de la computación ha permitido que los usuarios tengan contacto con dispositivos de gran poder gráfico con el cual pueden interaccionar de una forma más fácil, ya que no sólo se utiliza un teclado como método de entrada sino que también se pueden hacer uso pantallas táctiles u otros dispositivos. También, las características de simplicidad y autocontención en sus elementos, representaban un nuevo medio para los diseñadores, programadores y usuarios de la computación para describir procesos y definir datos. De esta manera, se creó el área denominada *Lenguajes Visuales* o *Sistemas Basados en Íconos*, la cual retoma formas de comunicación antiguas basadas en representaciones gráficas para aplicarlas a diversas áreas de la computación, además de que se basa en el hecho de que podemos asociar imágenes a conceptos de una forma más rápida para sacar provecho de las habilidades no verbales que poseemos.

1.1. Programación Automática y Lenguajes Visuales

Automatizar el proceso de desarrollo de software significa que la computadora participe en la tarea de elaboración del mismo, transfiriendo a la máquina el proceso de transformación de alguna o de todas las etapas del proceso. El desarrollo de sistemas de muy alto nivel tiene como objetivo proporcionar lenguajes constructores de abstracción procedimental y manejo de datos, con la finalidad de que el usuario escriba enunciados del programa en un sentido más descriptivo. Los sistemas de programación visual basados en esquemas y diagramas han venido a ser uno de los paradigmas en donde la interacción hombre máquina se basa en una comunicación con: íconos, diagramas, gráficas e imágenes [3] y [4]; el objetivo es facilitar la especificación en programación automática como un problema de lenguajes visuales [5].

1.1.1. Una perspectiva de los lenguajes visuales

Un lenguaje visual es una representación pictórica de entidades conceptuales y operaciones y es, esencialmente una herramienta mediante la cual los usuarios pueden construir íconos u oraciones visuales. Los íconos se refieren generalmente a la imagen física de un objeto. Los compiladores para los lenguajes visuales deben interpretar expresiones visuales y trasladarlas a un formato que permita la ejecución de la tarea o proceso descrito en el arreglo visual. El compilador no puede determinar el significado de la expresión simplemente conociendo los íconos involucrados en la expresión, sino que debe analizar el contexto de integración de la expresión. La disposición espacial, determinada por las reglas de construcción de las operaciones del lenguaje, permite determinar la semántica completa de la construcción. Lograr que la intensión del usuario coincida con la interpretación de la computadora es una de las tareas más importantes de un lenguaje visual.

1.1.2. Los íconos en la actualidad

En la vida, existen infinidad de representaciones gráficas que tienen un significado exacto asociado. Las personas sólo requieren ver una imagen para captar el mensaje. Un ejemplo claro de este proceso lo encontramos en la representación de las estaciones del Sistema de Transporte Colectivo Metro. Este uso de imágenes para transmitir información no es nuevo, tiene su origen desde hace siglos. Se puede citar a la escritura creada y desarrollada por diversas culturas a través de los tiempos: los Mayas (siglos X-XV) y los Egipcios (siglos XXVII-XXII a.C.). Todos ellos usaron elementos gráficos para expresar ideas, y a este concepto se le conoce como ideograma o ícono. Ícono, según el Diccionario Webster , es una imagen, representación, ilustración, grabado o esquema utilizado para representar un concepto, idea, dato u operación. Diferentes estudios han permitido llegar a la conclusión de que los procesos mentales son guiados más rápidamente por estructuras gráficas. El uso de este tipo de elementos agiliza y simplifica la comunicación, en donde han tenido una gran aceptación y su uso se incrementa rápidamente.

Existen algunas razones que invitan a utilizar elementos visuales dentro de un ambiente

computacional. A continuación se citan algunas ventajas de usar este tipo de representaciones [6]:

- Las imágenes son más didácticas que las palabras como medio de comunicación. Pueden transmitir más información por unidad de expresión.
- Las imágenes ayudan a entender y recordar
- Las imágenes pueden ser un incentivo para aprender a programar
- Las imágenes no están atadas a las barreras del lenguaje. Cuando se crean adecuadamente, son entendidas independientemente del idioma que se hable.

Con lo anterior no se quiere decir que la meta de los lenguajes visuales sea representar todo tipo de ideas y acciones mediante íconos sin incluir texto; la finalidad es usar de manera armónica los dos tipos de representaciones para integrar ideas más claras, logrando una comunicación más eficaz y eficiente. Debido a estas características, el número de áreas en las que se pueden emplear lo íconos como objeto de información es cada vez mayor. De estas, la que muestra más crecimiento es la de la computación. Algunas de las virtudes que adquieren los lenguajes de programación actuales al aplicar este tipo de metodología de creación, es que se ven enriquecidos con representaciones en dos o tres dimensiones, con atributos de color e iluminación, y con la posibilidad de representar este tipo de visualizaciones en forma estática y dinámica.

1.2. Motivación

Gran parte de los usuarios que utilizan las herramientas computacionales para realizar sus tareas no están familiarizados con el uso de una computadora o no tienen experiencia, llegan a tener problemas con el uso de estas ya que no les son intuitivas, no son descriptivas o sólo es texto en panatalla sin ninguna ayuda o pista gráfica, entonces se pueden llegar a desesperar o incluso a no usar más estás herramientas y preferirán volver a utilizar la manera en que tradicionalmente realizan sus tareas y esto podría implicar que no se hagan uso de la computadora ni de su gran poder. El área en que se quiere centrar esta tesis es en las herramientas para la recuperación de información y el análisis y la visualización de datos.

Con base en esto es motivo se tiene la hipótesis de que crear una herramienta con un lenguaje y entorno viusales logrará simplicar estas tareas y que además serán intuitiva para el usuario que con sólo ver la herramienta pueda deducir hasta cierto punto la funcionalidad y opciones de las que dispone; que sólo se tenga que añadir al espacio de trabajo una serie de íconos que reresentan funciones o procesos, después especificar el orden estos ícono y unirlos de tal forma que la salida de un ícono pueda ser la entrada para los subsecuentes íconos puede resolver este problema. Con LIDA [5] como base se puede desarrollar este proyecto.

El lenguaje estará enfocado en estadística descriptiva y análisis usando en lenguaje de programación R. Este es un lenguaje de alto nivel que ayuda al análisis de datos y un entorno para el análisis de datos [7]. Las características de R lo hacen una buena opción son:

- Se ejecuta en Mac OS, Windows, y varias plataformas Unix y Linux.
- Es gratis, de código abierto, y sometido a un continuo desarrollo y mantenimiento. Es una plataforma evolucionada pero estable con la que se podrá contar por muchos años.
- Tiene una extensa variedad de útiles funciones predefinidas y paquetes, y pueden ser fácilmente extendidas con técnicas de programación estándar.
- Tiene excelentes gráficas.
- Sus capacidades son similares a los excelentes programas comerciales y ampliamente utilizados pero muy caros tales como Matlab.
- Si es necesario para proyectos grandes y exigentes computacionalmente, R se puede utilizar para interaccionar con otros lenguajes de programación más rápidos.
- Una vez que se aprende su bastante simple sintaxis, es más fácil y más eficiente que una hoja de cálculo.
- Tiene muchas muestras de conjuntos de datos, los cuales ayudan a aprender a probar un programa.
- Es ampliamente usado en estadísticas, y es cada vez más utilizado en aplicaciones biológicas.

Las bibliotecas contienen funciones para estadística descriptiva como son las medias de dispersión y las medidas de tendencia central asi como también gráficas de pie, histrogramas, de caja, de dispersión, entre otras; y también funciones de estadistica inferencial que permiten hacer distintas predicciones con los datos que se tengan y los valores estadísticos antes mencionados [8].

1.3. Planteamiento del problema

La gran cantidad de datos generados por supercomputadoras y otras fuentes de datos hace muy difícil a los usuarios examinar cuantitativamente la información, es por ello que se requieren herramientas que faciliten el trabajo de observación, exploración y análisis de datos.

Los datos pueden ser secuencias de DNA, los modelos moleculares, el rastreo de imágenes médicas, mapas cerebrales, simulación de vuelo, simulaciones de flujos de fluidos, resultado de consultas a bases de datos, que por lo general en la forma en la que se presentan

no son útiles o por lo menos eso es lo que se podría llegar a pensar se muestran en forma de tabla con filas y columnas por ejemplo; pero qué pasa si estos datos se muestran de una forma gráfica, por ejemplo en una gráfica de barras, en un histograma, de manera casi inmediata uno puede realizar conclusiones, reconocer patrones o tendencias, identificar comportamientos, al igual que poder realizar comparaciones rápidamente entre distintos valores que estén plasmados en las gráficas, pudiendo ser una de estas comparaciones el tamaño las barras en una gráfica de barras.

Entonces la idea es crear un software, teniendo como base la programación visual, que sea capaz de encapsular esta funcionalidad, graficar y ofrecer mecanismos para poder leer y manipular estos datos, en un lenguaje de programación y un entorno visual para facilitar el uso de estos por los usuarios.

Con base en esto se plantean los objetivos en la siguiente sección para dar solución a este problema.

1.4. Objetivo general y objetivos específicos

El objetivo general de esta tesis es diseñar e implementar una herramienta visual para representar de forma gráfica frujogramas compuestos de íconos que representan funciones estadísticas, cuando estén listos dichos flujogramas ejecutarlos y mostrar el resultado en pantalla para que el usuario lo interprete.

Los objetivos particulares son:

1. Crear el lenguaje visual con el que trabajará la herramienta.
2. Diseñar e implementar un compilador del lenguaje REC con un enfoque estadístico.
3. Implementar una interfaz entre el lenguaje R y la herramienta visual.

1.5. Organización

La presente tesis se organiza en ocho capítulos con la siguiente estructura:

- **Capítulo 2:** Contiene el estado del arte en el área de programación visual mostrando una clasificación de esta y listando algunos trabajos en los que se aplicó.
- **Capítulo 3:** En este capítulo de describen los conceptos de la programación visual dando también su formalización que servirá de base para el desarrollo de esta tesis.
- **Capítulo 4:** En este capítulo se habla del lenguaje intermedio REC y de proyecto R, ambos indispensables para el desarrollo de la herramienta visual.
- **Capítulo 5:** En este capítulo se muestra el diseño de la herramienta con diagramas de clases, secuencia, casos de uso y también aspectos técnicos de las tecnologías que se usarán.

- **Capítulo 6:** En este capítulo se documenta la implementación de la herramienta de cómo se realizó, los problemas que se encontraron y cómo se solucionaron, además de explicar cómo se leen y guardan los archivos propios con la información de los programas visuales creados.
- **Capítulo 7:** En este capítulo se muestran algunos ejemplos utilizando la herramienta ya terminada así como un pequeño caso de estudio con la base de datos del SINAC.
- **Capítulo 8:** En este capítulo se hablan de los resultados, se discuten algunos puntos para llegar a las conclusiones y por último se habla del trabajo a futuro.

Capítulo 2

Estado del arte en Programación Visual

Los siguiente trabajos muestran algunas de las aplicaciones en las que se han usado lenguajes visuales, están ordenados según su característica más importante, incluyendo algunas imágenes de cómo es la estructura de leguaje visual que se implementó, y su aplicación usó para construir programas, consultas de una base de datos geográficas en las cuales los símbolos eran figuras que representaban ciudades o carreteras, o para hacer el diseño de software mucho más fácil, entre otros. Además habla acerca de algunas de las aplicaciones que se le han dado los lenguajes visuales principalmente en el área a de bases de datos, la sección 2.3, trata acerca de los lenguajes de consulta, se explica detalladamente LIDA desde sus inicio y cómo ha sido base para varias implementaciones posteriores de aplicaciones visuales.

La sección 2.1 se muestra una clasificación de la programación visual por paradigma, se encuentran los trabajos referentes a la programación, la sección 2.5 agrupa los trabajos referentes a la visualización de datos, la sección 2.6 muestra los trabajos en los cuales la programación visual se utilizó en la clasificación y el reconocimiento de patrones y por último la sección 2.7 se compone de los trabajos sobre lenguajes de modelado.

2.1. Clasificación de la programación visual

Cuando se examina el trabajo descrito en la literatura, se puede ver el proceso de programación visual en dos direcciones principales. En una dirección, las técnicas gráficas y los dispositivos apuntadores son usados para proveer entornos visuales para la construcción y ejecución de programas. En la otra dirección, los lenguajes son diseñados para manejar información visual (imágenes); soportar interacción visual; y para programar realmente con expresiones visuales. Estas tendencias son catalogadas en figura 2.1 y se describen a continuación.

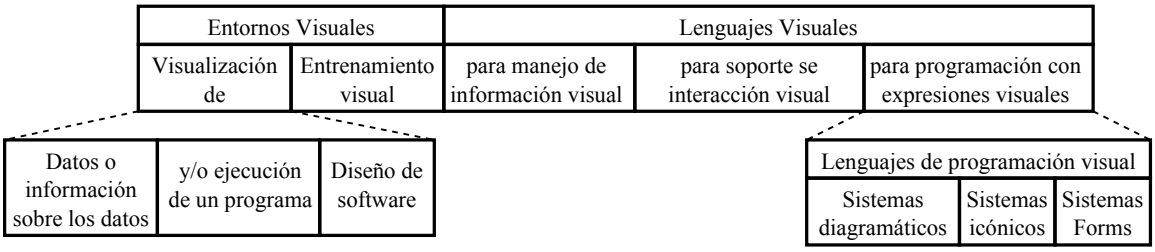


Figura 2.1: Programación visual.

Visualización de datos o información de los datos. Típicamente, la información o los datos son almacenados internamente en bases de datos, pero expresados en forma gráfica y presentada al usuario en un marco de trabajo espacial. Los usuarios pueden navegar por la superficie gráfica o enfocarla para obtener mejores detalles con un "joystick" o un dispositivo apuntador. Este enfoque permite que muchos tipos de cuestiones sean resueltas sin la necesidad de un teclado. En esencia, estos sistemas estás dedicados principalmente para usar una "manipulación directa" con un medio de recuperación de información, usando una vista gráfica de una base de datos para la visualización de la información recuperada.

Visualización y/o ejecución de programas. Otra categoría en el área de entornos visuales provee soporte para la visualización de programas, sus estados de ejecución y resultados. Los programas mismos son escritos en los lenguajes de programación tradicionales. El objetivo es usar las pantallas gráficas de alta resolución para hacer la tarea del desarrollo de programas y las pruebas más fácil. Las actividades en esta área abarcan un amplio espectro, que van de "impresiones bonitas" del código fuente para observar la ejecución en múltiples pantallas a formas animadas.

Visualización del diseño del software. Las técnicas gráficas se esperan que tengan una rentabilidad muy alta en un entorno de software que soporte un ciclo de vida de software completo. El objetivo es proveer un entorno de desarrollo de software en la cual los requerimientos, las especificaciones, el diseño de las decisiones y las estructuras de los sistemas son todos capturados en formas gráficas para la gente que lo diseñe, o dé mantenimiento a sistemas de software o que quieran averiguar sobre el sistema.

Entrenamiento Visual. Un número de sistemas se han explorado en la programación visual con la intención de reducir la brecha entre le proceso mental y el proceso de programación de la solución de programas. En el entorno provisto por estos sistemas los usuarios no necesitan mentalmente visualizar los efectos de las instrucciones mientras construyen sus programas. Los efectos se llevan a cabo en la pantalla antes de sus ojos. El proceso de programación confía casi completamente en la interacción gráfica. El estilo de interacción, en general, "imita la manera formal en que explicamos los programas, mostrando

imágenes de los datos y definiendo los cálculos sobre ellos señalando las secuencias similar al movimiento de la mano". La sintaxis del lenguaje en el sentido tradicional está ausente desde punto de vista del usuario. Por lo tanto, usamos el término "Entrenamiento visual" para caracterizar esta categoría. Muchos de los sistemas "programación por ejemplo" pertenecen a esta categoría.

Antes de seguir adelante, note que las primeras tres categorías difieren la una de la otra debido a que se centran en la visualización de tres distintas clases de objetos. En la cuarta categoría (entrenamiento visual) va más allá de la visualización usando señalizaciones visuales como medio para programar. Por lo tanto, en una mirada más cercana, estas cuatro representan cuatro distintas categorías de la programación visual.

Mirando desde un nivel más alto, sin embargo, es claro que estas cuatro tienen características en común: (1) todas proveen un entorno visual que capturan el espíritu completamente de una nueva forma en que los humanos interaccionan con la computadora; y (2) no proveen nada nuevo en términos de un enfoque para los aspectos de los lenguajes en el proceso de programación. En otras palabras, el énfasis es la interacción no el lenguaje. Esta segunda característica marca una clara distinción las dos áreas principales de la programación visual: entorno visual y lenguaje visual.

Dependiendo de sus objetivos, los lenguajes visuales se pueden clasificar además en tres categorías: lenguajes para el manejo de información visual; para el soporte de interacción visual; y para programación con expresiones visuales realmente.

Lenguajes para el manejo de información visual. Los lenguajes en esta categoría son diseñados principalmente para el procesamiento de información visual o imágenes. Están motivados por la necesidad de tener lenguajes fáciles de usar para la manipulación y la consulta de información pictórica. Típicamente, los lenguajes permiten una referencia directa a las imágenes que son manejadas. Aunque, la información se maneja por los lenguajes involucran imágenes, los lenguajes en sí mismos son textuales.

Lenguajes para el soporte de interacción visual. Los avances en la tecnología del hardware han preparado la manera del uso de íconos u objetos gráficos como medios de comunicación con las computadoras. Sin embargo, sin el software, los íconos y las imágenes podrían no venir a vivir en el mundo de la programación. Por lo tanto, es natural que los lenguajes fueran diseñados para definir, crear, y manipular símbolos pictóricos. Los lenguajes en esta categoría, en general, soportan representaciones visuales e interacciones visuales, pero los lenguajes en sí mismos son textuales, no visuales.

Lenguajes de programación visual: un marco de trabajo. Todavía otra categoría de los lenguajes visuales se concentra en permitir a los usuarios realmente programar con expresiones visuales. Pueden ser acertadamente llamados "Lenguajes de Programación

Visual" que más adelante de describirán. En esta categoría se tienen tres tipos que se definen a continuación.

Gráficas y Diagramas. Basados en los principios del diseño, muchos de los lenguajes de programación visual descritos en la literatura caen en estas tres amplias categorías. En un extremo, diagramas de flujo y diagramas que ya son usados sobre el papel son incorporados también en las construcciones de la programación, como extensiones para los lenguajes de programación convencionales, o creados en unidades interpretables por máquinas para que sean usados en la conjunción con los lenguajes de programación.

Lenguajes Icónicos o Pictóricos. En el otro extremo, los íconos o símbolos gráficos son diseñados deliberadamente para jugar el rol central en la programación. En los últimos años, se ha visto una oleada se sistemas icónicos o pictóricos en la literatura. Los roles de los íconos y las imágenes ya no se limitan a representaciones de objetos de escritorio e "instrucciones" para manipular estos objetos. Se han hecho intentos para enseñar y/o cumplir con los conceptos de programación mediante representaciones pictóricas.

Lenguajes de Tablas o basados en Forms. Entre los dos extremos (es decir, diagramas/gráficas y sistemas icónicos) caen en los lenguajes de programación visual del tercer tipo. En esta categoría, las representaciones gráficas son diseñadas como una parte integral del lenguaje. Sin embargo, a diferencia de los íconos en los sistemas pictóricos, estas representaciones no son las "superestrellas" del lenguaje. Sin embargo, a diferencias de los sistemas diagramáticos, estos lenguajes no son intentos para hacer "herramientas de papel y pluma" ejecutables.

2.2. Lenguajes visuales para bases de datos

Un *Lenguaje de Consulta* (LC) es un lenguaje de programación de alto nivel orientado a la recuperación y modificación de información almacenada en una base de datos de o en algún repositorio de información. Se asume que los usuarios de los lenguajes de consulta tienen experiencia limitada en aspectos técnicos. Por tal razón surgen los lenguajes visuales en una búsqueda por maximizar los atributos de usabilidad integrando nuevos paradigmas de lenguajes de programación, procurando integrar la riqueza semántica inherente a los componentes léxicos y sintácticos.

2.2.1. Funcionabilidad y usabilidad de los lenguajes de consulta

La mayoría de los nuevos lenguajes se sustentan sobre los lenguajes estándares ofrecidos por los sistemas manejadores de bases de datos más utilizados. Para estos se tienen nuevas tendencias, primero, los lenguajes de cuarta generación, ahora los lenguajes con características pictóricas. Con el afán de tener una caracterización cualitativa, se tiene

que la elección de un lenguaje, se condiciona invariablemente a dos aspectos: la funcionabilidad y su usabilidad. Descritas como dos componentes ortogonales describen un espacio de ubicación de los lenguajes que surtirán efectos y alcances. El resultado es una clasificación de los lenguajes de consulta a base de datos, según la figura 2.2. Estos dos aspectos deberán ser considerados en conjunto, pensando que en el desarrollo de nuevos lenguajes, se deberán enfrentar a ciertas fortalezas, debilidades y oportunidades. En el desarrollo de los lenguajes de consulta, de la gráfica se desprende, que siguiendo una línea a 45 grados se encuentran los lenguajes de consulta de una nueva generación.

En la figura 2.2 se resalta la tendencia hacia los elementos pictóricos o visuales. Esta categoría está en las fronteras de la nueva generación de los lenguajes de consulta. Las razones parecen claras, debido a que los atributos visuales contienen elementos semánticos autodescriptivos que facilitan la definición de esquemas o patrones complejos desde un punto de vista estructural, pero sencillos desde una perspectiva conceptual. Por esta razón resultó de vital importancia iniciar una serie de investigaciones sobre los paradigmas de la programación visual o pictórica, orientadas a la construcción de modelos estáticos y dinámicos, asociados a diversas etapas dentro de la construcción de aplicaciones computacionales. Estas etapas abarcan desde el análisis de un problema y recolección de requerimientos, pasando por diseños interactivos, hasta las fases de la concepción de soluciones, construcción e implementación de sistemas completos para la administración de datos.

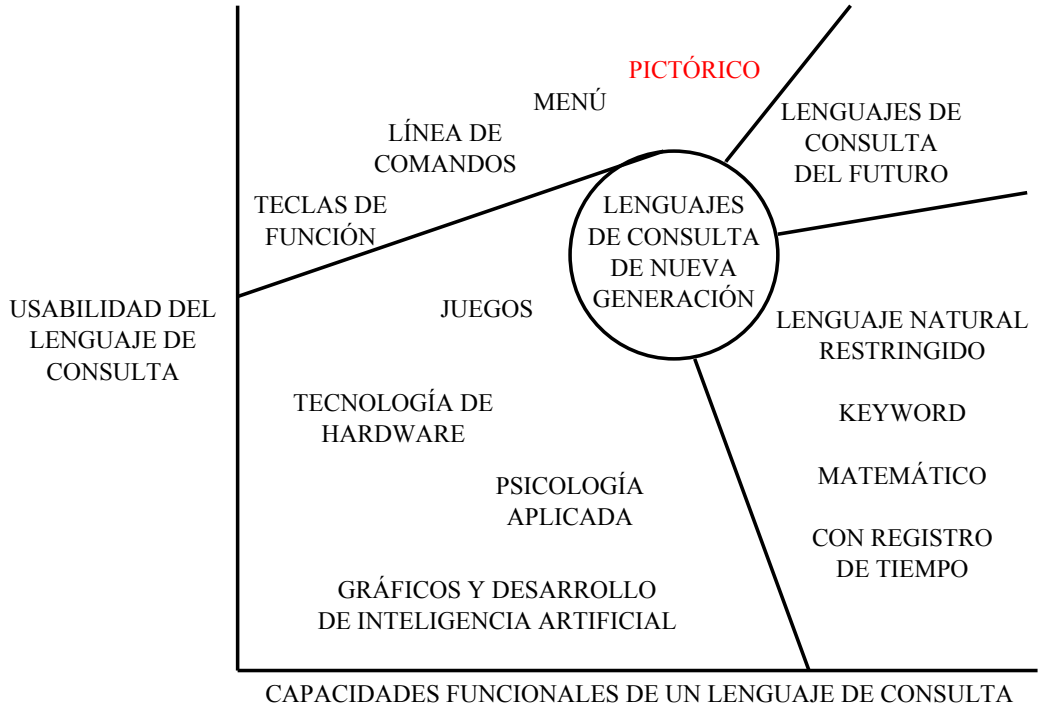


Figura 2.2: Tendencia en el desarrollo de los lenguajes de consulta en bases de datos.

2.3. Lenguajes visuales para consultas a base de datos

En esta sección se encuentran trabajos comparten la característica de que su función principal es recuperar información de bases de datos relacionales y geográficas a través de representaciones visuales de los objetos como son las relaciones, los campos, las operaciones de álgebra relacional; y para el caso de las bases de datos geográficas lo que son calles, ciudades, ríos, entre otras, además de implementar representaciones para expresiones aritméticas y lógicas para la construcción de las consultas.

2.3.1. LIDA

Lenguaje Iconográfico para el Desarrollo de Aplicaciones (LIDA) se ubica dentro de la categoría de lenguajes de programación visual [5]. Es un lenguaje de flujo de datos que tiene una descripción eminentemente visual. En su especificación los objetos de datos fluyen a través de las líneas de flujo de datos para entrar a módulos de transformación que son los íconos. Estos íconos tienen asociada una semántica a su imagen y como el uso de reglas sintácticas se disponen en una gráfica llamada *flujograma*.

Los *flujograma* son diagramas que representan las funciones concernientes al manejo de la información, dentro de la estructura de la organización y la transformación de la información misma. Son diagramas construidos principalmente de íconos que especifican procesos y arcos que especifican el flujo de la información. De esta forma los flujogramas resultan ser un modelo apropiado para describir el procesamiento de los datos para la automatización de los procedimientos, realizar aplicaciones y tener consultas a una base de datos. En la figura 2.3 se muestran la estructura de la interfaz principal. En ella se pueden apreciar dos regiones básicas: edición visual y monitor SQL. La primera región permite construir las expresiones visuales del lenguaje y la segunda muestra operaciones que se realizan a la base de datos y que están ligadas a la expresión visual.

Los resultados de las operaciones descritas visualmente, pueden visualizarse como tablas. En este punto el usuario tiene la opción de generar un archivo en Excel o uno en formato PDF para su distribución. También es importante resaltar que el sistema LIDA cuenta con un esquema de anotaciones, mediante el cual el usuario puede incrustar notas o grabaciones sobre las observaciones o incidencias detectadas durante el proceso de estudio. Estas características se pueden apreciar en la figura 2.4.

En resumen, podemos decir que las principales características de LIDA son:

1. Se basa en el modelo relacional de datos.
2. En un lenguaje de flujo de datos.
3. Su programación visual, basada en flujogramas se construye de íconos y arcos.
4. Tiene la capacidad de ejecutarlo en paralelo,

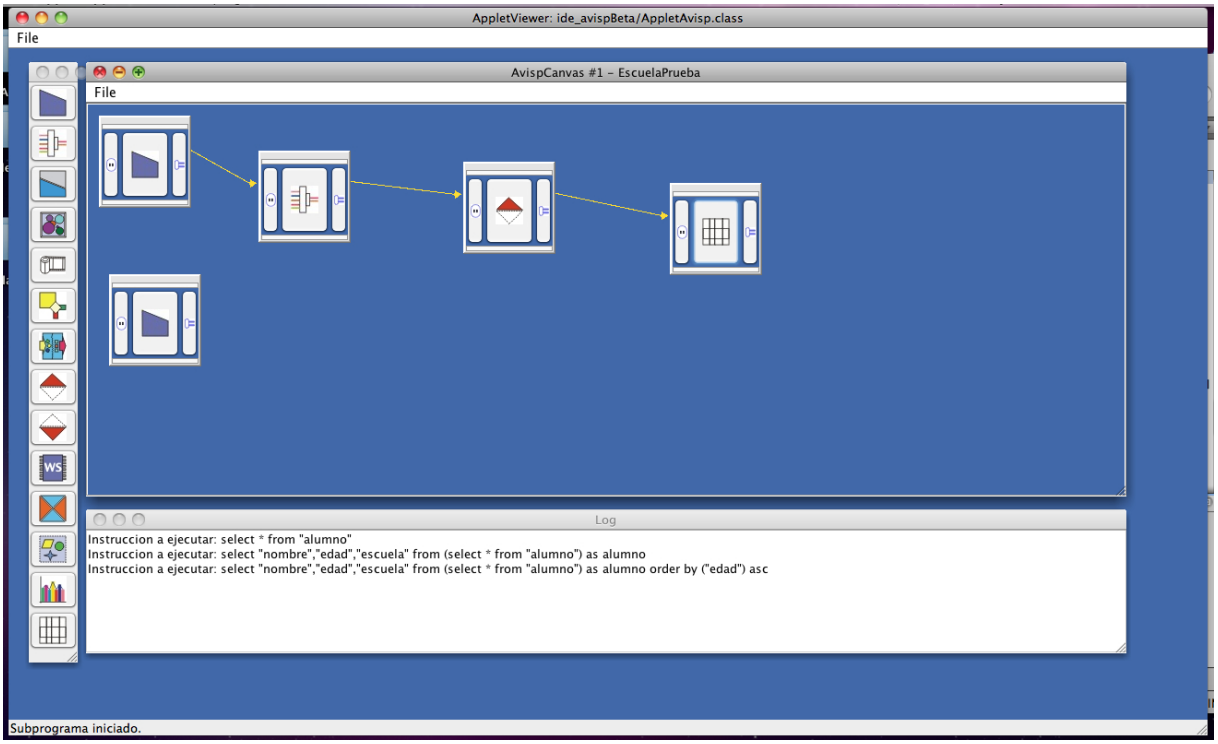


Figura 2.3: Pantalla de una sesión de trabajo con el sistema LIDA. Principalmente despliega un flujograma en el espacio de trabajo.

Una de las finalidades a mediano plazo es integrar un ambiente usando la gran variedad de lenguajes visuales para tener las herramientas necesarias desde la definición de procesos hasta la visualización de datos. El sistema de programación visual considera algunas extensiones a LIDA y la propuesta fue una interfaz de usuario basada en XWindows instalado en el sistema DEC3000. El editor gráfico como lenguaje visual, está construido por:

1. Diccionario de íconos, que corresponden a un conjunto de íconos elementales.

2. Diccionario de operadores, que es un conjunto de operadores genéricos que se aplican a íconos para crear íconos complejos.

3. Base de datos de acciones semánticas, que contienen la interpretación semántica asociada a los elementos del diccionario de íconos.

El editor gráfico se implementó usando VisualWorks de ParcPlace Systems. El ambiente de desarrollo VisualWorks es totalmente orientado a objetos y utiliza el lenguaje SmallTalk de ParcPlace. VisualWorks cuenta con mecanismos para reutilizar interfaces y el código de aplicaciones. La implementación del editor gráfico de LIDA se basa en la modificación del editor gráfico de VisualWorks, en donde se modificaron las clases principales que componen dicho editor gráfico.

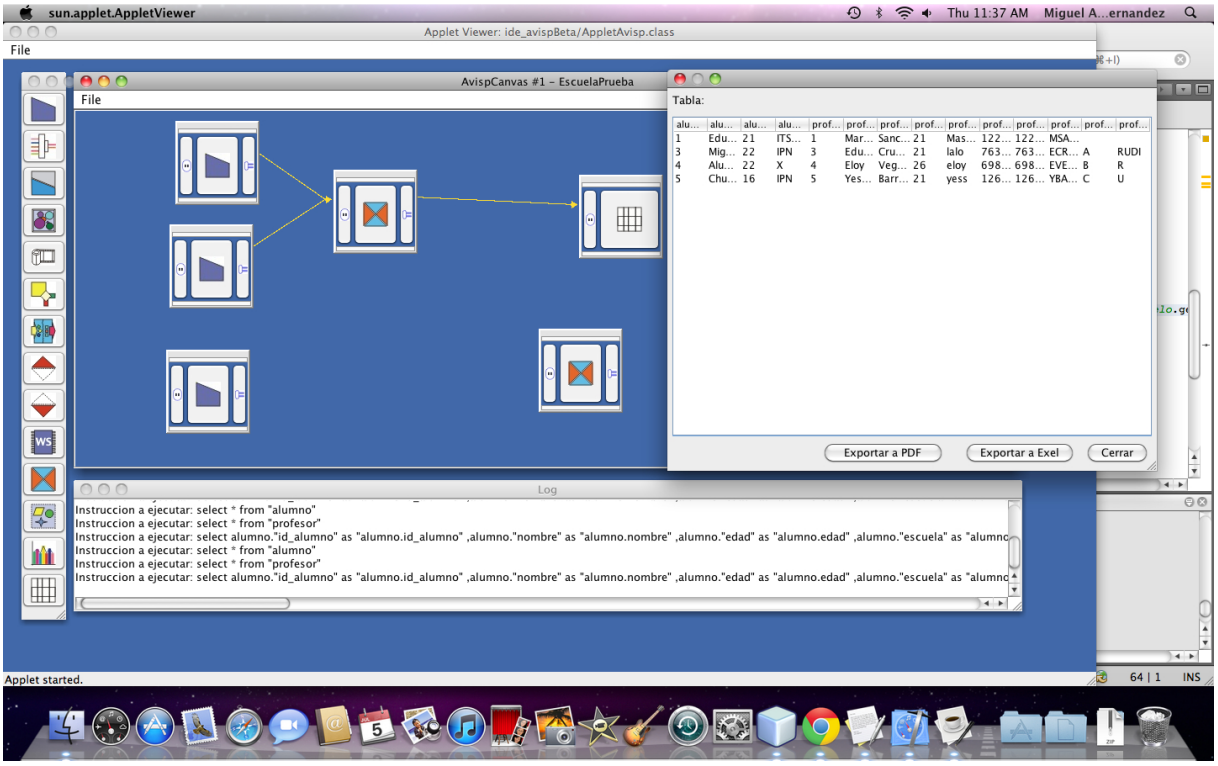


Figura 2.4: LIDA en una sesión de trabajo y el resultado de la ejecución de un programa visual.

El mapa de ruta del desarrollo de LIDA es el siguiente:

1. Creación de LIDA.- Implementado para PC, en lenguaje Pascal. Un ambiente visual, con un compilador de flujogramas e intérprete ISBL como lenguaje algebraico relacional. Un sistema manejador de base de datos basado en un descriptor de archivos [5].

2. LIDA en XWindows.- Un ambiente visual para LIDA desarrollado en VisualWorks de ParcPlace Systems, con una interfaz de usuario basada en XWindows instalado en el sistema DEC 3000. El editor gráfico como lenguaje visual, está construido por: el diccionario de íconos, diccionario de operadores y una base de datos de acciones semánticas.

3. LIDA/Web- Versión de LIDA orientada a sistemas cliente/servidor. Desarrollo de una interfaz de usuario, como cliente, con generación de código intermedio ISBL. Interconexión a un servidor con JDBC en un sistema Entreprise Data Base en una máquina Mac-G4 de Apple. Aplicación a la base de datos CDBB-500.

4. LIDA/REC.- Versión de LIDA orientada a sistemas cliente/servidor, con la sustitución de lenguaje intermedio ISBL por el lenguaje REC/BD. El lenguaje REC,

[9, 10] es desarrollado para su versión en Base de Datos, admitiendo operadores relacionales y operadores de aplicación de procesamiento de datos y funciones.

5. Desarrollo de LIDA para la gestión administrativa. La aplicación de LIDA para la automatización de las oficinas es un proyecto para ser implementado al Sistema de Información Académica (SINAC 2.0) para el CINVESTAV-IPN. En este proyecto la gestión de LIDA tiene tres niveles de aplicación: el primero como un workflow, segundo aplicación para base de datos, y tercero en ejecución de procesamiento de datos. El lenguaje intermedio es unificado con REC en diversas versiones.

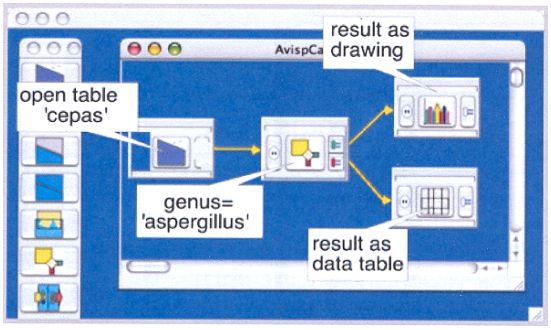
6. Desarrollo de LIDA para la bioinformática. Actualmente, se encuentran en desarrollo dos proyectos enfocados a la bioinformática. El primero, como lenguaje gráfico de workflow que carga procesos en la red para análisis de secuencias. El segundo toma algún proceso en particular y se construye un flujograma estadístico que hace interfaz con el sistema R que genera visualización de estadística descriptiva y análisis.

La última versión de LIDA es la que se ha mostrado en las pantallas de sesiones de trabajo. Esta versión está en actual desarrollo y se está integrando como un "framework" para la experimentación con bases de datos científicas y experimentales. Este framework se ha denominado *AVISP*, por las siglas de Ambiente Visual para la Solución de Problemas.

2.3.2. MVLAB-LIDA

EL proyecto MVLAB-LIDA (Microbiological Virtual Laboratoy on LIDA), tuvo como objetivo el diseño y construcción de un sistema basado en LIDA, como herramientas visual para definir todo tipo de consultas a una base de datos científica. Para la exploración de la información almacenada de la colección de microorganismos CDBB-500 del CINVESTAV y de uso externo por la Conabio. El sistema debía incluir otras importantes funciones sobre cultivos microbianos, tales como análisis estadístico, procesamiento de imágenes y visualización de vídeo mediante servicios remotos para la investigación en microbiología a través de internet. El sistema fue diseñado y construido como un sistema de información con acceso remoto a recursos, con la capacidad de trabajar en ambientes Web y arquitectura cliente/servidor. El núcleo es un sistema visual que el usuario final usa para definir una consulta o procesamiento de información estadística y multimedia. El sistema permite el acceso de base de datos con imágenes y vídeo. El desarrollo fue realizado usando servidores Apple, PostgreSQL, como DBMS y las tecnologías: Java, XML, XSTL, JavaScript, JDOM. La figura 2.5a muestra el diccionario de íconos del sistema.

(a) Diccionario de íconos en MV-LAB.



(b) Programa visual en MV-LAB.

Este es un ejemplo MVLAB-LIDA, el primer ícono abre una relación en la base de datos (cepas). El segundo selecciona algunos registros de cepas, género=“Asperguillus”). La figura 2.5b representa el programa visual descrito anteriormente.

2.3.3. EVEX

El objetivo del sistema EVEX [11], se presenta como mejora de EVE [12], fue tener una herramienta visual automática que contiene módulos especializados en cada etapa del diseño de las bases de datos. Su implementación se llevó bajo XWindows, la cual agrupa al protocolo X para redes TCP/IP, las primitivas de graficación bajo Xlib y las bibliotecas de alto nivel Xtoolkits y OSF/Motif. En EVEX, los módulos de análisis y traducción de diagramas permiten obtener los esquemas relacionales automáticamente, asegurando que cumplen con las tres primeras formas normales de Codd, mediante un proceso de normalización. El diseño físico, permite transformar los esquemas normalizados a especificaciones en un lenguaje de definición de datos. En particular, el Sistema Manejador de Bases de Datos propio es CDataX [13], desarrollado como parte del proyecto de tecnología de base de datos. La figura 2.6 muestra una sesión en trabajo en la herramienta.

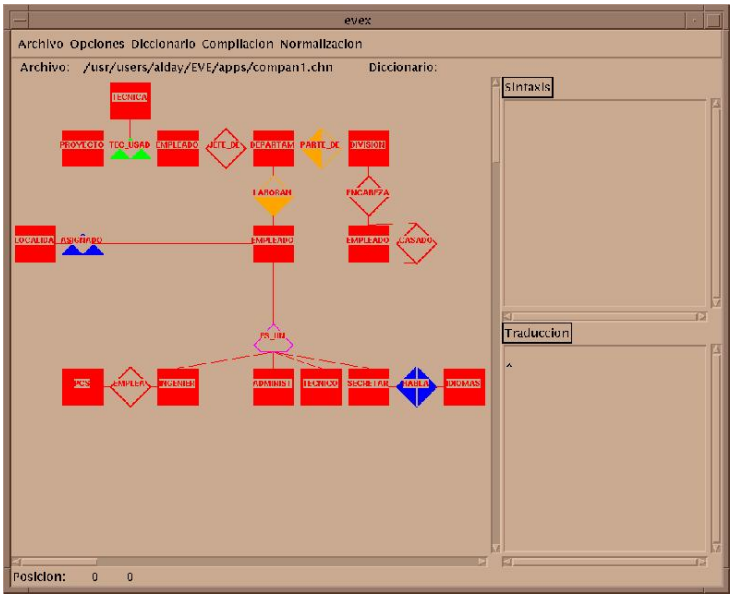


Figura 2.6: Pantalla de una sesión de trabajo en EVEX.

En resumen EVEX es un sistema visual que cubre con cada una de las tres etapas de base de datos: el diseño gráfico conceptual y el modelo gráfico y la descripción física.

1. **Diseño conceptual:** Editor de diccionario de atributos, editor de diagramas EVE y analizador sintáctico de diagramas.

2. **Diseño lógico:** Traductor de diagramas a esquemas racionales, normalizador de esquemas relacionales.

3. textbfDiseño físico: Constructor del lenguaje de definición de datos.

2.3.4. Kaleidoquery

En [14] se describe Kaleidoquery, un lenguaje visual de consultas para bases de datos de objetos con el mismo poder expresivo como OQL (Object Query Language). Se presenta cada una de las construcciones del lenguaje, dando ejemplos y relacionándolos con OQL. El lenguaje Kaleidoquery se describe independiente de los detalles de implementación, pero una interfaz 3D construyó para Kaleidoquery. Las consultas de esta implementación del lenguaje se convierten a OQL y luego se pasa a la base de datos objeto O_2 para su evaluación.

Se representan los objetos y relaciones como se muestra en la figura 2.7 y se arman las consultas uniéndolas con flechas que pueden estar etiquetadas con expresiones, también se implementa operadores lógicos como son el *and*, *or* y *not*, funciones de agrupación como el *count*, *min* y *max* y operaciones aritméticas, ver figura 2.8.

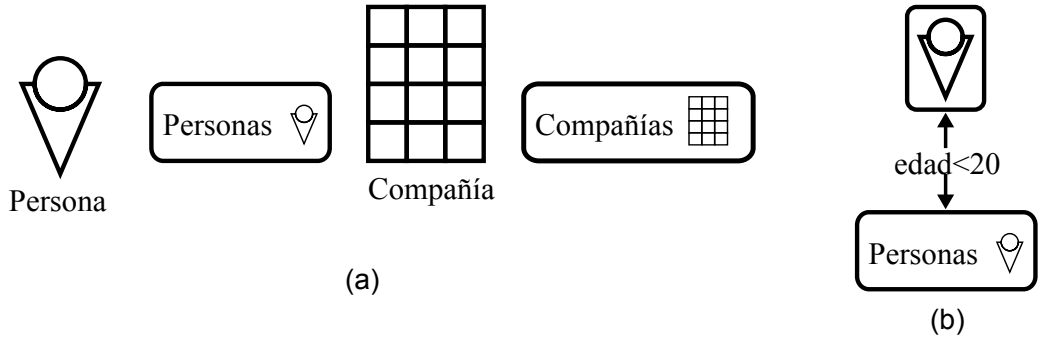


Figura 2.7: a) Ejemplo de esquema y b) una consulta simple.

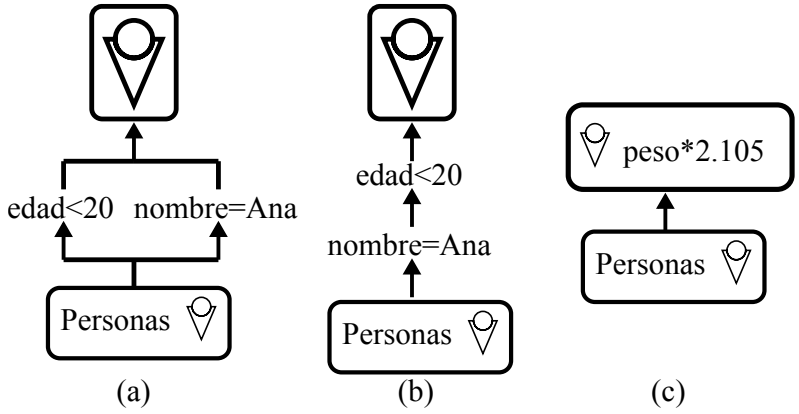


Figura 2.8: a) Ejemplo de un *and*, b) ejemplo de un *or* y c) ejemplo de una operación aritmética.

2.3.5. Lvis

En [15] se presentó la extensión del lenguaje Lvis que es un lenguaje visual basado en una filosofía de consulta-por-ejemplo para los datos espaciales ya que se han desarrollado aplicaciones orientadas a los ciudadanos usando los Sistemas de Información Geográfica (SIG), esta es la razón por la consulta visual parece ser crucial. Se proponen técnicas visuales, tales como globos y anclas con el fin de expresar criterios espaciales y temporales utilizando un modelo espacio-temporal. Las muestras se proponen en una aplicación de administración de caminos de riesgos. En esta base de datos de muestras temporales, tanto discretas y como continuas se tienen en cuenta: los puntos en movimiento, tales como camiones, y el ciclo de vida de los objetos, como los ríos.

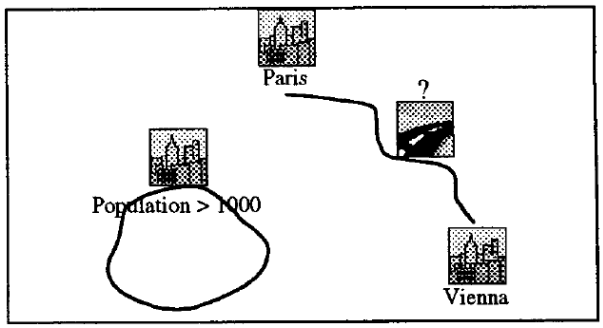


Figura 2.9: Ejemplo de consulta que resuelve la pregunta de cuántas rutas de Paris a Viena evitan pueblos de más de 1000 habitantes .

Las consultas espaciales se muestran construyendo bocetos en pantalla y se componen por medio de íconos que hacen corresponder a los objetos espaciales y los operadores, la cual es interpretada después por el sistema, ver figura 2.9. Lo que quiere decir que los operadores espaciales son directamente derivados desde el boceto, pero no son escogidos por el usuario y después se despliegan en pantalla, Además de que incorpora un editor del gráfico para el diagrama Entidad Relación que permite hacer consultas de partes que no son espaciales.

2.3.6. TVQL

En [16], se presenta los resultados de una evaluación de una interfaz de usuario para su lenguaje visual de consulta temporal (TVQL, por sus siglas en inglés). TVQL es una interfaz de consulta de manipulación directa única para especificar consultas relacionales temporales sobre eventos temporales tal como los datos de un vídeo. En su estudio de los usuarios, se comparó TVQL contra un lenguaje de consulta temporal basado en formularios (TForms). Sus resultados indican que los a los usuarios les tomó más tiempo aprender TVQL que TForms, fueron más eficientes y más precisos en especificar las consultas temporales con la interfaz TVQL que con la otra interfaz de llamada TForms.

2.3.7. Phenomena

En [17] se introduce un lenguaje de consulta visual, Phenomena, capaz de administrar los campos continuos, que representan acontecimientos del mundo real. En las aplicaciones de los SIG, los campos continuos representan los fenómenos relacionados con el medio ambiente y sus recursos. Recientemente, la atención se ha dedicado mucho a la vista del campo de los fenómenos geográficos, donde el mundo geográfico puede ser descrito por un número de variables, cada medida en cualquier momento. Mientras que los objetos se distinguen por sus dimensiones, y puede estar asociados con puntos, líneas o áreas, los campos se pueden distinguir por lo que varía, y la suavidad. Por lo tanto, cuando se trata de campos continuos, un requisito básico es representado por la capacidad de los usuarios a capturar algunas de las características de un escenario, mediante la selección de un área de interés y el manejo de los eventos heterogéneos involucrados. Phenomena hace esta tarea fácil de ejecutar, gracias a una representación visual muy intuitiva de ambos campos continuos y condiciones, que los involucran. Una característica peculiar del lenguaje visual es el uso de geometrías para seleccionar porciones de campos continuos, sobre la base de las condiciones espaciales. La heterogeneidad de los campos y objetos también se reflejan en la visualización de los resultados de la consulta. Implementaron operadores espaciales continuos como el área, concavidad, convexidad, puntos de inflexión, funciones estadísticas como la media, varianza moda, junta, reunión, entre otros, ver figura 2.10.

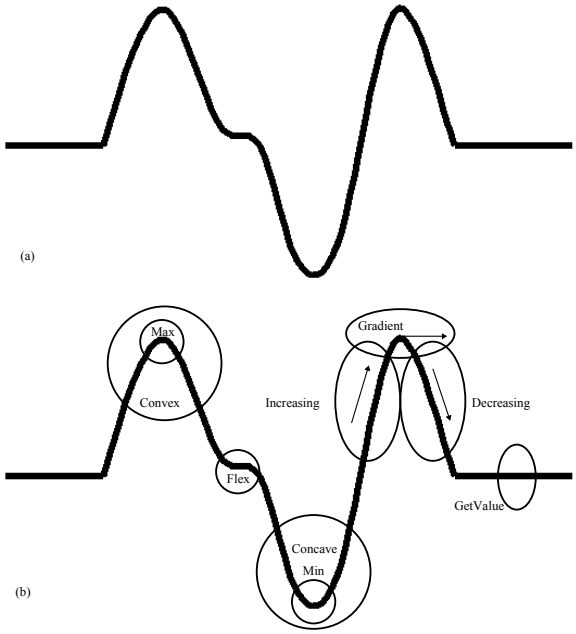


Figura 2.10: a) La metáfora función básica y b) sus zonas sensitivas correspondientes a los puntos críticos.

2.3.8. LIDA-Web

En [18] se muestra un sistema llamado LIDA/REC el cual consiste que a través de crear diagramas arrastrando y soltando íconos de operaciones en álgebra relacionales como proyección, selección y reunión de una forma válida genera reportes y consultas a una base de datos. Este proceso genera un código intermedio llamado REC el cual es la primera etapa que posteriormente sirve para generar el código en SQL para así ejecutarlo en el Sistema Gestor de Bases de Datos. La estructura del sistema se muestra en la figura 2.11.

El editor se llama LIDAWEB, está desarrollado en lenguaje de programación Java, es ahí donde el usuario puede generan los frujogramas para que después producir el código REC hecho especialmente para bases de datos en un archivo con extensión *txt*, después este archivo de texto es procesado por el compilador de REC, este ejecuta las instrucciones y envía las consultas a través de ODBC a PostgreSQL que posteriormente le envía los resultados de regreso, el programa en REC coloca estos resultados en un archivo de texto.

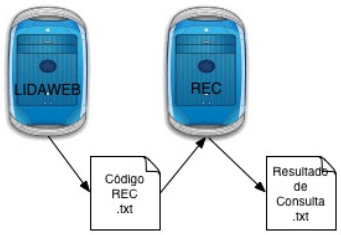


Figura 2.11: Panorama General del Sistema LIDA/REC

2.4. Lenguajes visuales para la programación

Los trabajos en esta sección comparten la característica de que el lenguaje visual se utiliza para crear programas mediante representaciones gráficas, que una vez que se ha armado dicho programa se compila y ejecuta, obteniedo los mismo resultado como si se realizara de forma tradicional con un lenguaje textual.

2.4.1. HEVICOP

Con la idea de simplificar los flujogramas se pensó en un nuevo paradigma (BLOX) como alternativa de representación visual de piezas geométricas que se pudieran ensamblar entre ellas con sintaxis dirigida. Con esta visión se suprimían las líneas de flujo de datos, pensando en representaciones gráficas más compactas, más elegantes y con una idea de programación lúdica. En 1953, el matemático Colomon Golomb tuvo esta idea. De esta forma, se tienen piezas de software las cuales son elementales, se pueden ensamblar para construir automáticamente programas; así, de manera constructiva se elaborarán programas más completos [3].

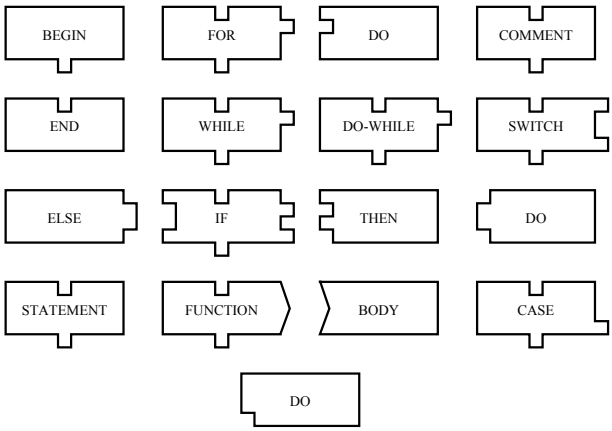


Figura 2.12: Figuras de las estructuras existentes en HEVICOP

BLOX es un medio ambiente gráfico y textual que puede ser enfocado hacia la representación visual de ciertas actividades, una de las cuales es la representación visual de programas, es un acrónimo de la palabra Blocks y una contracción de BLack bOXes. Con este enfoque cada bloque sólo se muestra superficialmente al usuario, teniendo la posibilidad de contener un nuevo conjunto de bloques encapsulados. El mundo BLOX tiene ciertas ventajas de representación y construcción de diagramas. Usado dentro de un sistema visual de programación reúne un conjunto de propiedades inherentes al sistema: el bloque, representado como un ícono, mantiene una dualidad; el diccionario de íconos del sistema; la sintaxis determina funcionalmente por la morfología de los íconos; las características estructurales de los lenguajes de programación son los que determinan el mundo BLOX que los representa.

En [3] se muestra herramienta llamada HEVICOP que es el acróstico de Herramienta visual para la construcción de programas, la cual mediante BLOX se pueden generar aplicaciones en el lenguaje C, los íconos representan las estructuras que tiene dicho lenguaje como son las sentencias *if*, *for*, *while*, *do-while*, *switch-case*, entre otros. Cuando está completado correctamente, la herramienta generará las sentencias en el lenguaje C

correspondientes a los íconos que lo conforman, además incorpora un analizador léxico y sintáctico usando las herramientas de LEX y YACC para validar que el código generado esté libre de errores y que se completamente compilable, o haciendo de forma inversa desde un archivo fuente escrito en lenguaje C. En la figura 2.12 se muestran algunas de los íconos empleados para la construcción de programas.

2.4.2. DENIM

En [19] crearon un lenguaje visual avanzado basado en bocetos que permite la creación de prototipos fáciles de grandes diseños complejos e interactivos ya que estas herramientas se asemejan a las prácticas de trabajo de los diseñadores de interfaces de usuario. En su forma de trabajo actual de la herramienta de diseño web DENIM, el lenguaje visual permite a los diseñadores crear bocetos de componentes reutilizables para elementos de página recurrentes, tales como barras de navegación, así como los condicionales para ilustrar y probar transiciones que dependen de la entrada del usuario. Los diseñadores también pueden especificar los sitios que aceptan la entrada de usuario más enriquecida que sólo hacer un click simple.

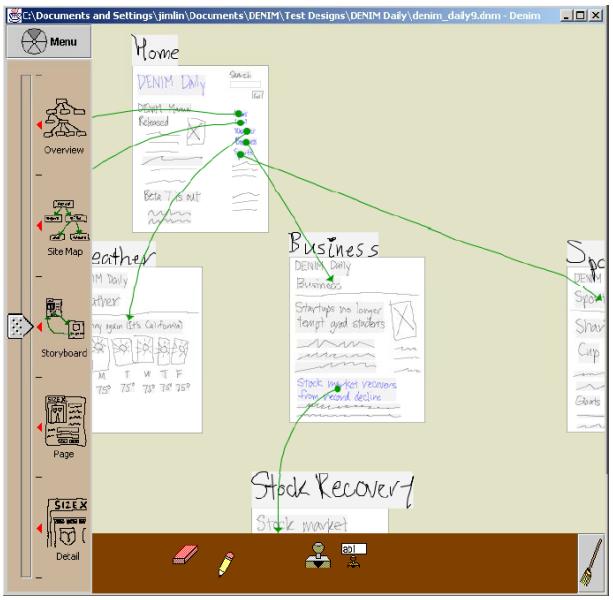


Figura 2.13: DENIM desplegando un sketch de 5 páginas web y 6 flechas de transición.

El lenguaje se compone de páginas y flechas, las páginas representan las páginas web, que se componen de dos partes: una etiqueta que describe la página, y un boceto que representa su apariencia física, ver figura 2.13, el diseñador puede dibujar o escribir en la página. Una flecha entre dos páginas representa una relación entre estas. Para crearla, el diseñador dibuja un trazo entre las dos páginas, si una flecha inicia desde un elemento en particular en una página, tales como una palabra, una imagen, o un botón, entonces el origen de la flecha se torna azul, como un hipervínculo en una página web. Además otros elementos están disponibles arrastrándolos dentro de la página como campos te texto, *radiobutton* y *checkbox*.

2.4.3. Visual Shading Language

En [20] se presenta un sistema para el desarrollo visual de los vértices complejos y sombreadores de fragmentos. El sistema hace uso de las ventajas de los lenguajes de programación visuales. El núcleo del sistema es un programa en Java. Con este programa los usuarios pueden desarrollar y probar diagramas de flujo de datos que describen la funcionalidad de los vértices OpenGL ARB y otros fragmentos de programas. Para obtener una información gráfica el sistema es capaz de mostrar escenas renderizadas y sombreado inmediatamente. La renderización de estas escenas tridimensionales se realiza con Java-OpenGL común enlazado con GL4Java, este último fue desarrollado y es un nuevo enlace para la conexión de Java con el Lenguaje de Sombreado de NVIDIA, *C for graphics* (*Cg*).

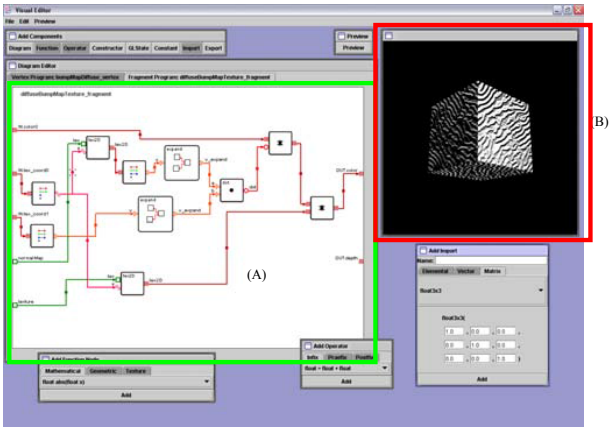


Figura 2.14: La aplicación junto con un flujograma (A) y su vista previa (B).

Para una manera fácil y rápida de la verificación, y la posibilidad de una futura integración en formatos basados en XML 3D como 3D extensible (X3D), la topología de los diagramas de flujo de datos se almacenan en un archivo XML. Una vista de la pantalla de la aplicación se muestra en la figura 2.14, que contiene el flujograma junto con su vista previa. El programa visual (programa de vértices o un fragmento) será transformado en diferentes lenguajes y condiciones antes de su ejecución. En el primer paso la topología del programa es exportada desde un archivo XML o un árbol DOM.

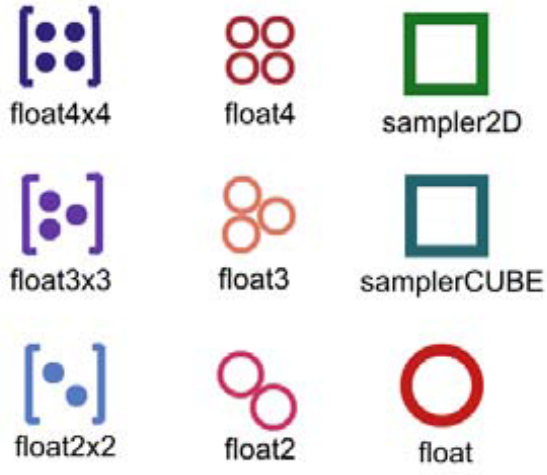


Figura 2.15: Íconos de algunos tipos de datos disponibles.

Mediante una procesador XSLT se realizará una transformación del programa al lenguaje de programación *Cg*. La generación de un programa ejecutable y la carga de un procesador correspondiente puede ser hecho con los recursos en tiempo de ejecución del *Cg*. La figura 2.15 muestra íconos de algunos tipos de datos disponibles.

2.4.4. ViTABaL-WS

La implementación de sistemas complejos basados en servicios web requieren de herramientas para describir eficazmente y coordinar la composición de los componentes de los servicios web es por esto que en [21] se ha desarrollado un nuevo lenguaje visual de dominio específico llamado ViTABaL-WS basado en su trabajo anterior, y se construyó un prototipo de herramienta de diseño para soportar el modelado de interacciones complejas entre los componentes de los servicios web. ViTABaL-WS utiliza una metáfora llamada "Herramienta de Abstracción" para describir las relaciones entre las definiciones del servicio, y múltiples puntos de vista del flujo de datos, flujo de control y propagación de eventos en un proceso de modelado. La herramienta es compatible con la generación de las definiciones del modelo *Business Process Execution Language* (BPEL). Es compatible con el modelado de los dos eventos de dependencia y de flujo de datos en el diseño de composiciones complejas de servicios web utilizando una notación visual.

La figura 2.16 muestra varios diagramas ViTABaL-WS que ilustran ejemplos de primitivas de composición en el paradigma de la abstracción de herramientas. Los servicios web son representados por óvalos verdes encapsulan el procesamiento de datos e interaccionan unos con otros a través de ambas invocaciones operacionales directas e indirectas compartidas utilizando estructuras de datos y las dependencias controladas por eventos que indican cambios de estado. Los servicios están conectados entre sí mediante puertos apoyando sólo ciertos tipos de mensajes y conexión. Estos son generados por un puerto de salida de servicio se distribuyen a los puertos de conexión de entrada de servicios web. Muchos esquemas de interconexión se apoyan incluyendo flujo unidireccional, petición-respuesta, el flujo asíncrono, y la recepción de notificaciones. Los controles adicionales soportan el flujo condicional, la comprobación de tipo dinámico, sincronización, iteración, etc.

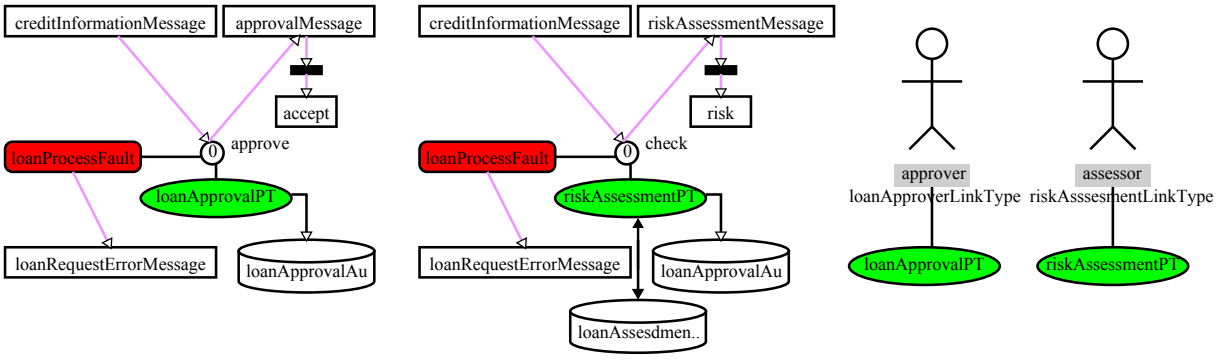
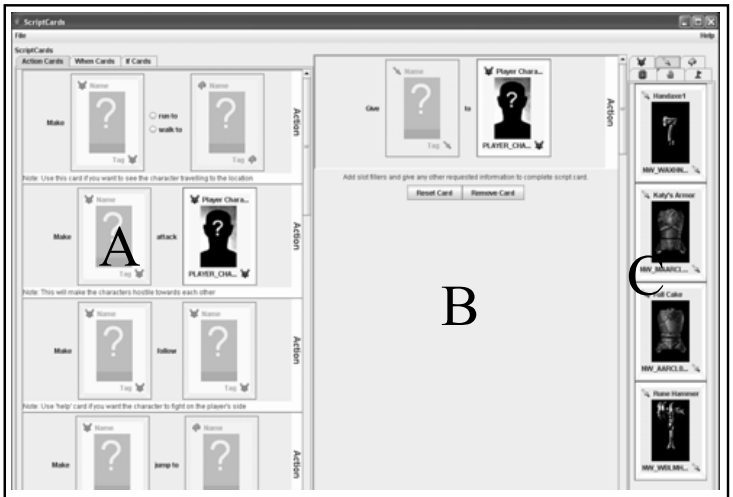


Figura 2.16: Ejemplos de composiciones de servicio web en ViTaBaL

2.4.5. LCD

La creación de historias interactivas en forma de los juegos de base narrativa puede tener beneficios motivacionales y educativos para los niños. En [22] se describe un enfoque de diseño centrado en el alumno (LCD) para la creación de un lenguaje de programación visual para eventos de scripts basados en historias de juegos interactivos. El enfoque de LCD, inusual en el ámbito de los VPLs, es esencial para asegurar que el software soporte eficazmente a sus usuarios niños que son el objetivo en el proceso de creación del juego. Un prototipo de sistema de alta fidelidad fue implementado en Java.



A: **Panel de Tarjetas Script**: Actualmente despliega *tarjetas de acción* listas para ser utilizadas en un gráfica de eventos.

B: **Panel Compositor**: Una tarjeta de acción ha sido agragada al panel. El usario ahora necesita llenar la ranura del "objeto" arrastrando sobre la tarjeta desde el panel de las *tarjetas embonadoras.* Despues pueden ser arrastrados sobre las *tarjetas script* adicionalespara completar la gráfica del evento.

C: **Panel de Tarjetas Embonadoras**: Actualemte despliega *tarjetas de artículos* para que sean usadas para llenar las ranuras de las *tarjetas script*.

Figura 2.17: Interfaz de las Tarjetas Script.

Los eventos se construyen utilizando tarjetas de bloques de construcción conocidos como *tarjetas script*. Hay tres tipos diferentes: *acción*, *cuando* y *si*. Un conjunto de tarjetas de acción se puede ver en el lado izquierdo de la interfaz en la figura 2.17.

Las tarjetas script representan una instrucción que se comunica como una expresión de lenguaje natural con ranuras vacías donde un objeto específico necesita ser referido. Para componer un evento, se arrastra una tarjeta sobre el panel de la izquierda hacia el panel compositor y llena los espacios en blanco soltando las apropiadas para que “embonen” en el panel de la derecha evitando errores de sintaxis. Las seis categorías de tarjetas embonadoras son de *criatura*, *artículos*, *objetos colocables*, *puertas*, *comerciante* y las de *punto de paso*. Una selección de tarjetas de artículos se puede ver en el lado derecho de la interfaz en la figura 2.17. El sistema limita al usuario permitiendo sólo tarjetas del tipo correcto para que sean colocadas en las ranura de otras tarjetas usando el paquete Java Advanced Imaging. Las tarjetas y ranuras están codificadas por colores y utilizar los íconos para dar al usuario indicaciones visuales adicionales en cuanto a que una tarjeta se ajuste.

2.4.6. ReactoGraph

El flujo de control dentro de los lenguajes de programación visuales de flujo de datos tales como Prograph es difícil de entender debido a dependencias ocultas creadas por las estructuras de control que descomponen los bloques de código en vistas independientes. En [23] se ha creado un modelo que surge de la investigación sobre ReactoGraph, derivado de Prograph, permite que el código en una clase general de los lenguajes de programación de flujo de datos sea controlado basado en estructuras gráficas anidadas. Estas estructuras, cuyo objetivo es la eliminación de las dependencias ocultas, ofrecen una alternativa razonable a las estructuras de control de casos de Prograph. Sin embargo, también se encontraron algunos problemas de usabilidad como la baja tolerancia a la modificación del diseño de código y dificultades escalando el número de elementos de código a medida que aumenta la profundidad de anidamiento. En este trabajo se presenta un nuevo uso de diversas técnicas de visualización de software aplicados a estas estructuras de control anidadas con el fin de permitir la modificación efectiva y la escalabilidad de código para la comprensión mejorada y un rendimiento en la programación.

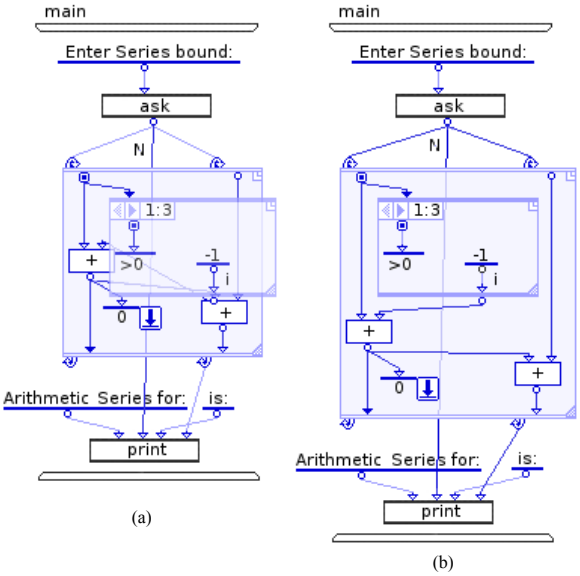


Figura 2.18: Ejemplo de diseño automático, a) diseño original y b) diseño ajustado automáticamente

Se enunciaron varios problemas a los cuales se les dio solución con las técnicas de transparencias para los objetos que están sobrepuestos, *holophrasting*, que es básicamente la contracción y expansión de elementos en los diagramas, y diseño automático que consisten en reacomodar los diagramas de una forma más adecuada, ver figuras 2.18.

La solución a estos problemas hace que los diagramas de flujo de datos sean más legibles y no sean sujetos a interpretaciones erróneas como resultado de un mal ordenamiento de los elementos visuales que lo componen.

2.4.7. Kaitiaki

Los usuarios finales necesitan a menudo la capacidad de adaptar las herramientas de diseño basadas en diagramas y para especificar los comportamientos dinámicos interactivos de interfaces gráficas de usuario. Sin embargo la mayoría quieren evitar tener que utilizar lenguajes textuales. En [24, 25] se describe un nuevo lenguaje visual para la especificación del manejo de eventos de la interfaz de usuario dirigidos a los usuarios finales. Este lenguaje visual proporciona a estos usuarios formas abstractas expresar ambos mecanismos de manipulación de eventos simples y complejos a través de las especificaciones visuales. Estas especificaciones incorporan filtrado de eventos, herramientas de consultas de estados e invocación de acciones.

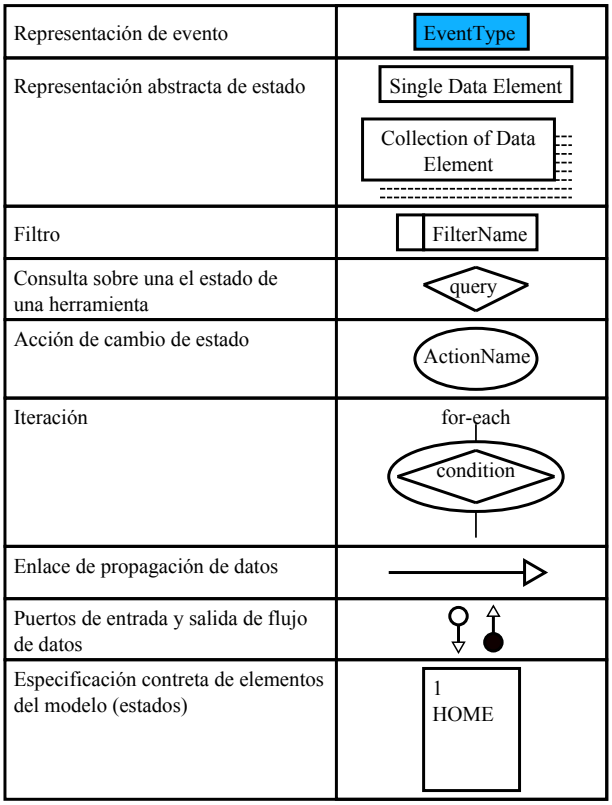


Figura 2.19: Construcciones visuales principales del lenguaje Kaitiaki.

La metáfora utilizada por Kaitiaki es un modelo "Evento-Consulta-Filtro-Acción". Esto se articula como "Cuando ocurra este evento, quiero que estos cambios se realicen en estos objetos". Las construcciones visuales son representaciones de eventos, filtros, herramientas de consultas de estados y acciones más iteración sobre colecciones de objetos de entrada y salida de los puertos y conectores de flujo de datos, y formas icónicas concretas, ver figura 2.19.

Se ha evitado el uso de estructuras de control abstractas y se agregaron a un paradigma de flujo de datos para reducir la carga cognitiva del usuario.

Un evento único o un conjunto de eventos son el punto de partida para una especificación de manejador de eventos en Kaitiaki. De este evento diversos datos fluyen hacia fuera (tipo de evento, objetos afectados, los cambios de valores de las propiedades, etc.). Las consultas, filtros y acciones están parametrizados con datos propagados a través de conectores de entrada. Los flujos múltiples son compatibles con conectores de flujo de datos múltiples que apunta a/desde una construcción visual. Las consultas recuperan elementos y salida de uno o más elementos de datos; los filtros seleccionan los elementos de su entrada; las acciones se aplican a las operaciones de elementos pasados a ellos.

2.4.8. VEISIG

Los patrones de diseño documentan exitosas soluciones a los problemas recurrentes en un dominio de desarrollo de software específico. Sin embargo, encontrar los patrones que se necesitan puede ser difícil y a menudo requiere que el diseñador comprenda una descripción narrativa larga para comprender los beneficios, implicaciones y compensaciones de cada patrón y de sus relaciones con los demás. En [26] se propone una notación visual con el apoyo de una herramienta de software que pueden ayudar a identificar patrones que podrían satisfacer los objetivos de un diseñador, así como la transmisión de las relaciones positivas y negativas entre los patrones, incluidas las dependencias y las colisiones entre los patrones seleccionados para un problema específico.

La notación en VEISIG. Una softgoal no tiene una definición clara o criterios en cuanto a si se cumple o no. Por ejemplo, "Diseña un sitio web usable y útil" puede considerarse una softgoal, ya que no hay forma de garantizar de una manera tajante que lo cumpla por completo, el diseñador debe simplemente aplicar heurísticas, directrices o pautas que ayuden a lograrlo. La figura 2.20 resume la notación VEISIG. VEISIG soporta no sólo la navegación, sino también la edición o modificación de cualquier componente en la notación.

VEISIG ayuda a identificar "¿Cuál es el modelo que necesito para resolver mi problema?" porque los diseñadores son presentados inicialmente con los objetivos que se deseen cumplir, en lugar de nombres de patrón, de modo que pueden darse cuenta del problema que pueden resolver cuando se aplica un patrón específico.

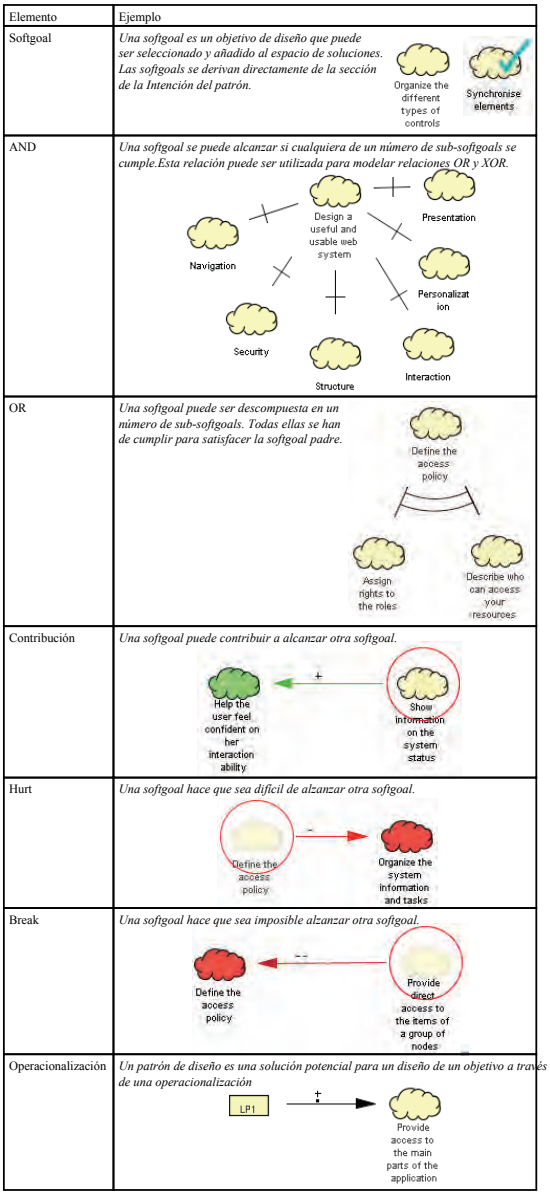


Figura 2.20: La notación de VEISIG para crear gráficas de interdependencia mejoradas visuales de lenguajes de patrones de diseño.

2.4.9. XDCL

En [27], se define un lenguaje visual genérico llamado XCDL basado en Redes de Petri Coloreadas permitiendo a los programadores no expertos para componer las operaciones de manipulación. El lenguaje está adaptado a XML, proporcionando a los usuarios de dispositivos componen operaciones orientadas de XML. El flujo de datos XML ha llegado más allá del mundo de la informática y se ha extendido a otras áreas como la comunicación de datos, comercio electrónico y la mensajería instantánea. La sintaxis del núcleo del lenguaje se presenta, junto con un prototipo implementado basado en Java. La siguiente figura se muestra el espacio de trabajo de la herramienta, en la parte superior se encuentran las notaciones visuales.

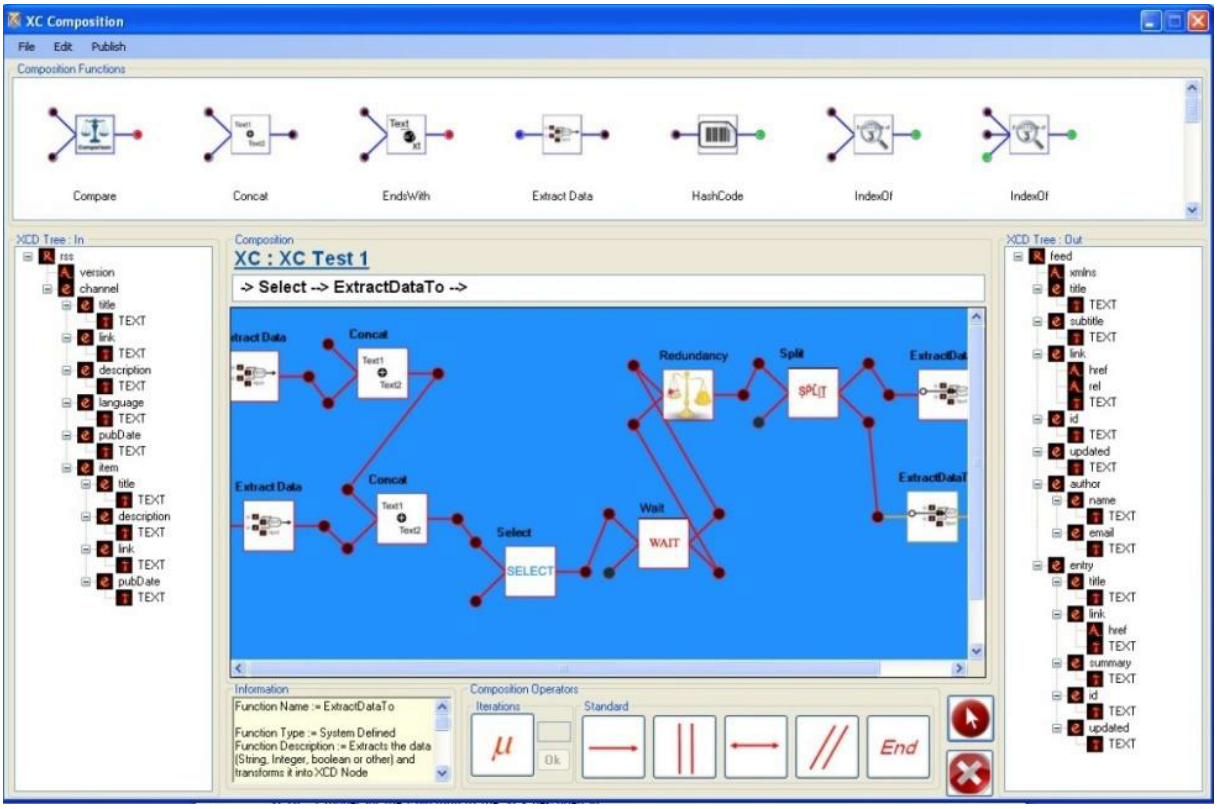


Figura 2.21: Pantalla de trabajo de XCDL.

2.5. Lenguajes visuales para Visualización de datos

En esta sección se encuentran los artículos en los que los datos se muestran de forma visual para que el usuario pueda darse una idea de estos datos además de que de esta manera el usuario puede sacar conclusiones y tomar decisiones, sin necesidad de .

2.5.1. DALI

En [28] muestran una herramienta visual para la representación gráfica de los datos. Esta herramienta utiliza los datos de salidas de que los flujogramas de LIDA dan como resultado después de ejecutarlos, estos datos provienen de realizar las operaciones de algebra relacional por lo tanto dan como resultado relaciones que se pueden representar como tablas, con estos datos se pueden generar distintos gráficos como son gráficas de curvas, gráficas de pastel, gráficas de barras o histogramas, las caras de Chernoff, ver figura , siendo esta última una de las aplicaciones más importantes de este sistema.

La estructura de DALI contiene varios módulos, ver figura 2.22:

a) Definición y Manejo de Archivos. El módulo de Definición de Archivos crea descriptores en donde se indica el nombre, tipo y tamaño de cada campo que compone el archivo. Una vez definido el archivo, el sistema permite introducir información. También permite leer y escribir desde y hacia el disco los archivos que se desean representar gráficamente.

b) Analizador de Datos. El módulo de Clasificador y Análisis verifica el tipo de datos que se desean visualizar, el número de campos o características del archivo, crea una Tabla Distribución y Matriz de Datos. Una vez creada la matriz, se analizan los datos para definir el tipo de gráfica más apropiada para su representación.

c) Constructor Gráfico. El módulo de Construcción Gráfica crea una estructura de datos para la representación gráfica denominada Graff, esta contiene toda la información que generó el Módulo de Análisis referente a la gráfica más apropiada. Toma esta información y construye la gráfica especificada utilizando las funciones de Xlib y Motif para su despliegue en pantalla.

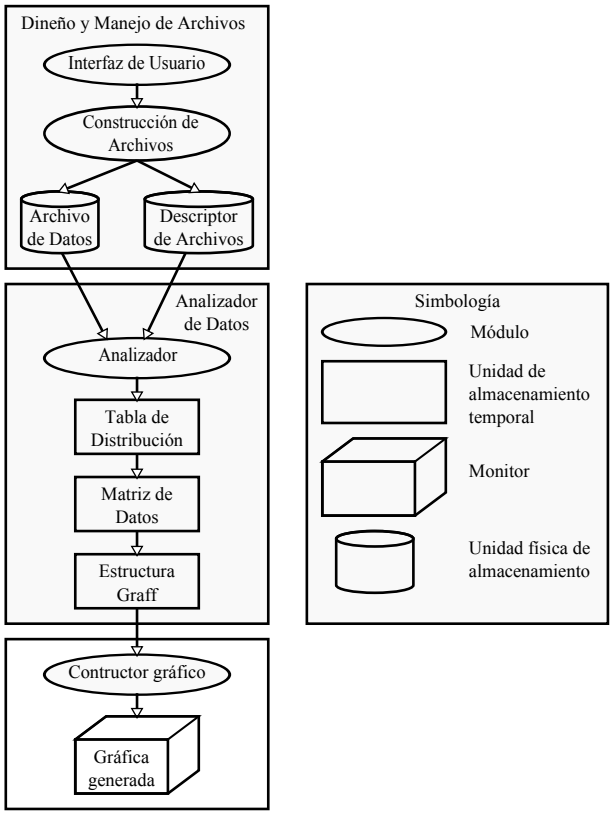


Figura 2.22: Estructura de DALI

2.5.2. LGraph

En el grupo de investigación del Dr. McIntosh, acerca del tema de autómatas celulares, surge el problema de gráficas con gran número de nodos y arcos, en particular los diagramas de de Bruijin. Así, se propone un tema de tesis donde surge LGraph [29], herramienta que nos permite editar, manipular y analizar gráficas complejas. Sin embargo, esto no significa que sólo puedan ser usados en este campo; de tal suerte, de se tiene herramienta valiosa para la edición y la configuración de gráficas, importante para bioinformática. Una de las aportaciones más importantes del proyecto es la definición de un lenguaje para la manipulación de gráficas. Usándolo, un usuario final puede aislar sesiones, colorear algunas características relevantes del diagrama, para de este modo facilitar su estudio y agilizar su comprensión. El desarrollo siguió la metodología Orientación a Objetos, en particular el uso de patrones. La figura 2.23 muestra la pantalla de una sesión de trabajo.

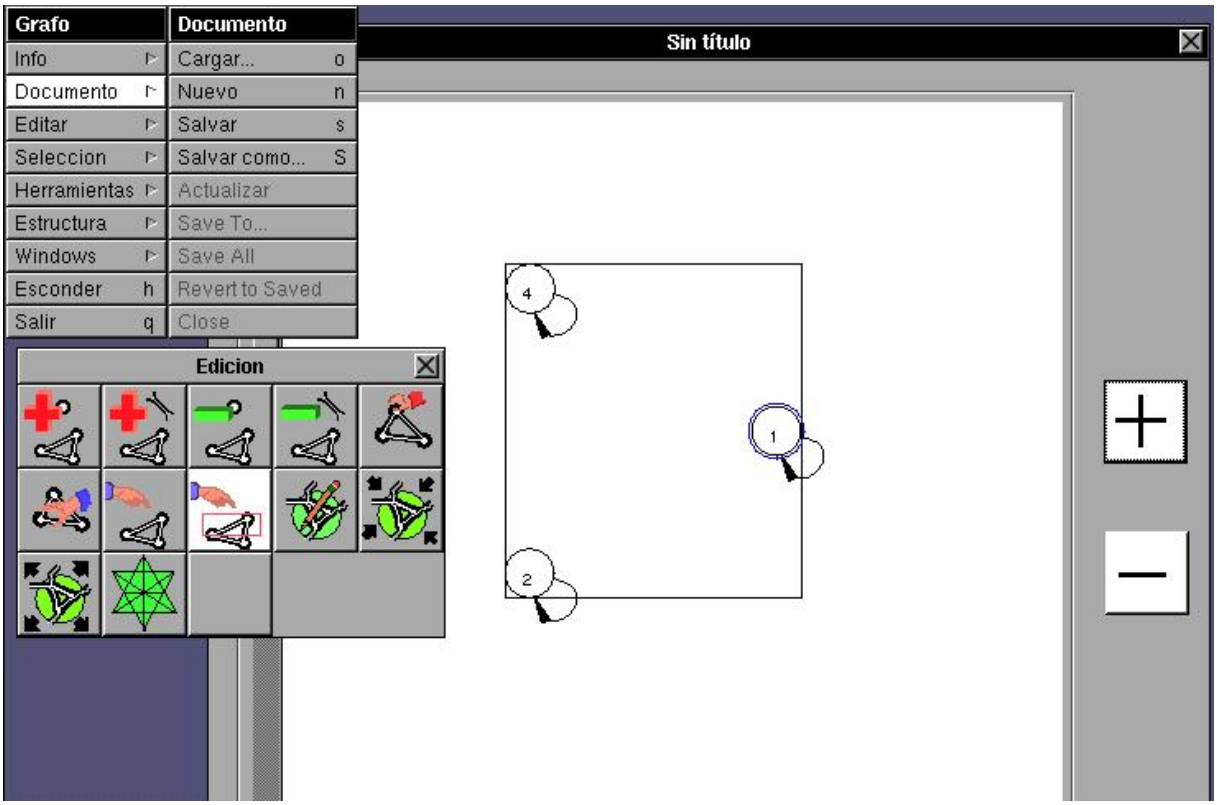


Figura 2.23: Pantalla de una sesión de trabajo con LGraph.

2.5.3. Sistema Dinámico de Consulta

En [30] se muestra un sistema de consulta estadística usando tecnología dinámica, que forma parte del proyecto SINAC propuesto por el Dr. Sergio Chapa para la administración de información académica, la tecnología llamada ColdFusion se compone de un lenguaje

de programación y un servidor de aplicaciones para compilar las páginas que se encuentren en ese lenguaje. La parte de la vista de sistema está implementada en ColdFusion así como los esquemas de seguridad para la autenticación de usuarios y la parte de la estadística descriptiva fue realizada en el lenguaje de programación Java, el cual no es un lenguaje para este propósito. Así desde la vista se seleccionan los datos y el tipo de la gráfica a mostrar, se envían estos datos y en un *applet* se muestran los gráficos resultantes, permitiendo personalizarlos. La tecnología ColdFusion no es software libre así que se tuvo que comprar la licencia para su uso.

2.6. Lenguajes Visuales para clasificación y reconocimiento de patrones

Los trabajos en esta sección tienen la característica de que el lenguaje visual que se aplicó se enfoca a la clasificación principalmente de imágenes además de que modelan algoritmos para dicha actividad, estás implementaciones son tanto en software como en harware, por ejemplo para ser ejecutadas por FPGAs.

2.6.1. VLMC

Aunque se ha estudiado durante muchos años, la clasificación de imágenes es todavía un problema difícil. En [31] se propone un método de lenguaje de visual de modelado para la clasificación de imágenes basada en contenido. Transforma cada imagen en una matriz de palabras visuales, y se supone que cada palabra visual es condicionalmente dependiente de sus vecinos. Para cada categoría de imagen, se construye un modelo de lenguaje visual utilizando un conjunto de imágenes de entrenamiento, que captura tanto la co-ocurrencia y la información de proximidad de las palabras visuales. De acuerdo a cuántos vecinos se toman en consideración hay tres tipos de modelos de lenguaje que pueden ser entrenados, incluyendo unigramas, bigramas y trigramas, cada uno de los cuales corresponde a un nivel diferente de complejidad del modelo. Dada una imagen de prueba,

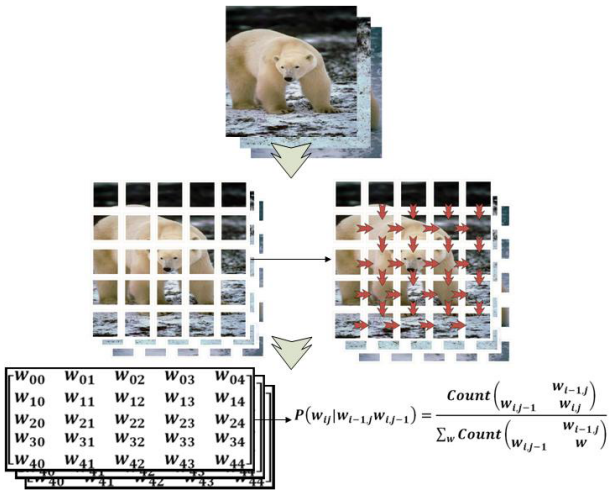


Figura 2.24: Proceso del entrenamiento del modelo del lenguaje triagrama

su categoría se determina mediante la estimación de la probabilidad de que se genera en una categoría específica. En comparación con los métodos tradicionales que se basan en los modelos de la "bolsa de palabras", el método propuesto puede utilizar la correlación espacial de palabras visuales eficazmente en la clasificación de imágenes. Además, se propone el uso de las palabras ausentes, que se refieren a las que aparecen con frecuencia en una categoría pero no en la imagen de destino, para ayudar a la clasificación de imágenes. Los resultados experimentales demuestran que su método puede conseguir una precisión comparable mientras el desempeño de clasificación va mucho más rápido. La figura 2.24 ilustra el proceso de análisis de una imagen.

2.6.2. VERTIPH

Las FPGAs se utilizan a menudo para el procesamiento de imágenes, pero las herramientas de diseño de FPGA carecen de construcciones sintácticas para algunas actividades especializadas que son importantes en este campo, tales como el tiempo, el manejo de los recursos y la programación. Esto obliga al programador trabajar a un nivel demasiado bajo y hace que sea difícil producir un diseño genuinamente descompuesto jerárquicamente. En [32] se esbozan estas deficiencias, como el fondo para una visión general y la justificación de cada una de las tres vistas en VERTIPH, un lenguaje de programación visual para la definición de algoritmos de procesamiento de imágenes en FPGAs. Se presentan los resultados de dos evaluaciones de usuarios de VERTIPH, una evaluación previa a la ejecución del usuario basada en papel no se encontró que se requirieran cambios importantes y una implementación-post-(parcial) de la evaluación del usuario.La representación de la Arquitectura de VERTIPH utiliza un paradigma de flujo de datos.

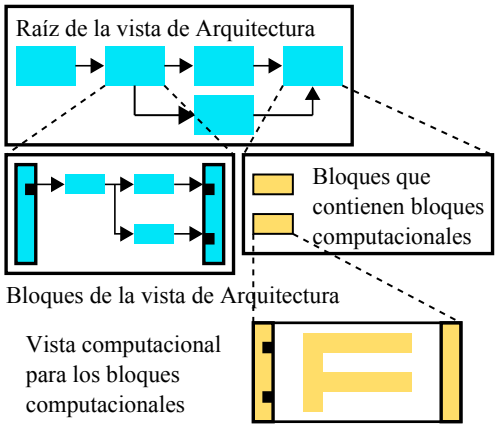
Así VERTIPH proporciona tres vistas gráficas, cada una representa un aspecto diferente de un algoritmo:

- La Vista de la Arquitectura de nivel superior: tiene por objetivo proporcionar al diseñador una perspectiva de todo el sistema y permitir la separación lógica de los operadores de procesamiento de imágenes, para mostrar el flujo de datos a través de los operadores, y para encapsular los datos relacionados con la operación y procesadores, ver figura 2.25a.

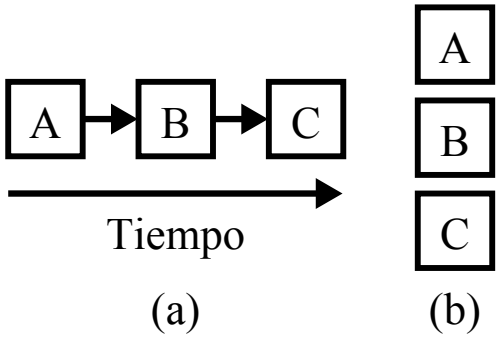
- La Vista Computacional: Los diseños FPGA generalmente comprenden redes paralelas o tuberías. Las operaciones paralelas y del tiempo, que debe ser precisados en tuberías, ambas implican relaciones complejas. Esto se adapta a una representación gráfica 2D más que una línea textual. VERTIPH (ver figura 2.25b y 2.26) representa operaciones secuenciales horizontalmente a lo largo de un eje de tiempo y (b) las operaciones paralelas verticalmente, como las tareas de diagramas de Gantt.

- La vista de Programación: Los procesadores (bloques de lógica computacional) en los sistemas complejos de procesamiento de imágenes en FPGAs necesitan ser programados, para evitar conflictos de recursos, para asegurar la correcta ejecución del programa, y para responder a los datos que llegan o que se soliciten. La ventana

inferior de la figura 2.27 muestra todas las opciones posibles para la programación de los procesadores como la vista Computacional. Sólo los procesadores concurrentes, marcados con (a), son relevantes para el algoritmo de histograma, los otros, (b) y (c) están ahí para ilustración. El tiempo aumenta de izquierda a derecha, los procesadores paralelos, identificados como (a), se apilan verticalmente, los procesadores secuenciales, etiquetados con (b), se dibujan de izquierda a derecha en una línea horizontal. Como una tubería es una combinación de procesamiento secuencial y paralelo, se dibuja una tubería, con la etiqueta (c), utilizando la misma notación que en la vista computacional.



(a) Visión general de la Vista de la Arquitectura



(b) a) Operaciones en serie y b) operaciones en paralelo

2.6.3. Third Eye

Trabajos de investigación existentes en el ámbito multimedia se centran principalmente en la indexación, recuperación, anotación, etiquetado, re-clasificación, de imágenes/vídeo, etc. Sin embargo, poco se ha contribuido al conocimiento visual de la gente. En [33], se propone un nuevo marco para extraer conocimiento de la gente visual a través de comunidades de múltiples idiomas. Dos retos se abordan: la representación de la cognición visual para una comunidad lingüística específica, y la comparación de la cognición visual entre diferentes comunidades lingüísticas. Lo llamaron "third eye", lo que significa que a través de esta forma las personas con diferentes antecedentes puedan entender mejor el conocimiento de la otra, y pueden ver el concepto con mayor objetividad para evitar el conflicto cultural. En este estudio, se utilizó el motor de búsqueda de imágenes para extraer la cognición visual de las diferentes comunidades. La suposición es que la distribución de la imagen semántica sobre los resultados de la búsqueda pueden reflejar la cognición visual de la comunidad. Cuando un usuario envía una consulta de texto, primero se tradujo a varios idiomas, y se introduce en los puertos de la imagen correspondientes motores de búsqueda para recuperar las imágenes de estas comunidades. Después

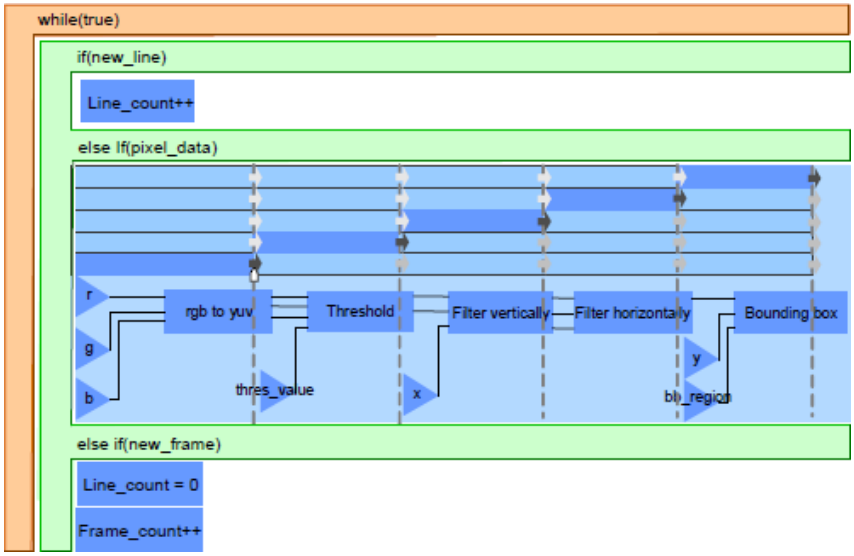


Figura 2.26: Tubería de base singular para la búsqueda de objetos

de la recuperación, las imágenes obtenidas se clasifican en diferentes grupos semánticos automáticamente. Por último, el ranking entre grupo semántico se emplea para clasificar los grupos semánticos de acuerdo con su relación con la consulta, mientras la clasificación intra-clustering se utiliza para clasificar las imágenes según su representatividad. La diferencia entre la cognición visual de estas comunidades indígenas se logra mediante la comparación de las distribuciones de la comunidad de imagen diferentes en estos grupos semánticos. Los resultados experimentales obtenidos fueron prometedores y muestran que el enfoque propuesto por la minería cognición visual es efectivo. La figura 2.28 muestra el marco de trabajo propuesto.

La tabla 2.1 muestra el paradigma de las herramientas que se describieron anteriormente, indicando detalles como son qué tecnología se utilizó para implementarla, principal área de aplicación

Cuadro 2.1: Íconos disponibles en el sistema

Herramienta	**Tecnología**	**Área de Aplicación**	**Paradigma**
LIDA	Pascal	Bases de Datos	Entorno visual Sistema Icónico
MVLAB-LIDA	Java PostgreSQL	BD Biológicas	Entorno Visual Sistema Icónico
EVEX	Xlib, Motif, CDataX	BD	Diseño de Software Sistema Diagramático
Kaleidoquery	OQL	BD	Entorno Visual

Continua en la siguiente página

Cuadro 2.1 – *Continuada desde la página previa.*

Herramienta	**Tecnología**	**Área de Aplicación Aplicación**	**Paradigma**
			Sistema Form
Lvis		BD Geográficas	Entorno Visual, Manejo de Información Visual
TVQL		BD Temporales	Entorno visual Manejo de Información Visual
Phenomena		BD Geográficas	Entorno Visual Manejo de Información Visual
LIDA-Web	Java, HTML		Entorno Visual, Sistema Icónico
HEVICOP	C, Lex, Yacc	Programación	Entorno Visual Sistema Diagramático
DENIM	Java	Programación Web	Diseño de Software
Visual Shading Languaje	Java, C, openGL	Programación	Diseño de Software Sistema Icónico
ViTaBaL-WS	BPEL, Java	Servicios Web	Diseño de Software Sistema Diagramático
LCD	Java, Advance Imaging	Educación	Entorno Visual Sistema Icónico
ReactoGraph		Programación	Visualización de Programas
Kaitiaki		Programación	Entorno Visual Sistema diagramático
VEISIG		Programación	Diseño de Software Sistema Icónico
XCDL	XML	Programación	Entorno Visual Sistema Icónico
DALI		Visualización de datos	Entorno Visual
LGraph		Visualización de datos	Entorno Visual
Sistema de Consulta Dinámico	ColdFusion	Visualización de datos	Visualización de datos o información sobre los datos
VERTIPH		Programación	Visualización de Programas
VLMC		Clasificación	Manejo de Información Visual
Third Eye	Google	Clasificación	Manejo de

Continua en la siguiente página

Cuadro 2.1 – *Continuada desde la página previa.*

Herramienta	Tecnología	Área de Aplicación Aplicación	Paradigma
			Información Visual

2.7. Lenguaje visual para modelado

Los trabajos en esta sección proponen una formalización para los lenguajes visuales para modelado, utilizando gramáticas, flujo de datos y lenguajes libres y naturales.

En [34] se presenta el enfoque basado en el formalismo de la Gramáticas Posicional Extendida para la especificación, diseño e implementación de lenguajes de modelado visuales. Con el fin de destacar las principales características del enfoque y resaltar su poder, se describe el uso del formalismo para implementar lenguajes de diagramas de estados que representan uno de los lenguajes de modelado visuales más complejos utilizados en el campo de la ingeniería de software. En el artículo se hace especial hincapié en la descripción de los beneficios derivados de la utilización de tales especificaciones formales como incrementalidad, la fácil personalización, y generación automática de entornos de programación visuales. Estas características resultan ser especialmente importantes porque los lenguajes visuales de modelado son sometidos a continuos cambios como la historia de los lenguajes de diagramas de estado y los diagramas de UML. Por otra parte, los lenguajes visuales pueden ser utilizados eficazmente sólo si se apoyan en un entorno visual de gran alcance dentro de ellos están incorporados y usados.

La Gramática Proposicional Extendida define a una sentencia visual un conjunto se objetos gráficos. Cada objeto es una instancia de un símbolo x el cual está definido como una tripleta (M, S, L) donde M es un conjunto de atributos que especifican la apariencia física del símbolo tales como el tamaño, el color, figura, entre otros; S es un conjunto de atributos, nombrados atributos sintácticos, cuyos valores dependen de la "posición" del símbolo en la sentencia; L es una estructura conceptual que define la semántica de un símbolo. Además de que manejan dos tipos de relaciones, la primera llamada *relación de conexión* se especifica en una secuencia de un número finito de regiones de unión como los atributos sintácticos. Las sentencias pueden ser construidas mediante la conexión de regiones de unión de los símbolos a través de los enlaces. Más de un enlace puede estar unido a una región de unión.

Una *relación espacial* se especifica en las coordenadas de los vértices superior izquierdo e inferior derecho del cuadro delimitador de los símbolos. Las sentencias pueden ser construidas mediante la combinación de símbolos a través de las relaciones tales como *contención* , *herencia*, *superposición*, entre otros. Como un ejemplo, el símbolo V en la figura 2.29 está relacionada a través de relación de contención con dos símbolos en su cuadro delimitador, etiquetado S y U. Utilizan Visual Language Compiler-Compiler (VLCC) que es un generador de entorno visual poderoso basado en esta gramática que

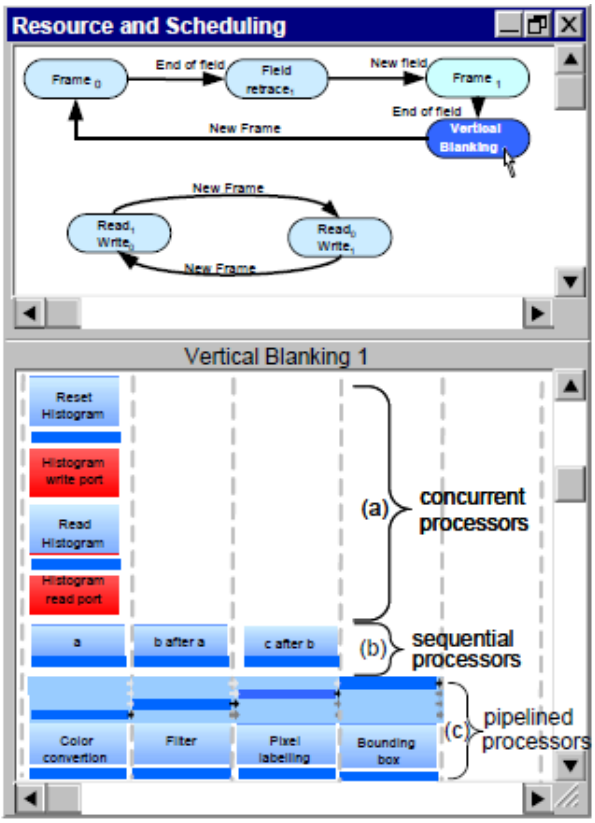


Figura 2.27: Editor extendido máquina de estados, Vertical Blanking seleccionado. En la pantalla de división inferior, los tres procesadores simultáneas asociados con el estado (con la etiqueta (a)), se muestran. Los procesadores secuenciales (b) y de tubería (c) no son parte de este diseño, sino que se han añadido para la integridad.

les sirve como herramienta para un rápido prototipado de lenguajes visuales complejos.

En [35] se muestra una formalización de una clase de lenguajes, lo que llamaron Flujo de Datos Controlado (CDL, por sus siglas en inglés). Este trabajo fue motivado por un estudio previo de un mecanismo de excepciones de los lenguajes visuales de este tipo, ya que para definir cómo el mecanismo de excepciones podría ser incorporados en cualquier CDL, se necesitaba un formalismo para capturar con precisión la sintaxis y la semántica de esta clase, incluyendo un protocolo para la inclusión de un lenguaje específico de estructuras de control. Para ilustrar el formalismo, se ofrece un ejemplo que muestra cómo se captura la ejecución condicional e iteración. También se reporta el uso de este formalismo como base para una implementación de nuevas herramientas de lenguaje CDL.

En [36] se propone una forma libre y natural para representar conceptos/ideas utilizando una disposición espacial de los símbolos gráficos, como íconos, líneas, flechas y formas. La ley de Gestalt de proximidad, la ontología, dinámica de secuencias de comandos, el razonamiento basado en agentes, y programación neurolingüística (NPL, por sus siglas en

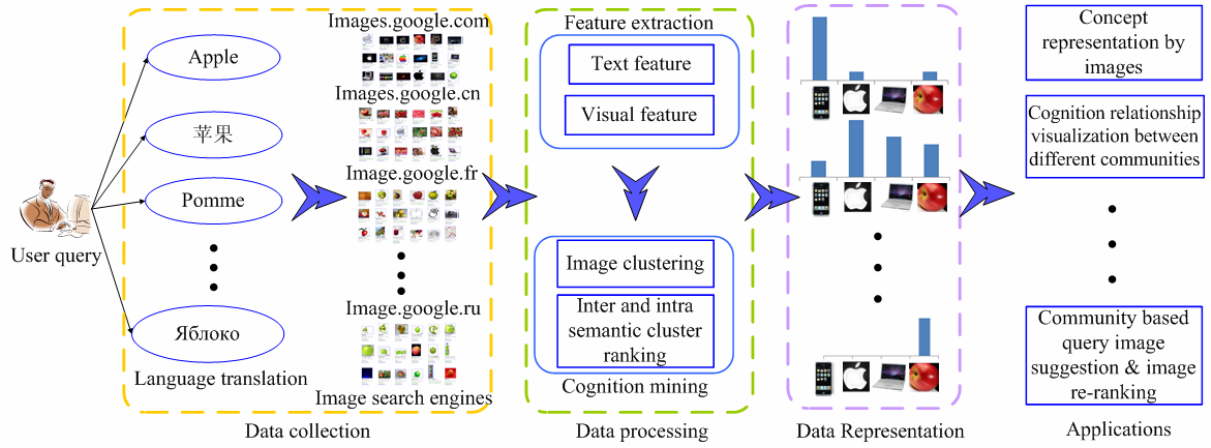


Figura 2.28: El marco de trabajo de la minería cognición visual propuesto.

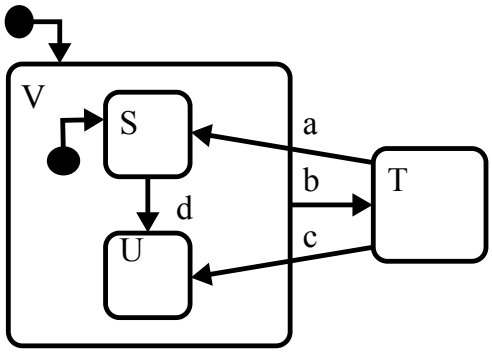


Figura 2.29: Ejemplo de sentencia visual.

inglés) se combinaron para la interpretación y la conversión de los mensajes comunicados. Unas pruebas con usuarios confirmaron que el lenguaje visual desarrollado podría servir como medio de comunicación.

2.8. Otras opciones que no son visuales

También hay opciones que resuleven este problema y tienen la característica de que no son lenguajes visuales, el mismo lenguaje R es uno de ellos, ya que através de sus funciones se pueden hacer estos análisis y se puede manipular datos. Matlab (Matrix Laboratory) también es uno de ellos y es directamente la compentencia de R, Matlab cuenta con su propio lenguaje llamando M con el cual se los usuario pueden crear y manupular matrices y arreglo, orefiendo mecanismo para poder leer completamente columnas o filas con sólo una intrucción, también ofrece funciones para graficar estas matrices.

Otra forma de acceder a esta información es a través de un sistema gestor de base de datos, que mediante en consultas escritas en SQL se puede recuperar la informac-

ción guardada en los repositorios, estos sistemas gestores también ofrecen funciones para análisis como son obtener el valor mínimo de una columna de valores, el valor máximo, el promedio, entre otras.

A su vez muchos de los jenguajes de programación ofrence bibliotecas de clases y funciones para porder hacer análisis estadístico con los datos primitivos de estos lenguajes ya sean enteros o doubles; como ejemplo tenemos la bliblioteca Java Statistical para Java, también en Statistic Library de .NET de Microsoft, jSat para JavaSrctipt, y así se puede seguir mencionando otros más; pero cabe resaltar que ese no es el principal objetivo del lenguaje a diferencia de R, es por eso que se decidió utilizarlo.

Capítulo 3

Formalización de la Programación Visual

A continuación se describen algunos conceptos útiles para el desarrollo de esta tesis, como es la Programación visual, su definición y cómo se clasifica, además se hace especial enfasis es lo que son los Lenguajes de programación visual, explicando cada una de las partes que los conforman detalladamente, incluyendo además los aspectos que los diferencian entre los lenguajes de programación tradicionales. Los lenguajes de marcado, concretamente el Lenguaje de Marcación de Hipertexto (HTML), que servirá como base para la implementación de la parte visual de este trabajo, mostrado cómo ha ido evolucionando a través del tiempo y resaltando las novedades de su ultima versión, HTML 5. También se habla de la visualización de datos, qué es y se dan algunos ejemplos para contrastar una representación en forma de tabla de información y una representación gráfica especificando las diferencias.

Debido a que la programación visual es un campo relativamente nuevo, se tienen una idea diferente de qué es. Se usa el término "Programación Visual" para referirnos al uso de representaciones gráficas significativas en el proceso de programación [37].

La programación puede ser definida como la especificación de un método para hacer algo que la computadora pueda hacer en términos que la computadora pueda entender. Hay muchos aspectos de la programación: los lenguajes y los entornos usados para las especificaciones; los métodos mismos; la determinación de si la computadora ha ejecutado una especificación como se esperaba; el despliegue de los datos involucrados en la ejecución de una especificación; etc. La programación visual puede ser aplicada a todos los aspectos de la programación. Una pregunta importante es: ¿hay algunos objetos gráficos significativos (no meramente decorativos) involucrados en un aspecto de la programación?

Uno puede argumentar si el uso de ventanas superpuestas o un mosaico de ventanas es programación visual. La respuesta merece más que un simple "sí" o "no". Teniendo múltiples ventanas es útil en muchas situaciones. Esto permite al usuario ver algunos objetos o controlar un número de actividades separadas más fácilmente.

Sin embargo, las ventanas no necesariamente contienen objetos gráficos pertenecientes a la programación, y la programación no necesariamente depende de múltiples ventanas. Por lo tanto, según esta definición, el mero uso de ventanas no es suficiente ni necesario en la determinación de un aspecto en la programación es visual o no. En otras palabras, el estilo de la interacción que emplea ventanas y dispositivos señaladores es incuestionablemente valioso, independientemente de si algunos objetos gráficos están involucrados o no. Pero para ser cubierto en la programación visual, algunas representaciones gráficas significativas deben ser usadas en el proceso de programación.

3.1. Lenguajes de programación visual

Definición Un lenguaje de programación visual puede ser informalmente definido como un lenguaje en el cual usa algunas representaciones visuales (agregando o en lugar de palabras o números) para realizar los que de otra manera tendría que ser escrito en un lenguaje de programación tradicional en una dimensión [37].

Note que esta definición no impone restricciones sobre el tipo de datos o información. No tiene importancia si los objetos que son operados o mostrados al usuario por un lenguaje visual que es textual, numérico, pictórico, o incluso audio. Lo que es importante es que, para ser considerado un lenguaje de programación visual, el lenguaje mismo debe emplear algunas expresiones visuales significativas como un medio para la programación.

En [38] un lenguaje visual se define como un sistema **(DI,DO,DBAS)**, donde:

DI: representa un conjunto de íconos que es la base de datos de los íconos elementales disponibles en el sistema.

DO: representa un conjunto de operadores genéricos que se aplican a los íconos para crear íconos complejos, también llamadas **proposiciones visuales**; y

BDAS: contiene la interpretación semántica que está asociada a los elementos del conjunto **DI**

3.1.1. Diccionario de Íconos

Los elementos de este conjunto tienen la estructura de un ícono generalizado, el cual es definido por dos partes una semántica y otra física.

Sea $\mathbf{X}$ un ícono generalizado, y $(\mathbf{X_l}, \mathbf{X_f})$ su representación donde:

$\mathbf{X_l}$: es la parte semántica del ícono o su significado, y

$\mathbf{X_f}$: es la parte física o su representación gráfica.

Los íconos generalizados se clasifican en dos tipo, ver figura 3.1:

Íconos primitivos o elementales. Son de la forma $\mathbf{X_l}, \mathbf{X_f}$ y representan a los íconos base de un sistema. Según sean los valores de $\mathbf{X_l}$ y $\mathbf{X_f}$ se obtiene la siguiente clasificación, en la siguiente tabla **a** denota un significado, **b** una representación gráfica y **&** el nulo.

X_l	X_f	Tipo de ícono
&	**&**	Vacío
a	**&**	Significado
&	**b**	Imagen
a	**b**	Completo

Íconos Complejos. Son construidos a partir de un grupo de íconos elementales de la forma $\mathbf{X_{li}}, \mathbf{X_{fi}}$ dando como resultado un ícono con la siguiente estructura:

$$(\{OP, X_{l1}, X_{l2}, ..., X_{ln})\}, \{OP', X_{f1}, X_{f2}, ..., X_{fn}\})$$

donde: OP es un operador que actúa sobre las partes semánticas de los íconos para obtener el significado del ícono complejo.

OP' actúa sobre las partes físicas de los íconos obteniendo la representación gráfica del ícono complejo.

Estos íconos complejos se componen de los estructurales que son en los que se sabe los íconos elementales que interviene para su construcción pero no la forma en que deben combinarse; y los compuestos en los cuales se incluye la forma en que estos íconos se construyen.

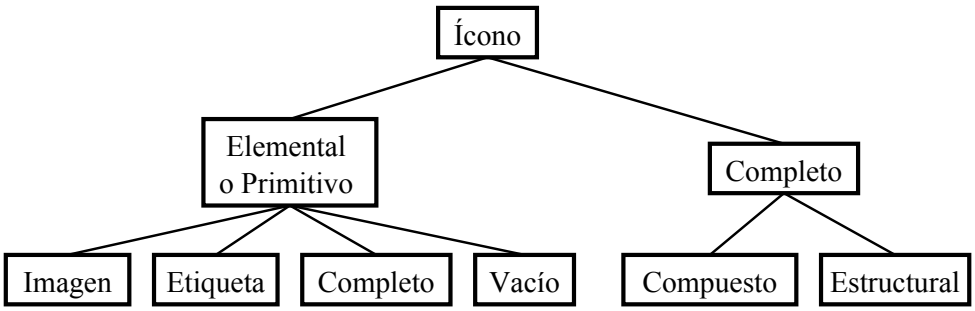


Figura 3.1: Clasificación de los íconos.

Estructura interna del DI. La parte semántica de un ícono (su significado), representa la acción a realizar o la interpretación que se le dará al ícono dentro de uno complejo. Una de las técnicas para especificar esta parte es usar acciones semánticas: son procesos que pueden ser activados en algún momento, por ejemplo cuando el ícono al que se le asocia sea seleccionado por el usuario o por el sistema.

Representación gráfica de un ícono. La imagen es un símbolo gráfico. Esta representación gráficas está formada por diferentes patrones primitivos, los cuales son generados o contruidos por diferentes técnicas, algunas de estas son:

- Usando un mapa de píxeles
- Vectorial incluyendo atributos de color
- Usando una gramática básica

La primera alternativa, **usando un mapa de píxeles**, utiliza para el diseño un editor de íconos, con opción para almacenamiento en disco. Esta imagen es recuperada y mostrada cuando el sistema lo requiera. La siguiente figura muestra una imagen y a un lado su representación interna

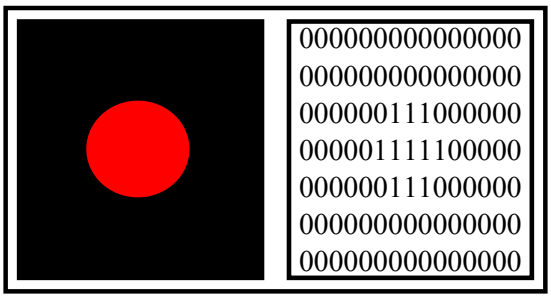


Figura 3.2: Técnica usando mapa de píxeles.

La **representación vectorial** nos permite representar a casa ícono mediante un conjunto de vectores. La siguiente figura muestra una imagen obtenida usando este técnica y a un lado su representación interna.

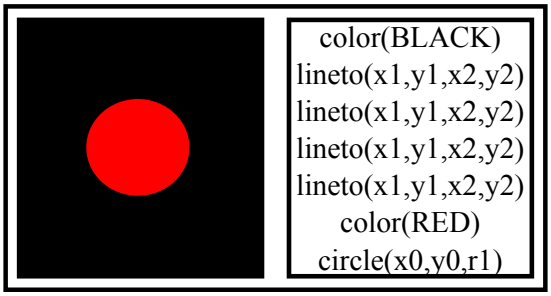


Figura 3.3: Técnica basada en vectores incluyendo atributos de color.

La técnica usando una **gramática gráfica** es la más refinada. Utiliza una gramática (G_0) cuyas producciones utilizan como elementos algunos trazos o figuras geométricas y como operadores ciertas operaciones dentro de un espacio euclidiano. Se define de la

siguiente manera:

(G_0) es una gramática libre de contexto (N, T, S, P) , donde

N :	Símbolos no terminales
$T = T_l \bigcup T_{sp}$:	Símbolos terminales
$T_l = \{i = (i_l, i_f)\}$	
$T_{sp} = \{\&, \hat{}, +\}$	
$\{\&, \hat{}, +\}$	Representan las operaciones de concatenación vertical, horizontal y superposición
S:	Símbolo inicial
P:	Reglas de producción de G_0

Generalmente los íconos no terminales en G_0, a excepción del símbolo inicial, representan objetos icónicos compuestos, es decir, pueden ser derivados en uno o más pasos iniciando desde un no terminal N dado, el cual es diferente de S. En otras palabras los íconos complejos son obtenidos por acomodos espaciales de los íconos elementos.

La figura 3.4 la representación interna de un ícono usando esta técnica. En ella se ve un ícono complejo y a un lado se representa la conjunción de dos elementos usando la operación de superposición (+).

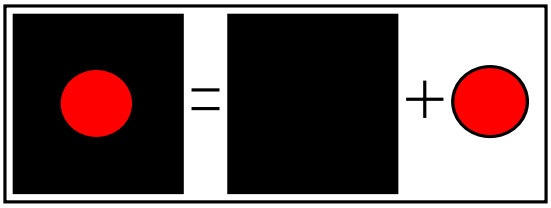


Figura 3.4: Ejemplo usando una gramática G_0.

Todos los íconos involucrados en un sistema (elementales, complejos, proposiciones visuales) son objetos con la representación dual de una semántica y una física. Esto se verifica inmediatamente para los íconos elementales, partiendo de la estructura del DI. Para los complejos y proposiciones visuales, la parte física y la parte semántica se derivan de los íconos elementales que los integran.

3.1.2. Diccionario de Operadores

Una proposición visual, que consiste de un arreglo espacial de íconos, es considerada la contraparte bidimensional de la secuenciación estándar que se tienen en los lenguajes de programación tradicionales. Mientras en los lenguajes basados en cadenas, la única regla de construcción es la concatenación de las cadenas de texto. En el caso de los lenguajes visuales se tienen por lo menos tres reglas de construcción para ser usadas en el arreglo espacial de los íconos, que corresponden a los siguiente operadores espaciales:

& : concatenación horizontal
^ : concatenación vertical
+ superposición espacial

Por lo que se recomienda incluir tanto operadores espaciales e íconos elementales en el vocabulario de símbolos terminales de la gramática del lenguaje visual, lo que no es necesario hacer en una gramática basada en cadenas.

3.1.3. Base de datos de Acciones Semánticas

Una vez definido un ícono complejo, es necesario contar con los elementos necesarios que nos permitan interpretarlo correctamente. Estos elementos integran precisamente a la BDAS.

Recordemos que cada ícono en el sistema tiene información de cómo debe interpretarse, cualquier elemento icónico formado por un conjunto de íconos elementales también cuenta con esta información, la cual es obtenida de los íconos que lo están formando. La manera en que debe combinarse es especificada por este componente del lenguaje visual.

La interpretación la realiza una parte específica del sistema, y puede regresar como resultado un único dato, o en su defecto, puede ir generando resultados intermedios dependiendo del contexto del trabajo y del significado de cada ícono en el lenguaje.

La figura 3.5 trata de explicar el papel que juega la BDAS dentro de un sistema que use un lenguaje visual.

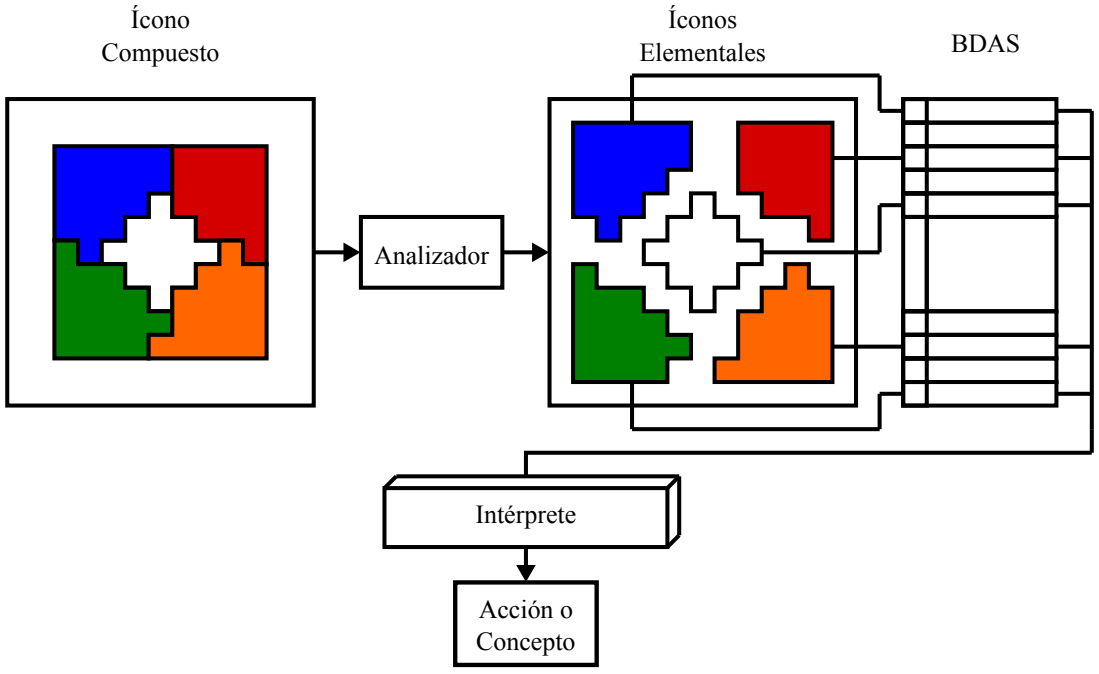


Figura 3.5: Estructura interna del funcionamiento de una DBAS.

Cualquier sistema que posea las tres partes que se describieron anteriormente: el Dic-

cionario de Íconos (DI), un Diccionario de Operadores (DO) y una Base de Datos de Acciones Semánticas (BDAS), es considerado formalmente como un lenguaje visual.

3.2. Aspectos importantes de los lenguajes de programación

Cuando intentamos evaluar un lenguaje de programación, dos aspectos vienen a la mente, el nivel del lenguaje y alcance del lenguaje.

Generalmente se ha acordado que el nivel del lenguaje es una medida inversa a la cantidad de detalles que el usuario tenga que dar a la computadora para alcanzar los resultados deseados. Un lenguaje no es procedural (y en el más alto nivel) si los usuarios le dicen a la computadora sólo lo que va a ser realizado, pero no cómo hacerlo. Un lenguaje es procedural si los usuarios necesitan especificar los pasos que la computadora debe seguir. El número y el tamaño de los pasos requeridos varían en los lenguajes de programación procedurales. Para alcanzar el mismo resultado un lenguaje altamente procedural (nivel bajo, por ejemplo, un lenguaje ensamblador) requiere muchos pasos pequeños pero detallados, mientras que un lenguaje menos procedural (alto nivel, por ejemplo FORTRAN) requiere menos pero pasos más grandes con menos detalles por parte del usuario. Por esta medida, FORTRAN está en un nivel más alto que un lenguaje ensamblador.

El alcance del lenguaje, va de lo general y extensamente aplicable a lo específico y reducidamente aplicable, representa cuánto un lenguaje es capaz de hacer. Usando FORTRAN y un lenguaje ensamblador como ejemplos otra vez, el usuario podría utilizar FORTRAN otra vez para realizar cálculos científicos complejos, pero probablemente no lo usaría para manejar operaciones multitarea. Un lenguaje ensamblador, por otro lado, puede ser usado generalmente para ambos. Por lo tanto, decimos que el lenguaje ensamblador tiene un dominio del problema más grande o más extenso de aplicabilidad que FORTRAN.

Por supuesto, hay otras maneras de clasificar o caracterizar un lenguaje. Sin embargo, para los procesos más prácticos, estos aspectos son considerados dos de las dimensiones más fundamentales en la evaluación de los lenguajes de programación. Son aplicables para los lenguajes de programación en general, a pesar de que sea visual o no.

3.3. Las tres dimensiones de los lenguajes de programación visual

Para poner los lenguajes de programación en perspectiva, se introduce una dimensión adicional: grado de expresión visual.

Las expresiones visuales son representaciones visuales (no textuales) significativas por ejemplo, íconos, gráficas, diagramas, imágenes) usadas como componentes del lenguaje para alcanzar el propósito de programar. EL grado de la expresión visual es una medida relativas de qué tanto las expresiones visuales son incorporadas en el lenguaje de programación. Mientras las representaciones sean más visuales es mayor el grado visual. Si

no hay expresiones visuales en el lenguaje (aunque la información que sea procesada o mostrada tenga imágenes), la tercera dimensión simplemente no aplica.

Análisis dimensional de los lenguajes de programación visual. Un enfoque analítico aunque cuantitativo es propuesto para la comparación de los lenguajes de programación visual. En esencia, esto involucra la construcción del perfil de un lenguaje, que caracteriza al lenguaje en un marco de trabajo de tres dimensiones. Gráficamente, podría ser representado mediante las medidas relativas del lenguaje en los tres ejes etiquetados como "nivel del lenguaje", "alcance" y el "grado visual" como se muestra en la figura 3.6.

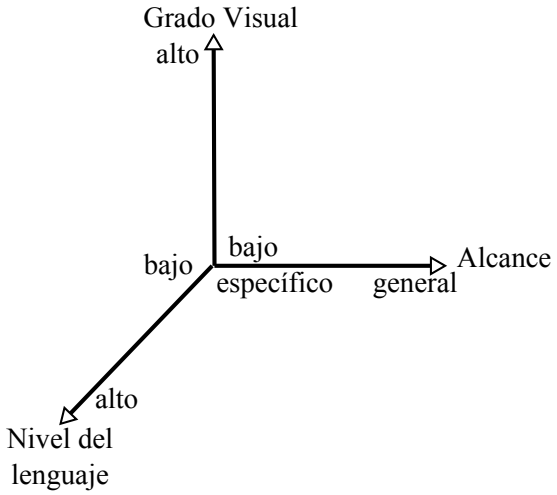


Figura 3.6: Tres dimensiones de la programación visual.

Basado en los principios del diseño, muchos de los lenguajes de programación descritos en la literatura caen dentro de estas tres categorías generales:

En un extremo, los íconos son diseñados deliberadamente para jugar el rol central en la programación

En el otro extremo, los diagramas de flujo que ya están en uso en el papel están también incorporados en las construcciones de la programación, como extensiones para los lenguajes de programación visuales, o creadas como unidades interpretables por las máquinas para que sean usadas en conjunción con los lenguajes de programación convencionales.

En el centro, las representaciones gráficas están diseñadas como una parte integral del lenguaje. Sin embargo, a diferencia de los íconos de los sistemas icónicos, ellos no son las "superestrellas" del lenguaje; y a diferencia las extensiones gráficas, el lenguaje no está diseñado para que sea usado conjuntamente con algunos otros lenguajes y que no pueden funcionar sin las representaciones gráficas.

3.4. Visualización de Datos

La visualización de datos es la representación gráfica de la información abstracta para dos propósitos [39]: el sentido de decisiones (también llamado análisis de datos) y la comunicación. Historias importantes viven en los datos y la visualización de datos es un poderoso medio para descubrir y comprender estas historias, y luego presentarlas a los demás. La información es abstracta porque describe cosas que no son físicas. La información estadística es abstracta. Tanto si se trata de ventas, la incidencia de la enfermedad, el rendimiento deportivo, o cualquier otra cosa, a pesar de que no pertenezca al mundo físico, podemos mostrarla visualmente, pero para ello se tiene que encontrar una manera de dar forma a lo que no tiene ninguna. Esta traducción de lo abstracto en los atributos físicos de la visión (longitud, posición, tamaño, forma y color, por nombrar algunos) sólo puede tener éxito si entendemos un poco acerca de la percepción visual y la cognición. En otras palabras, para visualizar datos de manera efectiva, se deben seguir los principios de diseño que derivan de un entendimiento de la percepción humana.

Como dice el refrán, "una imagen vale más que mil palabras" - a menudo más - pero sólo cuando la historia es mejor contada gráficamente en lugar de verbalmente y la imagen está bien diseñada. Uno podría mirar a una tabla de números todo el día y no ver lo que sería inmediatamente evidente cuando se mira en una buena imagen de los mismos números. Para ilustrar, en la figura 3.7 hay una tabla simple de los datos de ventas - un año de provecho - dividida en dos regiones:

2009 Sales (thousands of U.S. $)

Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Domestic	1,983	2,343	2,593	2,283	2,574	2,838	2,382	2,634	2,938	2,739	2,983	3,493	31,783
International	574	636	673	593	644	679	593	139	599	583	602	690	7,005
Total	2,557	2,979	3,266	2,876	3,218	3,517	2,975	2,773	3,537	3,322	3,585	4,183	38,788

Figura 3.7: Tabla de valores de ventas divididas en dos regiones.

Esta tabla hace dos cosas extremadamente bien: expresa estos valores de venta precisamente y proporciona un medio eficaz para buscar valores para una región y mes en particular. Pero si se está en busca de patrones, tendencias y excepciones entre estos valores, si se quiere una idea rápida de la historia contenida en estas cifras, o se tiene que comparar conjuntos enteros de números en lugar de sólo dos a la vez, esta tabla falla.

Ahora en la figura 3.8 muestra la misma información en forma de una gráfica de líneas:
Varios hechos ahora saltan a la vista cuando se observa esta imagen:

- Las ventas nacionales fueron considerablemente y consistentemente superiores que las internacionales.
- Las ventas nacionales mostraron una tendencia al alza durante el año en su totalidad.
- Las ventas internacionales, en cambio, se mantuvieron relativamente estables, con una notable excepción: se redujeron fuertemente en agosto.

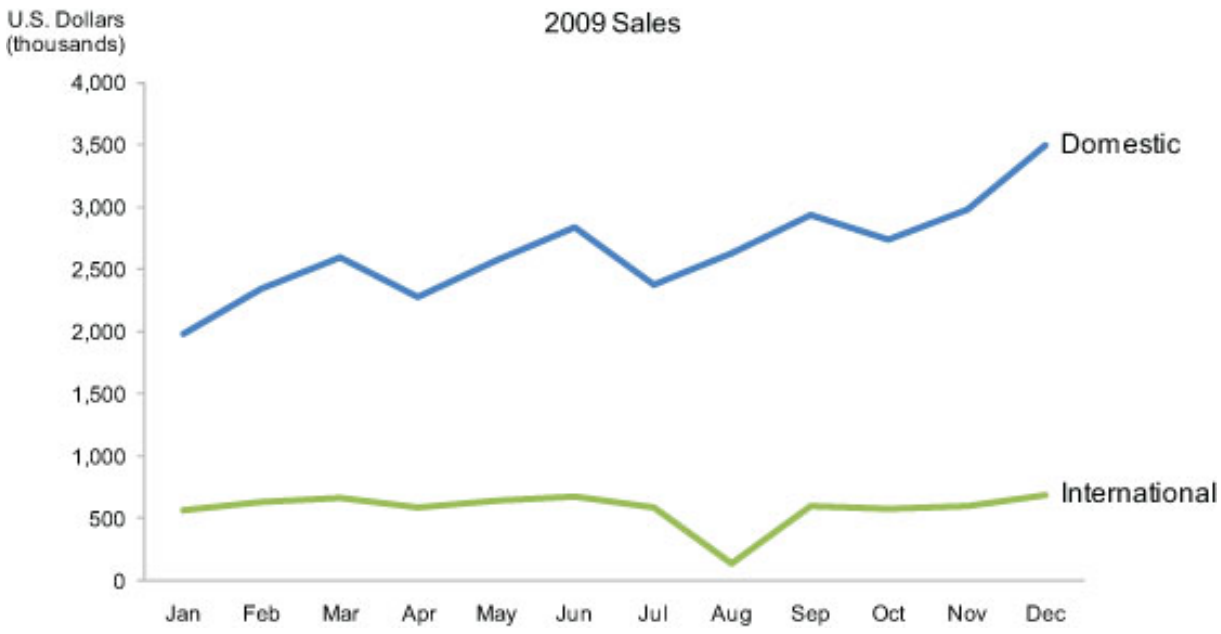


Figura 3.8: Tabla de valores de ventas divididas en dos regiones.

- Las ventas nacionales mostraron un patrón cíclico - arriba, arriba, abajo - que se repitió en forma trimestral, siempre llegando a la cima en el último mes del trimestre y luego disminuye drásticamente en el primer mes del siguiente.

Lo que estos números no se pudieron comunicar cuando se presenta en forma de texto en una tabla, que nuestros cerebros interpreten a través del uso del procesamiento verbal, se hace visible y comprensible cuando se comunican visualmente. Este es el poder de la "visualización de datos".

Aunque la visualización de datos por lo general cuenta las relaciones entre los valores cuantitativos, también puede mostrar las relaciones que no son de naturaleza cuantitativa. Por ejemplo, las conexiones entre las personas en un sitio de redes sociales como Facebook o entre los sospechosos de terrorismo pueden visualizarse utilizando una visualización de un nodo y su enlace. En el siguiente ejemplo, las personas son los nodos, representados como círculos, y sus relaciones son los enlaces, representadas como líneas que las conectan.

Las visualizaciones que cuentan con relaciones entre entidades, como la gente en el ejemplo anterior, se pueden enriquecer con la adición de información cuantitativa también. Por ejemplo, el número de veces que dos personas han interaccionado podría ser representado por el grosor de la línea que los conecta.

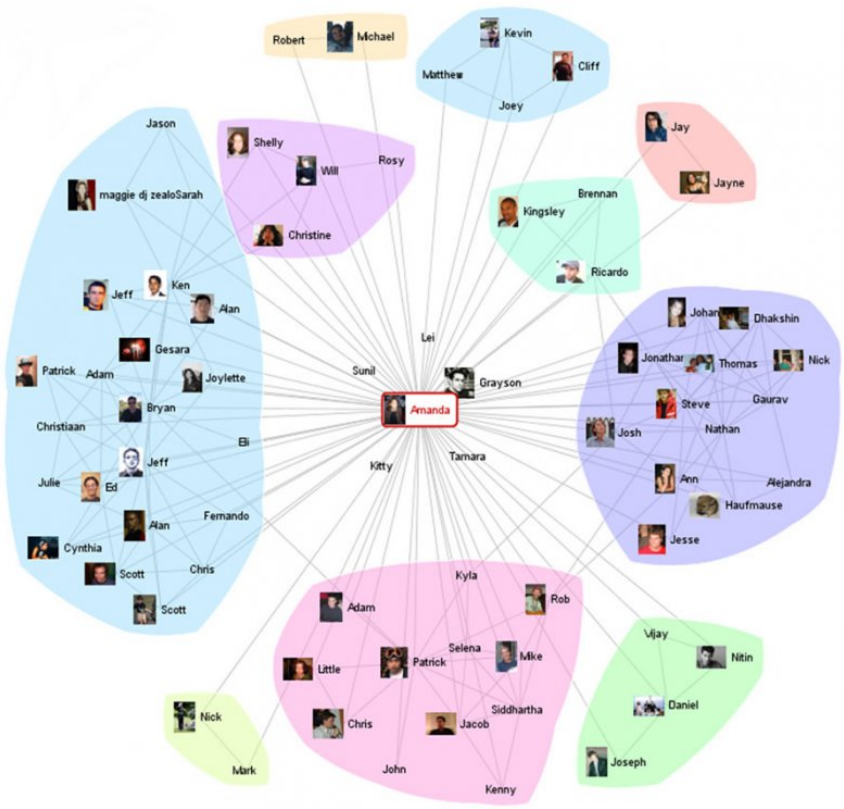


Figura 3.9: Gráfica que muestra una red social en la cual los nodos son las personas y las líneas que los conecta la relación que hay entre ellas..

3.5. Lenguaje de marcación de hipertexto

3.5.1. Lenguajes de etiquetas

Uno de los retos iniciales a los que se tuvo que enfrentar la informática fue el de cómo almacenar la información en los archivos digitales. Como los primeros archivos sólo contenían texto sin formato, la solución utilizada era muy sencilla: se codificaban las letras del alfabeto y se transformaban en números.

De esta forma, para almacenar un contenido de texto en un archivo electrónico, se utiliza una tabla de conversión que transforma cada carácter en un número. Una vez almacenada la secuencia de números, el contenido del archivo se puede recuperar realizando el proceso inverso, ver figura 3.10.

El proceso de transformación se denomina **codificación de caracteres** y cada una de las tablas que se han definido para realizar la transformación se conocen con el nombre de **páginas de código**. Una de las codificaciones más conocidas (una de las primeras) es la ASCII.

Una vez resuelto el problema de almacenar el texto simple, se presenta el reto de almacenar los contenidos de texto con formato. En otras palabras, ¿cómo se almacena un texto en negrita? ¿y un texto de color rojo? ¿y otro texto azul, en negrita y subrayado?

Utilizar una tabla de conversión similar no es posible, ya que existen infinitos estilos posibles para aplicar al texto. Una solución técnicamente viable consiste en almacenar la información sobre el formato del texto en una zona especial reservada dentro del propio archivo indicando donde comienza u dónde termina este formato.

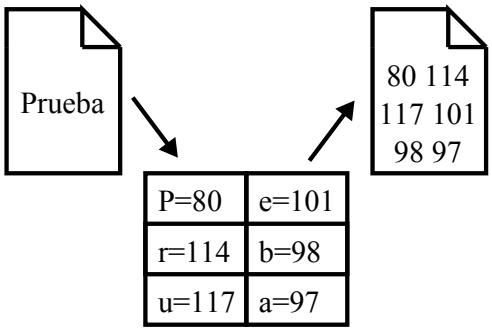


Figura 3.10: Ejemplo sencillo de codificación de caracteres.

No obstante, la solución que realmente se emplea para guardar la información con formato es mucho más sencilla: el archivo electrónico almacena tanto los contenidos como la información sobre el formato de esos contenidos. Si por ejemplo se quiere dividir el texto en párrafos y se desea dar especial importancia a algunas palabras, se podría indicar de manera muy descriptiva de la siguiente manera:

<parrafo> Contenido de texto con <importante>algunas palabras</importante> resaltadas de forma especial. </parrafo>

En general las etiquetas se indican por pares y se forman de la siguiente manera, una etiqueta de apertura: <, seguido del nombre de la etiqueta (sin espacios en blanco) y >; y una etiqueta de cierre: <, seguido de "/", seguido del nombre de la etiqueta y >.

Así, la estructura típica de las etiquetas HTML es:

<nombre_etiqueta atributos=valor> ... </nombre_etiqueta>

La principal ventaja es que son muy sencillos de leer y escribir por parte de las personas y de los sistemas electrónicos. La principal desventaja es que pueden aumentar mucho el tamaño del documento, por lo que en general se utilizan etiquetas con nombres muy cortos.

3.5.2. ¿Qué es HTML?

Definiéndolo de forma sencilla, "HTML es lo que se utiliza para crear todas las páginas web de Internet". Aunque HTML es un lenguaje que utilizan los ordenadores y los programas de diseño, es muy fácil de aprender y escribir por parte de las personas, HTML son las siglas de HyperText Markup Language.

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado World Wide Web Consortium (W3C) [40]. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de forma muy similar en cualquier navegador de cualquier sistema operativo.

El propio W3C define el lenguaje HTML como "un lenguaje reconocido universalmente y que permite publicar información de forma global". Desde su creación, el lenguaje HTML ha pasado de ser un lenguaje utilizado exclusivamente para crear documentos electrónicos a ser un lenguaje que se utiliza en muchas aplicaciones electrónicas como buscadores, tiendas online y banca electrónica.

3.5.3. Evolución de HTML

El origen de HTML se remonta a 1980, cuando el físico Tim Berners-Lee, trabajador del CERN [41] (Organización Europea para la Investigación Nuclear) propuso un nuevo sistema de "hipertexto" para compartir documentos. El "hipertexto" permitía que los usuarios accedieran a la información relacionada con los documentos electrónicos que estaban visualizando. De cierta manera, los primitivos sistemas de "hipertexto" podrían asimilarse a los enlaces de las páginas web actuales.

Tras finalizar el desarrollo de su sistema de "hipertexto", Tim Berners-Lee lo presentó a una convocatoria organizada para desarrollar un sistema de "hipertexto" para Internet. Después de unir sus fuerzas con el ingeniero de sistemas Robert Cailliau, presentaron la propuesta ganadora llamada World Wide Web (W3).

El primer documento formal con la descripción de HTML se publicó en 1991 bajo el nombre "HTML Tags" [42] y todavía hoy puede ser consultado online a modo de reliquia informática.

La primera propuesta oficial para convertir HTML en un estándar se realizó en 1993 por parte del organismo IETF [43] (Internet Engineering Task Force). Aunque se consiguieron avances significativos (se definieron las etiquetas para imágenes, tablas y formularios) ninguna de las dos propuestas de estándar, HTML y HTML+, consiguieron convertirse en estándar oficial.

En 1995, el organismo IETF organiza un grupo de trabajo de HTML y consigue publicar, el 22 de septiembre de ese mismo año, el estándar HTML 2.0, que fue el primer estándar oficial de HTML.

A partir de 1996, los estándares de HTML los publica otro organismo de estandarización llamado W3C [40] (World Wide Web Consortium). La versión HTML 3.2 se publicó el 14 de enero de 1997 y es la primera recomendación de HTML publicada por el W3C, incorpora los últimos avances de las páginas web desarrolladas hasta 1996, como applets

de Java y texto que fluye alrededor de las imágenes.

HTML 4.0 se publicó el 24 de Abril de 1998 (versión corregida de la publicación del 18 de diciembre de 1997) y supone un gran salto desde las versiones anteriores. Entre sus novedades más destacadas se encuentran las hojas de estilos CSS, la posibilidad de incluir pequeños programas o scripts en las páginas web, mejora de la accesibilidad de las páginas diseñadas, tablas complejas y mejoras en los formularios.

La última especificación oficial de HTML se publicó el 24 de diciembre de 1999 denominada HTML 4.01. Se trata de una revisión y actualización de la versión HTML 4.0, por lo que no incluye novedades significativas.

Desde la publicación de HTML 4.01, la actividad de estandarización de HTML se detuvo y el W3C se centró en el desarrollo del estándar XHTML. Por este motivo, en el año 2004, las empresas Apple, Mozilla y Opera mostraron su preocupación por la falta de interés del W3C en HTML y decidieron organizarse en una nueva asociación llamada WHATWG [44] (Web Hypertext Application Technology Working Group).

La actividad actual del WHATWG se centra en el futuro estándar HTML 5, cuyo primer borrador oficial [45] se publicó el 22 de enero de 2008. Debido a la fuerza de las empresas que forman el grupo WHATWG y a la publicación de los borradores de HTML 5.0, en marzo de 2007 el W3C decidió retomar la actividad estandarizadora de HTML [46].

De forma paralela a su actividad con HTML, W3C ha continuado con la estandarización de XHTML, una versión avanzada de HTML y basada en XML, cuya primera versión se denomina XHTML 1.0 y se publicó el 26 de enero de 2000 (revisada después el 1 de agosto de 2002).

Especificación estándar

El organismo W3C elabora las normas que deben seguir los diseñadores de páginas web para crear las páginas HTML. Las normas estándares están escritas en inglés y se pueden consultar de forma gratuita:

- Especificación estándar de HTML 4.01 [47]
- Especificación estándar de XHTML 1.0 [48]

El estándar XHTML 1.0 incluye el 95 % del estándar HTML 4.01, ya que sólo añade pequeñas mejoras y modificaciones menores. Afortunadamente, no es necesario leer las especificaciones y recomendaciones estándares de HTML para aprender a diseñar páginas con HTML o XHTML. Las normas estándares están escritas con un lenguaje bastante formal y algunas secciones son difíciles de comprender.

HTML y XHTML

Las diferencias más importantes con HTML [49] son las siguientes:

- Estructura del documento
  - El DOCTYPE de XHTML es obligatorio.
  - El atributo de espacio de nombres de XML en <html> es obligatorio.
  - Las etiquetas <head>, <html>, <title>, y <body> son obligatorias.
- Elementos XHTML
  - Deben estar correctamente anidados.
  - Deben ser siempre cerrados.
  - Deben estar en minúsculas.
  - Deben tener un elemento raíz.
- Atributos XHTML
  - Los nombres de los atributos deben estar en minúsculas.
  - Los valores de atributo debe ser encerrados entre comillas.
  - La minimización de atributos está prohibido.

HTML y CSS

Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas, por lo que es conveniente separarlos. CSS es el mecanismo que permite separar los contenidos definidos mediante HTML y el aspecto que deben presentar esos contenidos:

Figura 3.11: Esquema de la separación de los contenidos y su presentación.

Una ventaja añadida de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles, ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de dispositivos móviles, impresoras y dispositivos utilizados por personas discapacitadas.

De esta forma, utilizando exclusivamente XHTML se crean páginas web "feas" pero correctas. Aplicando CSS, se pueden crear páginas "bonitas" a partir de las páginas XHTML correctas.

3.5.4. HTML 5

La especificación de html5 no se adscribe a una sintaxis o a la otra, sino que admite ambas: html y xhtml [50]. De esta manera, los creadores de contenido pueden escoger entre un enfoque práctico aunque poco riguroso (sintaxis html) y una visión académica y estricta (sintaxis xhtml). Con el tiempo, el W3C ha acabado aceptando que html y xhtml sean recomendaciones paralelas que pueden coexistir.

Html5 no sólo define cómo se deben analizar los documentos, sino también cómo se deben interpretar si no son válidos o si están mal formados. Actualmente los navegadores corrigen los errores de sintaxis de distinta manera, de modo en que a los fabricantes les resulta más práctico. Html5 trata de poner fin a esa necesidad de ingeniería inversa de los navegadores, que compiten por definir cómo se deben corregir los errores.

DOM

Una de las novedades principales es la inclusión del *document object model* (DOM) como fundamento del lenguaje. DOM describe la estructura de un documento de acuerdo con el paradigma de la orientación a objetos, define el conjunto de entidades que están en un documento y las acciones que pueden realizarse sobre ellas. El DOM se había tratado de forma separada, pero ahora forma parte del estándar, así se garantiza que los navegadores interpretarán adecuadamente la sintaxis de html y que al mismo tiempo implementarán las funciones del DOM.

Presentación y recuperación de información

Html5 incluye elementos nuevos destinados a enriquecer la presentación de documentos. Son ejemplos de ello los elementos semánticos *article*, *header*, *hgroup*, *nav*, *section*, *aside* y $footer$. Los blogs y los sitios de noticias han influido en esta evolución. Html5 define más de una docena de nuevos controles (email, range, date, time, placeholder, autofocus, color, etc.) en los formularios para enviar datos al servidor que actúan sin necesidad de utilizar JavaScript que a veces está inhabilitado, por tanto que sean los navegadores quienes faciliten la entrada y la validación de datos.

Flash y contenido multimedia

Para incrustar contenido multimedia, html ya contaba con el elemento *object*, pero ahora incorpora directivas nuevas que actúan como contenedores de vídeo, gráficos vectoriales y audio que hacen más eficiente su reproducción, sin requerir componentes externos como Flash. Todo apunta a que los navegadores añadirán estas capacidades y que muchas páginas web se modificarán para aprovecharlas. Todavía no se cuenta con un formato de vídeo estándar, no se sustenta el visionado a pantalla completa ni se permite escoger la calidad de reproducción.

Capítulo 4

Lenguaje Intermedio REC y el Proyecto R

La siguiente figura 4.1 muestra la arquitectura de la aplicación, la aplicación consta de 3 módulos principales, la vista de la aplicación, en la cual se genera el código intermedio REC una vez que el programa visual esté contruido; el servidor de la aplicación, en el cual está el compilador de REC para generar el código en R; y por último la interfaz, la cual se encarga de enviar este código R para que sea ejecutado. Una vez que el codigo R es ejcutado em módulo de interfaz con R devuelve los resultados al servidor para que este a su vez los envíe a la vista para que sean mostrados al usuario.

Este capítulo describe al lenguaje REC, que significa Regular Expression Compiler, su sintaxis y pequeños ejemplos de cómo se comporta el flujo dependiendo de la evaluación de las expresiones. Además, se describe al proyecto R que es un entorno y un lenguaje de programación orientado al análisis y visualización de datos, el cual es la base para el desarrollo de esta tesis.

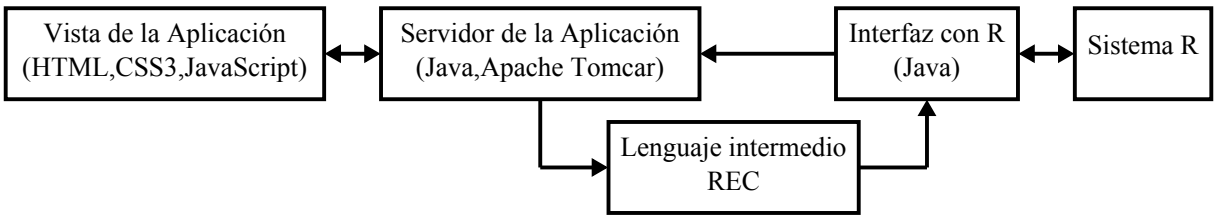


Figura 4.1: Arquitectura de la aplicación indicando los módulos que la forman.

4.1. Lenguaje Intermedio REC

La sintaxis de REC en BNF extendido es la siguiente [9, 10]:

$Prog : := Expr$ | " { " { $Prog\ char$ } $Prog$ "}"
$Expr : :=$ "("{ {$Prog$ | $pred$} (" : " | " ; ")}{ $Prog$ | $pred$} ")"

donde Prog y Expr representan una expresión o un programa de REC, respectivamente, char representa algún carácter ASCII imprimible excepto la llave de cierre ("}"), y pred representa algún predicado (usualmente uno o más caracteres, pero en algunos casos un lexema más complicado, por ejemplo, una cadena que esté entre comillas).

Un programa es una lista de subrutinas (cada uno de ellas un programa seguido por su nombre) y un programa principal. El predicado @a está reservado para la llamada a la subrutina la cual su nombre es a. Las llamadas a subrutinas son hechas mediante una tabla y nombres que son ligadas dinámicamente: las definiciones dadas dentro de una lista encerrada entre llaves son activadas cuando el correspondiente programa principal empieza su ejecución y las previas definiciones para los mismos nombres son almacenados para la recuperación cuando el programa principal termine. Una expresión de REC es una lista encerrada entre paréntesis de cadenas de predicados o programas separados por dos puntos o punto y coma; puede haber una cadena final terminada por el paréntesis derecho. Una expresión de REC es ejecutada de izquierda a derecha, con el flujo de control siendo alterado en 4 casos:

- Un signo de dos puntos causa que salte al inicio de la expresión, ver figura 4.2
- Un punto y coma causa que la ejecución continúe a la derecha de la expresión (la expresión termina en true), ver figura 4.3
- La expresión de un predicado, además de su operación de asignación, resulta en un valor de verdad. Si es true, el flujo del control no es alterado; si es $false$, la ejecución de la cadena en el cual aparece termina, y continúa a la derecha de la cadena terminadora (dos puntos, punto y coma o paréntesis derecho). Ciertos predicados son siempre *true*, entonces son distinguidos siendo llamados "operadores".
- La llegada de un paréntesis derecho causa que la expresión se convierta en false causando por ello un salto al siguiente segmento en la expresión que encierra, como con un predicado false. Si no hay un expresión que encierre, pero la expresión que termina es una subrutina, o el programa principal de una subrutina, el predicado correspondiente que lo llamó se convierte en false.

Aunque la notación es mínima, las combinaciones booleanas más complicadas de predicados pueden ser realizadas a través de la configuración del flujo control por los elementos sintácticos que hemos mencionado. Si π_1 y π_2 son dos predicados, $(\pi_1; \pi_2;)$ es su or lógico, $(\pi_1; \pi_2;)$ es su *and*. $(\pi_1(\pi_2); \pi_2(\pi_1);)$ es el *or* exclusivo pero, como ya se comentó, puede

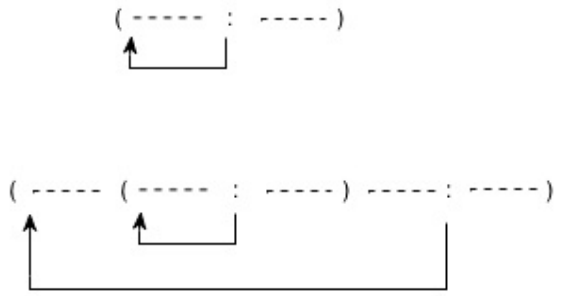


Figura 4.2: Comportamiento de los 2 puntos

Figura 4.3: Comportamiendo del punto y coma

involucrar hacer π_1 y π_2 dos veces. Debido a que la evaluación de una serie de predicados es secuencial, de izquierda a derecha, no muchas de ellas serán ejecutadas ya que no son necesarias para alcanzar una decisión. Consecuentemente las operaciones booleanas no son conmutativos realmente, y se debe tener cuidado con su orden. En la práctica esta situación es más una ventaja que una desventaja. También conforma la importante innovación que McCarthy incorpora en LISP, las correspondientes alternativas para las ramas no usadas nunca son evaluadas sólo para ser descartadas. Esta distinción es importante porque las cantidades en la peor secuencia de flujo son frecuentemente indefinidas.

Las construcciones de programación estructurada son también fácilmente expresadas en REC: si π es una condición, por ejemplo un predicado, y ω,ω_1 y ω_2 son operadores (predicados los cuales el valor de verdad es true), entonces
$(\pi\omega_1;\omega_2;)$ es si π entonces ω_1 si no ω_2
$(\pi\omega :;)$ es mientras π haz ω
$(\pi\omega :;)$ es repite ω hasta π
La primera expresión puede ser claramente extendida para cubrir la construcción if-else:

$(\pi_1\omega_1;\pi_2\omega_2;\pi_3\omega_3;\omega_4;)$ es si π_1 entonces ω_1
o si π_2 entonces ω_2
o si π_3 entonces ω_3
si no ω_4.

Pocos programas pueden subsistir sin subrutinas, y REC no es la excepción; en efecto, esto es una parte de lo que le da el poder computacional a REC más allá expresiones regulares planas. El mecanismo de la definición de subrutinas es para encerrar una lista de las subrutinas deseadas y sus nombres, junto con los programas el cual está para utilizarlas, en una parte de las llaves. La combinación podría mostrarse de la forma $\{(...)a(...)b...(...)n(...)\}$, en la cual el primer paréntesis (el cual por sí mismo podría estar encerrado entre llaves debido al anidamiento de subrutinas adicional) es la definición de una subrutina a, y así sucesivamente. Se acuerda que todas las definiciones de las subrutinas serán validadas dentro del programa principal y en los demás, pero solamente dentro de llaves; sin embargo, pueden ser sustituidas dinámicamente por una definición nueva de una subrutina en un nivel más profundo de llaves. También está entendido que la construcción entre llaves tendrá el valor de verdad de su expresión principal (final), tal como una expresión entre paréntesis es un predicado el cual el valor de verdad depende de si su ejecución terminó con el punto y coma, su paréntesis derecho, o la falsedad de un predicado final.

4.2. El software R

R es un entorno especialmente diseñado para el tratamiento de datos, cálculo y desarrollo gráfico. Permite trabajar con facilidad con vectores y matrices y ofrece diversas herramientas para el análisis de datos. El lenguaje de programación R forma parte del proyecto GNU y puede verse como una implementación alternativa del lenguaje S, desarrollado en AT&T Bell Laboratories. Se presenta como software libre, donde el término software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Se trata de un lenguaje creado específicamente para la visualización y exploración de datos así como para su uso en modelización y programación estadística.

En la página web [51] se encuentra disponible toda la información acerca de R. La instalación de R se realiza a través de la CRAN (Comprehensive R Archive Network). Además, R es un entorno en el que se han ido implementando diversas técnicas estadísticas. Algunas de ellas se encuentran en la base de R pero otras muchas están disponibles como paquetes (packages). Estos paquetes están disponibles en la web http://cran.au.r-project.org/. En resumen, R proporciona un entorno de trabajo especialmente preparado para el análisis estadístico de datos. Sus principales características son las siguientes:

- R proporciona un lenguaje de programación propio. Basado en el lenguaje S, que a su vez tiene muchos elementos del lenguaje C. Sin embargo, la semántica es muy distinta a la de este último. Esto es porque R permite ejecuciones de comandos

en línea (compilación y ejecución unidos en un mismo paso), lo cual hace que su semántica esté más próxima a la de un lenguaje de programación funcional.

- Objetos y funciones específicas para el tratamiento de datos.
- R es software libre. Permite la descarga de librerías con implementaciones concretas de funciones gráficas, métodos estadísticos, algoritmos...
- En su momento el lenguaje S evolucionó por un lado hacia R, software libre con licencia GNU, y por otro hacia S-plus, software comercial y con un entorno gráfico mucho más "amigable". Esto hace que pasar de R a S-plus o al revés sea relativamente sencillo (bajo los dos subyace el mismo lenguaje de programación).

Estas notas pretenden ser una guía esquemática para una introducción superficial a R y su entorno, para profundizar en cualquier punto de ellas se pueden consultar tanto los manuales específicos como la propia ayuda de R (a todo ello se puede acceder desde el menú help).

4.2.1. Comandos y conceptos básicos

Aquí se describe únicamente la utilidad específica de cada comando, para consultar la sintaxis precisa de cada uno de ellos habrá de echar mano de la ayuda o de los manuales.

Obteniendo ayuda: Para obtener ayuda acerca de, por ejemplo, la función plot, se podrían utilizar los siguientes comandos ?plot, help(plot), help.search(plot). Los dos primeros son equivalentes y el tercero permite realizar búsquedas más avanzadas. Además, también se pueden utilizar los correspondientes elementos del menú help.

- **Case sensitivity:** El lenguaje R es "case sensitive", es decir, distingue entre mayúsculas y minúsculas.
- **Robustez:** El lenguaje R es un lenguaje robusto. En otras palabras intenta, en la medida de lo posible, no dar mensajes de error. Esto en general es cómodo, pero hay que tener cuidado, es posible que al ejecutar un comando no obtengamos ningún error y sin embargo R no esté haciendo lo que nosotros pretendíamos.
- **Importar código:** El comando *source("comandos.R")* permite introducir comandos procedentes de archivos. Útil para cargar funciones elaboradas por el usuario o bien para ejecutar macros y scripts.
- **Exportar:** El comando *sink("archivo")*, nos permite que la salida de los comandos posteriores se almacene en *"archivo"*. Para devolver la salida a la pantalla se usa nuevamente el comando *sink()*. Usado principalmente para guardar salidas de datos en archivos de texto.
- **Funciones:** Para consultar el código de una función en R, ya sea una función propia de R o bien una del usuario, basta con teclear el nombre en la línea de comandos (sin paréntesis ni ningún tipo de argumentos).

- **Workspace:** El comando *getwd()* nos devuelve el directorio de trabajo y el comando *setwd()* nos permite modificarlo. Al iniciar el programa R se abre automáticamente un espacio de trabajo, en él se almacenan todos los datos, funciones, usadas durante esa sesión. En cualquier momento se pueden guardar todos los objetos del espacio de trabajo como un archivo con extensión ".RData" (menú Archivo). Como norma general, las funciones y objetos de datos es mejor guardarlos en archivos propios; esto evita tener objetos innecesarios en el "workspace". La mejor forma de hacer esto será guardando uno o varios objetos en un archivo (de extensión .RData) con el comando *save(objectos, file="nombre.RData")*. Si en sesiones posteriores se necesitan se pueden cargar con la función *load()*.

- **Bibliotecas:** Al iniciar el programa R se cargan por defecto unas bibliotecas básicas. A veces es necesario cargar otras librerías para realizar ciertos análisis, o llamar a algunas funciones "no básicas". Esto se hace a través del comando *library(nombre)*.

- **Listados:** El comando *ls()* nos proporciona un listado de los objetos que hay actualmente en el espacio de trabajo. Para hacer búsquedas de objetos en el espacio de trabajo se pueden utilizar los comandos *apropos()* y *find()*. Además, el comando *search()* nos da una lista de las librerías cargadas (más adelante veremos que no es lo único que lista este comando).

- **Borrado:** Para eliminar uno o varios objetos del espacio de trabajo se usa el comando *rm(objectos)*.

- **Historial de comandos:** Mediante el teclado (flechas) se puede acceder a los últimos comandos ejecutados. Además, el comando *history()* nos devuelve un archivo de texto con los últimos comandos ejecutados.

4.2.2. Objetos y operaciones básicas

En el lenguaje R se puede trabajar con varias clases de objetos; algunos de ellos son estándar en cualquier lenguaje de programación y otros son objetos específicos de R, objetos pensados para ser manejados con propósitos estadísticos.

Vectores

Son la estructura de datos más sencilla con la que trabaja R. A continuación mostramos varios ejemplos de asignaciones para obtener vectores. R utiliza el operador de asignación "$<-$" (también se puede poner en sentido contrario), en general no se hacen asignaciones con "=".

IMPORTANTE: Hay que acostumbrarse a no usar "=" como operador de asignación en R. Normalmente su uso no devuelve ningún mensaje de error, pero tampoco realiza la asignación deseada. `x<-c(2,4,6,8,10)`, proporciona un vector formado por 5 números pares. La función *c()* se usa para concatenar objetos, aunque resulta especialmente cómoda para crear vectores. En el ejemplo anterior x[4] nos devolvería el valor 8.

`x<-numeric(50)`, crea un vector de 50 componentes, todas ellas 0. También se podrían definir vectores usando otros modos distintos: character, double, etc. `x<-1:5; y<-5:1`, estas asignaciones devolverían vectores x e y con los 5 primeros números naturales en orden creciente y decreciente respectivamente.
`seq(1,9,by=2)`, devuelve 5 primeros números impares (ir de 1 a 9 con paso de 2).
`seq(1,9,lenght=6)`, devuelve los números 1.0 2.6 4.2 5.8 7.4 9.0 (ir de 1 a 9 en 6 pasos).
`rep(3,10)`, proporciona un vector de 10 coordenadas, todas ellas con valor 3.
`x<-c(''a'',''e'',''i'',''o'',''u'')`, devuelve un vector de caracteres con las 5 vocales. Todos los elementos de un vector han de ser del mismo tipo, pero se admiten dos excepciones, los valores NA (not avaliable) y NaN (not a number). Estos dos valores son muy importantes en el análisis estadístico: NA porque a veces hay valores perdidos en algunos campos de una muestra y NaN puede ser el resultado de alguna indeterminación. R nos permite hacer operaciones con vectores aunque en alguna de sus componentes tengan NA o NaN.

Operaciones básicas con vectores. Los operadores básicos son +,-, *, /, y ^ para la potencia; estos operadores funcionan componente a componente. Por ejemplo, dados los vectores x e y, $x+y$ nos daría el vector obtenido al sumar x e y componente a componente $1/x$ nos devolvería el vector que tiene en cada componente el inverso de la componente de x. Otras funciones básicas implementadas en R son *log*, *exp*, *sin*, *cos*, *tan*, *sqrt*. Además, *sum(x)* nos devuelve la suma de los elementos de x, *prod(x)* su producto, *mean(x)* su media, var(x) su quasivarianza.

En R tenemos los dos valores lógicos: TRUE y FALSE. Los operadores lógicos son $<$, $<=$, $>$, $>=$, $==$ para igual y $!=$ para distinto. Además '&' se usa para indicar intersección ("y"), '|' indica disyunción ("o") y '!' indica la negación. Hay veces que dado un vector nos interesa hacerle algún tipo de filtro para quedarnos con los elementos de este que verifican una cierta propiedad. Por ejemplo, con los comandos:

```
x<-c(1,-1,2,-2)
x>0
```

Obtendríamos el vector: TRUE FALSE TRUE FALSE

```
x<-c(1,-1,2,-2)
y<-x[x>0]
```

El vector y estaría compuesto por los números 1 y 2 (los positivos de x). Otros posibles comandos serían: `x[1:3]` para quedarnos con los tres primeros elementos de x, `x[-(1:3)]` para quedarnos con todos los elementos de x salvo los 3 primeros. `x[c(T,F)]` nos devolvería los elementos en coordenadas impares de x, mientras que el comando `x[c(F,T)]` devolvería los de las coordenadas pares (independientemente de la longitud de x). Esto es debido a que R cuando tiene que operar con dos vectores de distinto tamaño intenta reciclar el pequeño tantas veces como sea necesario hasta igualar el tamaño del grande.

A veces interesa quitar los valores NA y NaN de un vector, esto se haría con el comando `x<-x[!is.na(x)]`. Este comando elimina del vector x todos aquellos elementos que sean

NA o NaN, para quitar sólo los NaN se puede usar el comando `x<-x[!is.nan(x)]`. Del mismo modo, `x[is.nan(x)]<-0` reemplazaría todos los NaN de x por 0. La expresión *is.objeto(x)* funciona para la mayoría de los objetos de R.

`which(is.nan(x))` devuelve las posiciones de los elementos de x que toman el valor NaN.

Arrays y matrices

Son la extensión natural de los vectores. Un array o una matriz se podrían definir del siguiente modo:

```
x <- array(1:20,dim=c(4,5))
```

genera un array de 4 filas y 5 columnas con los números del 1 al 20. En R el array se llena por columnas (al contrario que en el lenguaje C), es decir, la primera columna del array x serían los números del 1 al 4, la segunda los números del 5 al 8 y así sucesivamente. Lo más importante a la hora de definir un array es especificar el vector de dimensiones.

En el ejemplo anterior, `x[3,2]` nos daría el segundo elemento de la tercera fila de x, `x[,1]` sería la 1a columna y `x[3,]` la tercera fila.

Aunque la función array se puede usar para definir matrices, para este caso particular existe la función matrix, que nos permite decir simplemente el número de filas o de columnas de la matriz; no es necesario decir los dos ya que R adapta los datos de la única forma posible. Además, el parámetro opcional byrow permite decir si llenamos la matriz por filas o por columnas.

Operaciones básicas con arrays y matrices. Dada una matriz A, la función *t(A)* calcularía la traspuesta de la matriz A. Dado un array de dimensión cualquiera, la función *aperm()* permite obtener trasposiciones del mismo (una trasposición no es más que intercambiar los índices en la matriz, esto en general se hace permutando los índices de columnas, filas. Dada una matriz A, *nrow(A)* y *ncol(A)* nos dicen el número de filas y columnas de la matriz A. Dadas dos matrices A y B, el producto de ambas se hace mediante el operador %*%. Si A y B tienen la misma dimensión, A*B nos devolvería el producto componente a componente de sus elementos, no el producto matricial. Análogamente *sin(A)* nos daría la matriz que tiene en cada componente el seno del correspondiente elemento de A. Por otro lado, la función *solve()* aplicada a una matriz nos calcula su inversa.

Listas y data frames. Por último, presentamos dos clases de datos todavía más generales: las listas y los data frames.

Una lista es un objeto cuyas componentes pueden ser arrays, listas, Data Frames, variables lógicas, y cualquier combinación de éstos (ésto nos permite almacenar juntos datos de distinta naturaleza como nombre, edad, ingresos, trabajos anteriores...) Los distintos elementos de la lista no han de ser necesariamente del mismo tipo. Por ejemplo

```
Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

devuelve una lista con formada por 4 objetos, dos variables de tipo carácter, un valor numérico y un vector de tres componentes. Para referenciar a cada objeto de la lista se hace con `Lst[[i]]` de este modo `Lst[[1]]="Fred"` y `Lst[[4]][3]=9`. Al trabajar con listas puede ser más cómodo usar los nombres de cada objeto de la lista, de este modo `verbLst$wife="Mary"`. Para saber los nombres de los objetos que componen la lista se usa *names(Lst)*.

Los data.frames (campos de datos) son el objeto más habitual para almacenar datos. La forma de pensar en un data.frame es considerar que cada fila representa a un individuo de una muestra y el correspondiente valor para cada columna se corresponde con la medición de alguna variable para ese individuo (fila-individuo, columna-variable). Si por ejemplo tenemos 100 alumnos y las notas numéricas de ellos en cada examen junto con la calificación final (tipo carácter), esto podría en un data.frame de 100 filas con una columna por cada examen y una columna más para la calificación final.

Por último, a veces resulta latoso trabajar con un data.frame o una lista con muchos campos anidados (muchos subniveles). Por ejemplo, en el caso de la lista antes definida `Lst$wife` se usaba para referirse al 2o objeto de la lista, `Lst$name` al primero y así con los demás. Esta notación se puede hacer menos pesada mediante el comando *attach(Lst)*, este comando hará que Lst "suba un nivel en el workspace" y a partir de ese momento podremos utilizar, por ejemplo, child.ages en vez de `Lst$child.ages` para acceder al vector de edades. Al hacer esto hay que tener cuidado de que en el espacio de trabajo no existan ya objetos con el mismo nombre que alguno de los objetos/columnas de la lista/data.frame. Para volver a la situación anterior se usa el comando *dettach(Lst)*. En su momento dijimos que el comando *search()* permitía ver el número de bibliotecas cargadas en el espacio de trabajo, del mismo modo permite ver los objetos a los que se ha "subido un nivel" con `attach`. Dado un objeto se puede consultar su tipo, su clase (numérico, carácter, lista, matriz) a través de las funciones `mode`, `class` o bien `typeof`; si bien son muy parecidas, no son equivalentes; dependiendo de la situación puede interesar más usar una que otra. La clase de un objeto es algo más específico que su modo, si un objeto no tiene clase, class() nos devolverá lo mismo que *mode()*. Además, también existe la función attributes, que nos proporciona características de un objeto tales como dimensión, clase, nombres de columnas.

Dado un objeto de un cierto tipo, es posible forzarlo a otro distinto (siempre que esto tenga sentido para el objeto en cuestión). Esto es posible mediante la familia de funciones `as.object()`. Por ejemplo dado un data.frame, Data, la función `as.matrix(Data)` lo convierte en una matriz si esto es posible. Un uso indebido de estas imposiciones de tipo puede llevar a resultados no deseados (el que no haya habido mensaje de error no quiere decir que la conversión se haya hecho como nosotros queríamos).

Para realizar operaciones numéricas con data.frames o listas, lo más conveniente es convertir a matriz, array o vector la parte con la que se desea operar y después hacer las operaciones.

En su momento se vio cómo definir un vector a través de la función *c()*, es importante tener en cuenta que esta función se puede utilizar para concatenar cualquier tipo

de objetos, no sólo variables numéricas, enteras. De la misma familia son las funciones *cbind* y *rbind* que permiten combinar una serie de objetos bien por columnas o por filas. Habitualmente se usan para construir unas matrices a partir de vectores.

4.2.3. Procedimientos gráficos

El lenguaje R dispone de varias funciones preparadas para la representación gráfica de datos. Estas funciones se dividen en dos grandes grupos:

- Gráficos de alto nivel: Crean un nuevo gráfico en la ventana de gráficos.
- Gráficos de bajo nivel: Permiten añadir líneas, puntos, etiquetas... a un gráfico ya existente.

Gráficos de alto nivel: De entre todos los gráficos de este tipo destaca la función plot, que tiene muchas variantes y dependiendo del tipo de datos que se le pasen como argumento actuará de modos distintos. Lo más común es plot(x,y) para representar un diagrama de puntos de y frente a x. Otras funciones de alto nivel importantes son, por ejemplo, hist para dibujar histogramas, o persp para representaciones tridimensionales (superficies).

Parámetros más importantes comunes a la mayoría de gráficos de alto nivel:

- `add=TRUE`**:** Fuerza a la función a actuar como si fuese de bajo nivel (intenta superponerse a un gráfico ya existente). Hay que tener cuidado, no sirve para todas las funciones.
- `type`**:** Indica el tipo de gráfico a realizar, cabe destacar `type="p"` para representar puntos (opción por defecto), `type="l"` para representar líneas, `type="b"` para representar los puntos unidos por líneas.

Gráficos de bajo nivel: Son de gran utilidad para completar un gráfico, compararlo con otro superponiendo ambos... Destacan los siguientes:

- `lines`**:** Permite superponer nuevas funciones en una gráfica ya existente.
- `points`**:** Permite añadir puntos.
- `legend`**:** Para añadir una nueva leyenda.
- `text`**:** Añade texto.

Parámetros más importantes comunes a la mayoría de gráficos, tanto de alto como de bajo nivel (para una descripción completa de estos y otros parámetros *help(par)*):

- `pch`**:** Indica la forma en que se dibujaran los puntos.
- `lty`**:** Indica la forma en que se dibujaran las líneas.

- `lwd:` Ancho de las líneas.
- `col:` Color usado para el gráfico (ya sea para puntos, líneas...)
- `font:` Fuente a usar en el texto.

Una vez que en la ventana de gráficos tenemos algo representado, existen dos funciones que nos permiten trabajar interactivamente sobre dicha ventana

La función *locator()* sitúa el cursor en la ventana de gráficos, cada vez que pulsemos el botón izquierdo del mouse nos devolverá las coordenadas del punto en el que hayamos marcado. La función *identify()* nos permite marcar uno de los puntos del gráfico (con el ratón, igual que antes), y nos devuelve la componente de los vectores representados que dio lugar a ese punto. Muy útil en estadística para identificar *outliers*.

4.2.4. Programando en R

La principal ventaja de R (y S-plus) sobre la mayoría del software estadístico es que permite combinar sus bibliotecas especializadas con un lenguaje de programación propio. Esto resulta mucho más potente que, por ejemplo, las macros en SPSS. A continuación veremos las principales características de la programación en R.

- **Agrupando comandos:** R permite escribir varios comandos en la misma línea. Esto se hace poniendo un marcador de final de comando antes de empezar el siguiente. El marcador utilizado en R es ";".
- **Expresiones:** Una serie de comandos escritos entre llaves forman una expresión, el resultado de la misma será el de su último comando; la estructura general de una expresión es $\{cmd\_1; cmd\_2; ...; cmd\_n\}$. Puede haber expresiones anidadas, que se van evaluando de dentro hacia fuera. Son el componente principal de funciones, bucles.
- **Ejecución condicional:** Como en todos los lenguajes de programación, esto se hace con las sentencias if. Estas sentencias en R tienen la siguiente sintaxis:
  $if\ (expr\_1)\ \{expr\_2\}$
  $else\ if\ (expr\_3)\ \{expr\_4\}$
  $else\ \{expr\_5\}$
  donde $expr\_1$ y $expr\_3$ deben devolver un valor lógico. En R también se pueden construir sentencias if anidadas (no es más que anidar expresiones). La sentencia condicional switch también está definida en R.
- **Bucles R** admite los bucles for, repeat y while, su sintaxis es la siguiente:
  - `for:` $for(name\ in\ expr\_1)\ expr\_2$. Por ejemplo, la sintaxis para un bucle que mueva un índice i desde 1 hasta n es:

    ```
    for (i in 1:n) {expr}
    ```

- `repeat`: *repeat expr*. En este caso hay que asegurarse de que en algún momento de expr se llega a un `break/return` que nos saca del bucle.
- `while`: *while*(*cond*) *expr*.

- **Depurando el código:** Las funciones *print()* y *cat()* nos permiten imprimir datos en pantalla durante la ejecución de un programa. Útil tanto para presentar la salida de resultados como para depurar el código.

Funciones: El lenguaje R permite al usuario definir sus propias funciones. El esquema de definición de una función es muy sencillo, aunque no por ello deja de ser potente

```
nombre_func <- function(arg_1, arg_2, ...) expr
```

Si queremos que por ejemplo `arg_2` tome por defecto el valor `'val'` bastaría escribir

```
nombre_func <- function(arg_1, arg_2=val, ...) expr
```

Todo argumento que no tenga un valor por defecto será obligatorio. Una función tomará el valor de la expresión, es decir, el valor del último comando ejecutado en `expr` (que no tiene por qué ser el último comando de `expr`, la sentencia `return` puede ser utilizada para devolver un valor por el medio de una expresión y terminar la ejecución de la función).

Las funciones en R pueden ser recursivas; una función puede llamarse a sí misma. En una función en R se distinguen los siguientes tipos de variables:

- **Parámetros formales:** Son los argumentos de la función, cualquier modificación que se haga sobre los mismos se pierde al salir de la función. Si se quiere que un cambio permanezca, la asignación debe hacerse con el operador $<<-$.

- **Variables locales:** Aparecen en la función al serles asignado algún valor, desaparecen al salir de la función.

- **Variables libres:** Son variables que aparecen en una función sin que sean parámetros ni se les haya asignado previamente ningún valor. R busca de dentro de la función hacia fuera (podemos tener funciones anidadas) hasta que encuentra alguna variable que con su nombre. Esto se conoce como "alcance lexicográfico" y es una de las principales diferencias entre R y S-plus (este último buscaría directamente una variable global). Hay que ser muy cuidadoso con el uso de este tipo de variables.

4.2.5. Bibliotecas

Al iniciar una sesión en R se cargan una serie de bibliotecas básicas, pero en muchas ocasiones es necesario cargar bibliotecas específicas que tengan implementadas algunas funciones concretas. A continuación listamos las bibliotecas que podremos llegar a usar a lo largo de la asignatura:

- Bibliotecas básicas (se cargan por defecto)
  - `base`: Contiene las bases de R.
  - `ctest`: Biblioteca para realizar test estadísticos clásicos.
  - `methods`: Biblioteca para trabajar en R con métodos y clases (no la usaremos).
  - `modreg`: Trae implementadas técnicas para regresiones.
  - `nls`: Biblioteca para llevar a cabo regresiones no lineales.
  - `mva`: Análisis multivariante.
  - `ts`: Series de tiempo.
- Bibliotecas específicas
  - `MASS`: Librería bastante variada, tiene desde funciones avanzadas para la realización de histogramas, hasta transformaciones box-cox y funciones para el análisis discriminante.
  - `lattice`: Librería con gran cantidad de funciones gráficas implementadas.
  - `scatterplot3D`: Librería para hacer gráficos de dispersión en 3D.
  - `KernSmooth`: Funciones para inferencia no paramétrica a través de estimadores tipo núcleo.
  - `foreign`: Librería para importar datos, principalmente de SPSS.
  - `RODBC`: Importar datos de bases usando ODBC.
  - `stepfun`: Librería para representar funciones escalonadas (funciones de distribución empíricas).
  - `gregmisc`: Herramientas estadísticas varias.
  - `DAAG`: Herramientas para el análisis gráfico de datos.

4.2.6. Importando datos

Ya hemos resaltado anteriormente que R es un entorno pensado para el análisis de datos, por tanto es importante ser capaces de importar datos externos, a continuación se muestra como importar datos de SPSS, Excel y archivos de texto:

- **Teclado:** Además de las formas vistas hasta ahora para introducir datos por teclado existen otras dos. La primera es la función *scan()*; después de ejecutar la función hay que introducir los valores del vector separados por espacios. La segunda (mucho más cómoda), consiste en crear un vector, matriz, inicializado a 0 y después aplicarle la función *data.entry()* para abrir una hoja que nos permite editar fácilmente el conjunto de datos.

- **R:** El programa R trae con su instalación múltiples conjuntos de datos. Para ver un listado de los mismos basta teclear *data()*, para cargar uno concreto se ha de usar el comando data(name).

- **Excel:** Hay que cargar la librería RODBC, se guarda el archivo excel en el directorio de trabajo de R y se abre la conexión a la base de datos:

  ```
  conex<-odbcConnectExcel("archivo.xls")
  ```

  Esto almacena en conex las referencias a las hojas del archivo excel. El comando sqlTables(conex) nos proporcionaría las tablas disponibles, para cargar una de ellas bastaría escribir el comando

  ```
  tabla <- sqlFetch(conex, "Nombre_tabla")
  ```

- **SPSS:** Se necesita cargar la biblioteca foreign. Después obtenemos una lista con los datos del archivo spss con la ayuda del comando *read.spss(."rchivo.sav")*. El argumento `to.data.frame=TRUE` nos permitiría obtener un data.frame en vez de una lista.

- **.txt:** Si un conjunto de datos viene guardado en un archivo .txt se pueden leer con la función read.table que nos devuelve un data.frame. Entre sus argumentos se puede decir cuál ha de ser el separador de campo, si tiene encabezado o no, así como alguna otra especificación más.

Capítulo 5

Diseño de la Aplicación

Este capítulo describe el diseño de la aplicación, los elementos que la conforma, así como los diagramas que dan un vistazo del funcionamiento de la misma, desde lo que es la parte visual creada con HTML hasta la parte del procesamiento de compilación y ejecución de los flujogramas.

5.1. Diseño de la vista

A continuación se muestra el esquema de la vista de la aplicación, explicando cuáles son las partes que la componen y para qué van a servir estas tas partes dentro de la herramienta visual.

5.1.1. Maqueta de la herramienta

La figura 5.1 muestra la maqueta de la vista de la herramienta, esta se divide en tres partes importantes: Área de menús, Área de herramientas y, por último, Espacio de trabajo y área de resultados, dichas partes se describen a continuación.

5.1.2. Área de menús

Esta área será la parte de la herramienta que contendrá opciones referentes las operaciones con los archivos y los flujogramas, como por ejemplo:

- para archivos: “Abrir”, “Guardar”, “Nuevo”,
- y para la compilación y ejecución de los flujogramas.

Esta sección tendrá una longitud del 100 % del ancho de la página web que se despliegue en el explorador o una longitud mínima en la que puedan caber todos sus menús sin que se apilen unos con otros por no caber en este contenedor.

5.1.3. Área de herramientas

Esta área contendrá las operaciones que estarán disponibles en la herramienta con las cuales se podrán construir los flujogramas, estás operaciones podrían ir desde operaciones simples como la suma de números hasta operaciones más complejas como serían las referentes a cargar diferentes conjuntos de datos o la selección de alguna gráfica para que se aplique a dichos conjuntos de datos. Esta área tendrá una longitud de 200 px de ancho que será fija y una altura dinámica con respecto a la altura de la ventana del explorador.

5.1.4. Espacio de trabajo y área de resultados

Esta sección de la herramienta será donde se podrán crear los flujogramas a partir de las operaciones que se encuentren en el área de herramientas. En este el usuario podrá interaccionar mediante el uso de un dispositivo apuntador llamado *mouse* o *ratón* con el cual se seleccionarán, arrastrarán y colocarán dichas operaciones.

Las dimensiones de este espacio de trabajo tendrán una longitud mínima fija de 800 px de ancho y de 600 px de alto, la longitud máxima que se pueda tener dependerá de las dimensiones de la pantalla para lograr un ajuste de la herramienta y se tenga más espacio para programar.

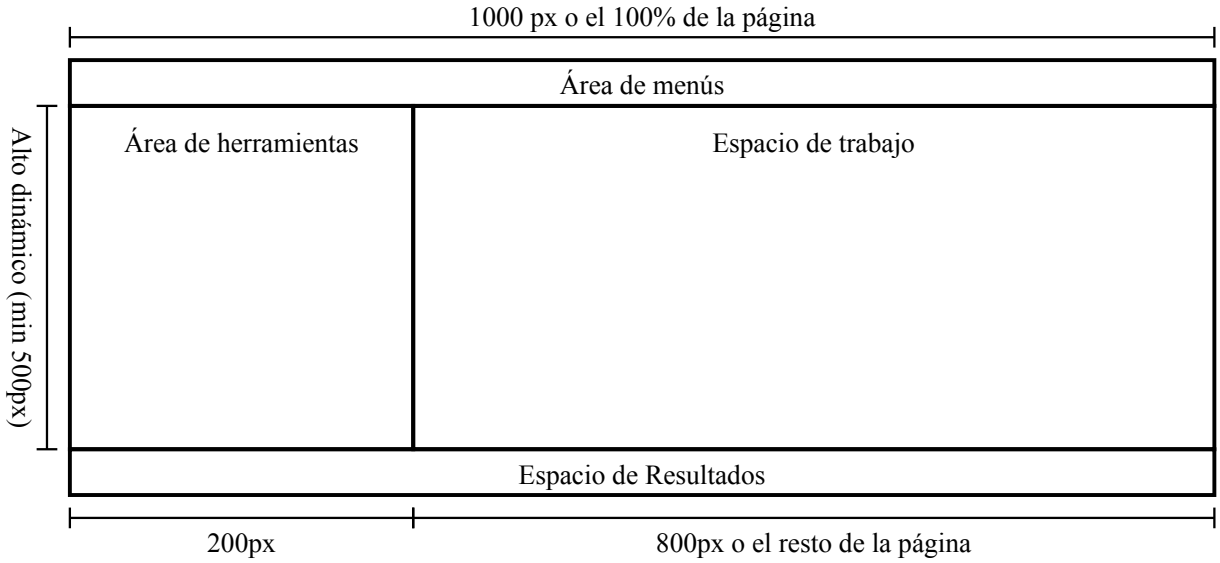


Figura 5.1: Maqueta de la vista

5.2. Descripción de REC Configurable

REC consta de una biblioteca implementada en lenguaje C; para la compilación y ejecución de programas escritos en REC, su naturaleza configurable, permite que tales funciones

sean adaptadas para diferentes propósitos. REC es una herramienta que permite escribir programas mediante un código especial, en el cual se pueden describir operadores, predicados, comentarios y cualquier tipo de sentencia, como se haría en cualquier lenguaje de programación. Este código es interpretado y ejecutado por el compilador REC, el cual después de compilar un programa lo coloca en un arreglo de apuntadores a la función que los ejecuta; de esta forma llevar acabo la ejecución del programa.

El funcionamiento descrito anteriormente se realiza por medio de las funciones *rec_c*() y *rec_x*(), las cuales se encargan de la compilación y la ejecución respectivamente, tales funciones están Incluidas en la biblioteca de REC y es suficiente con invocarlas para lograr que trabaje el compilador.

La figura 5.2, es un diagrama donde se muestra el flujo del funcionamiento de REC. Podemos apreciar que, como anteriormente se mencionó, la función *rec_c*() se encarga de la compilación del código fuente. Entonces *rec_c*() realiza la lectura del código y el análisis sintáctico. Después si no se encontraron errores en la compilación, *rec_x*() realiza la ejecución de las operaciones para producir los resultados. La función *rec_c*() se invoca de la siguiente manera:

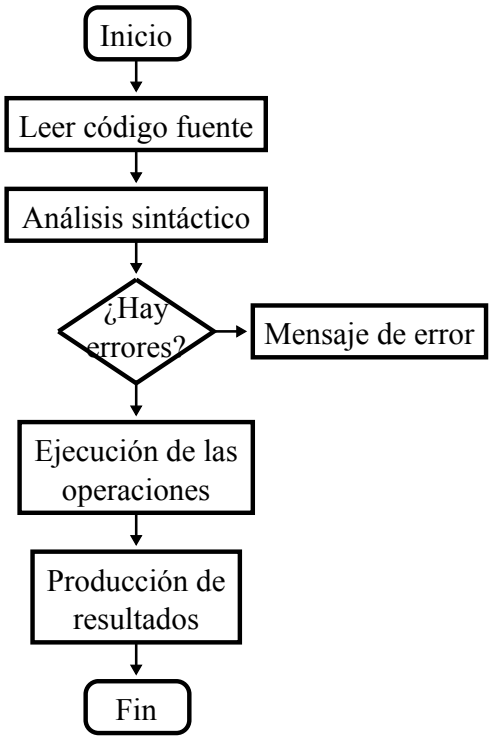


Figura 5.2: Diagrama de flujo del funcionamiento.

int rec c (char *source, Inst *prog, int plen, struct fptbl *table)

Sus principales argumentos son:

- El programa fuente (char *source). Es un apuntador al programa escrito en REC el cual debe compilarse.
- Un arreglo de apuntadores donde sería colocado el programa ya compilado (Inst *prog).
- Una tabla de predicados (struct fptbl *table) que contiene, para cada código ASCII un apuntador a la función de compilación, un apuntador a la función que ejecuta las operaciones correspondientes y un apuntador a una cadena de caracteres que sirve para poner comentarios, mismos que pueden ser desplegados durante la edición del programa.

Esta función retorna un 1 si no hay errores en la compilación y -1 si su terminación fue ocasionada por un error. Este valor debe ser usado para controlar la ejecución. La función

que se encarga de la ejecución de un programa REC es $rec\_x()$ y esta debe ser llamada solamente en caso de que el valor devuelto por la función de compilación sea 1; porque una compilación con error sería causa de problemas en la ejecución.

La función $rec\_x(Inst * prog)$ recibe como parámetro el arreglo de apuntadores donde fue colocado el programa compilado, por medio de estos apuntadores se invocan a las funciones de ejecución.

En la biblioteca de REC también están incluidas las funciones necesarias para compilar los símbolos de control (mediante estos símbolos se controla la ejecución del programa) esenciales para la estructura de REC y funciones para compilar y ejecutar predicados con estructuras más complejas.

Un elemento importante en la biblioteca de REC, es la tabla de predicados, la cual puede ser modificada de acuerdo a las necesidades de cada aplicación. En esta tabla se especifica el símbolo ASCII, la función de compilación y la función de ejecución asociadas a este.. La estructura fptbl es declarada en rec.h como sigue:

```
struct fptbl {
   int (*cfun)(); /* función de compilación*/
   int (*xfun)(); /* función de ejecución*/
   char *r_cmnt; /* comentario descriptivo */
};
```

Esto es, que por cada símbolo ASCII que es leyendo del código fuente de REC, el compilador, mediante un apuntador, localiza la función que lo compila y la función que lo ejecuta. Por ejemplo tenemos el siguiente código REC:

```
( @x ; )
```

Enseguida vamos a ver cómo el compilador encuentra las funciones de compilación y de ejecución para el carácter leído del código fuente. En este caso el carácter es @. La entrada para el carácter @ en dicha tabla es la siguiente línea:

```
r_pred1, r_call, "@ - llama a la subrutina",
```

donde r_pred1 es una función para compilar un operador simple, *rcall* es la función definida por el programador para un carácter ASCII en particular, en esta función se especifican los procedimientos a ser ejecutados; enseguida tenemos la cadena de caracteres delimitada por doble comilla, la cual nos proporciona información acerca del propósito de la función. Se tiene una línea con los tres parámetros como en el ejemplo anterior, para cada carácter ASCII en la tabla de predicados. Para explicar mejor lo anterior, se puede decir, que según el orden (en la tabla de predicados) en que se encuentren la función de compilación, la función de ejecución y los comentarios, es como se asignan a los caracteres del código ASCII. Cuando el compilador encuentra un carácter en el código fuente REC, será localizado en la tabla de predicados y ejecutará las funciones

correspondientes a ese carácter. Si se quiere adecuar el compilador de REC para un caso particular únicamente se necesita proporcionar la tabla de predicados e implementar las funciones con las operaciones correspondientes para cada predicado, según las necesidades para cada aplicación.

Aparte de darle un enfoque estadístico a REC se harán varias adaptaciones a REC, estas son:

- Realizarlo en lenguaje Java para facilitar la interacción con R.
- los nombres de las rutinas serán de más de un sólo carácter.
- las rutinas aceptarán un número ilimitado de parámetros.
- declaración de variables

5.3. Análisis de Requisitos

5.3.1. Requisitos funcionales

- El sistema debe permitir construir flujogramas.
- El sistema debe permitir guardar los flujogramas.
- El sistema debe permitir ejecutar los flujogramas.
- El sistema debe generar un lenguaje intermedio en REC cuando se ejecuta el flujograma.
- El sistema debe permitir cargar un flujograma que se halla guardado previamente.
- El sistema debe contar con un panel de herramientas.
- El sistema debe contar con un espacio de trabajo donde se construirán los flujogramas.
- El sistema debe mostrar el resultado de la ejecución del flujograma.

5.3.2. Requisitos no funcionales

- La aplicación requiere un explorador web que dé soporte a la especificación para HTML 5, ya que los elementos en la vista están implementados en nuevos elementos como las etiqueta <svg> para el despliegue de gráficos vectoriales con la cual se dibujarán los flujogramas.

5.4. Diagramas

Esta sección muestra los diagramas que se construyeron para la implementación de la herramienta, ayudan a realizar de una forma más rápida las aplicaciones.

5.4.1. Diagrama de casos de uso

Este es un diagrama general de casos de uso de la herramienta.

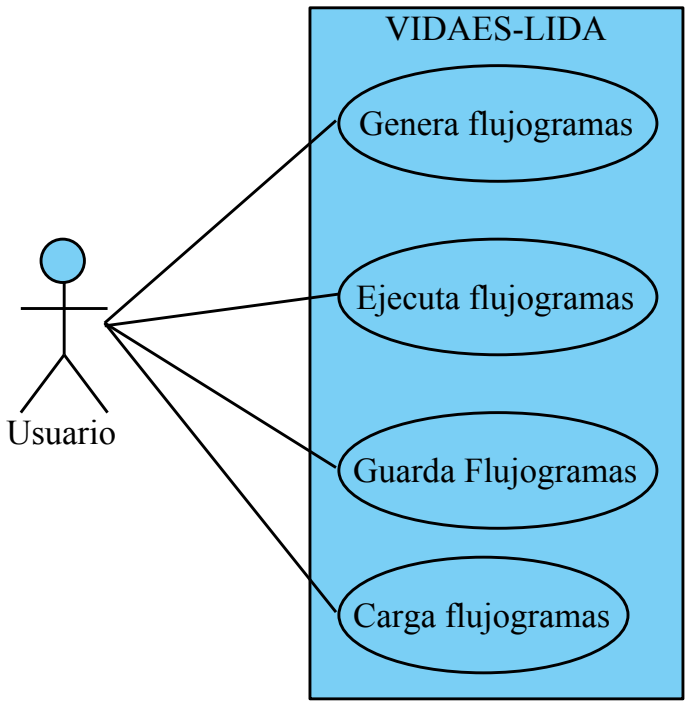


Figura 5.3: Diagrama de Casos de Uso general de la Herramienta

Caso de uso: Genera flujograma.
Actor: Usuario.
Pasos para el escenario:

1. Selecciona la opción de "Nuevo" de la Área de menús.
2. Selecciona los operadores desde la Área de herramientas.
3. Arrastra los elementos hacia el Espacio de trabajo.

Caso de uso: Ejecuta flujograma.
Actor: Usuario.
Pasos para el escenario:

1. Una vez que el flujograma es generado, se selecciona la opción .ªEjecutar"de la Área de menús.

2. Si no hay errores se muestra al usuario el resultado, de lo contrario se informa al usuario sobre el error.

Caso de uso: Guarda flujograma.
Actor: Usuario.
Pasos para el escenario:

1. Una vez que el flujograma es generado, se selecciona la opcion "Guardar"de la Área de menús.

2. Si ya ha sido guardado anteriormente, simplemente se sobre escribe con el mismo nombre, si no se ha guardado, a abre un cuadro de diálogo para guardar en la ubicación y con el nombre que el usuario especifique.

Caso de uso: Carga flujograma.
Actor: Usuario.
Pasos para el escenario:

1. Selecciona la opción de "Abrir" de la Área de menús.

2. Se muestra un cuadro de diálogo para que el usuario seleccione la ubicación y el archivo con el flujograma.

3. Da aceptar en el cuadro de diálogo, y enseguida se carga en el espacio de trabajo.

5.4.2. Diagrama de clases

El siguiente diagrama es el diagrama de clases para esta herramienta visual, la clase *IconoElemental*, como vimos anteriormente, va a servir para formar mediante la composición de estos a los íconos compuestos, la relación que mantiene con la clase *IconoCompuesto* es de composición; la clase *Vista* sirve para representar la interfaz con el usuario; la clase *Control* es la que provee el framework de Java para aplicaciones Web y es la encargada de procesar las peticiones que vengan de la vista; la clase *RCaller* es la interfaz la cual provee los mecanismos para hacer una comunicación con el sistema R.

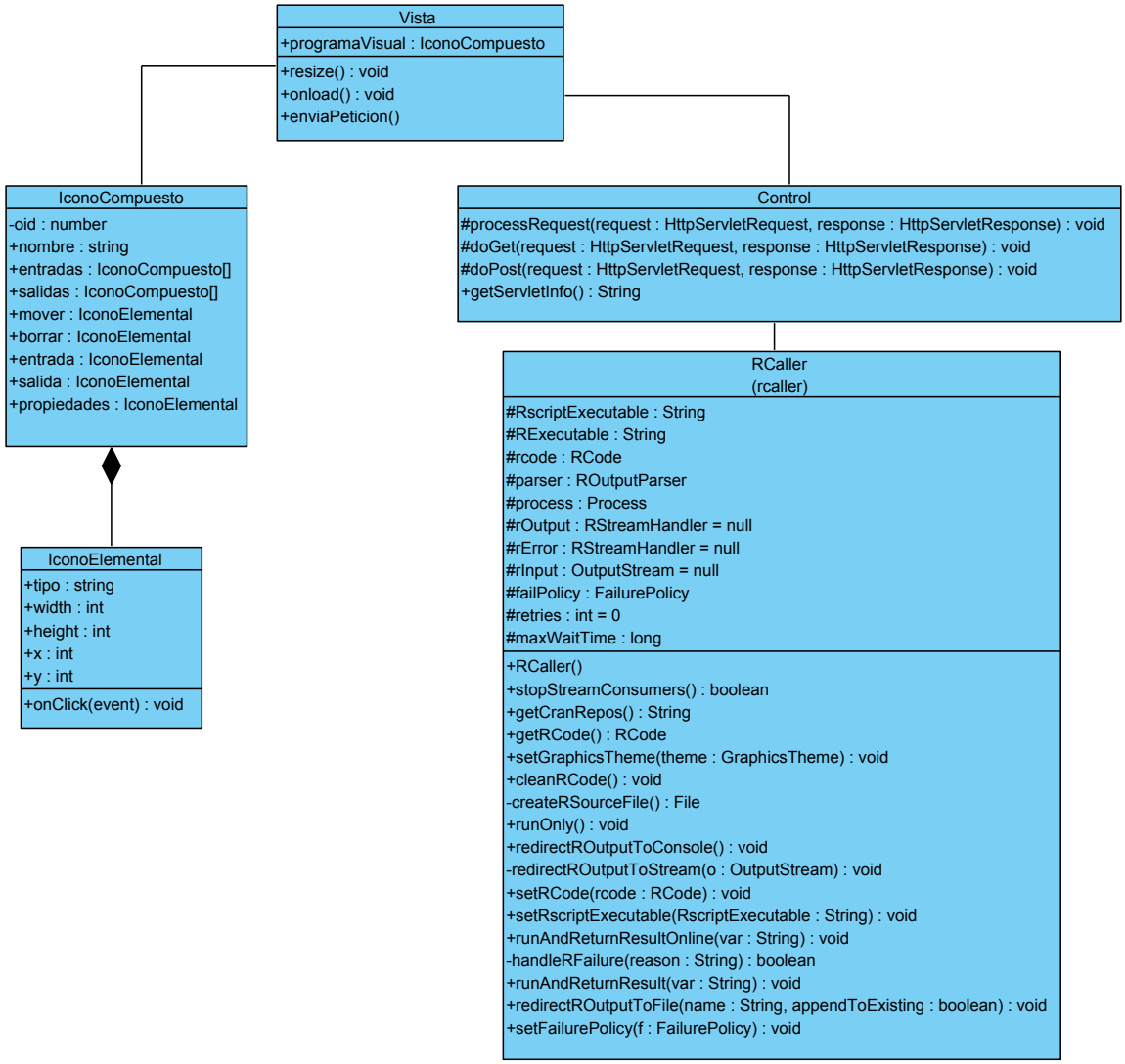


Figura 5.4: Diagrama de Clases

5.4.3. Diagrama de secuencia

Esta sección contiene los diagramas de secuencia de la herramienta visual, muestran la secuencia de operaciones y los objetos que intervienen completar las tareas que el usuario puede realizar. En la figura 5.5 muestra el diagrama se secuencia para cuando el usuario desea añadir íconos al espacio de trabajo seleccionándolos primero de la caja de herramientas de la vista, después se crea una instancia de un ícono compuesto y como este a su vez se compone de otros íconos, se tienen que crear; una vez que todos los íconos ha sido creados, se agregan al espacio de trabajo y se muestran al usuario.

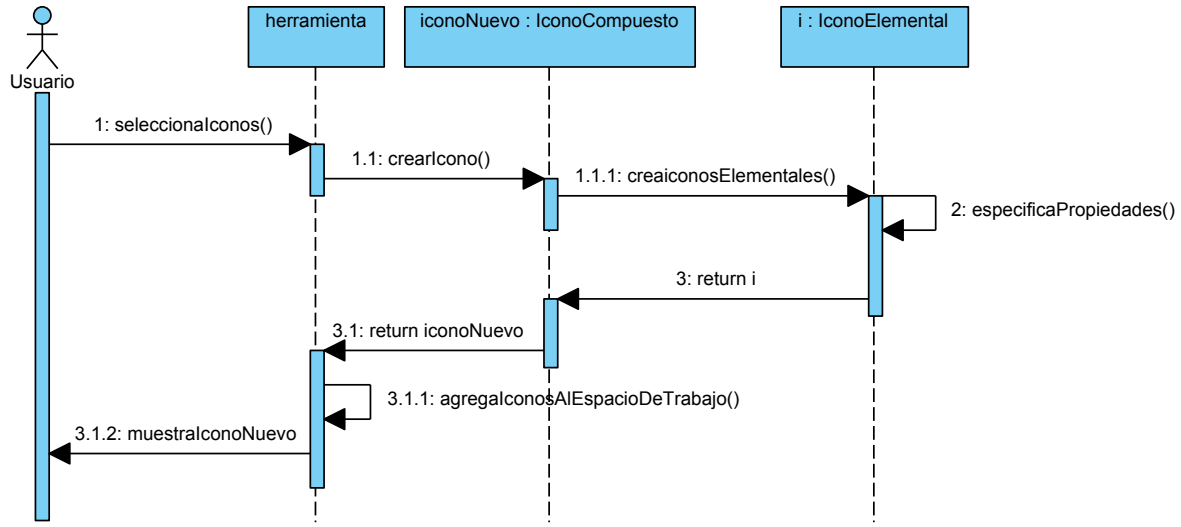


Figura 5.5: Diagrama de secuencia para añadir íconos al espacio de trabajo

En la figura 5.6 se muestra el diagrama se secuencia para cuando el usuario desea borrar íconos al espacio de trabajo, primero el usuario da click en el ícono compuesto en la parte que para borrar, después se van removiendo cada ícono del espacio de trabajo, se eliminan las conexiones que se tengan con los demás íconos y por último se actualiza el espacio de trabajo.

En la figura 5.7 se muestra el diagrama se secuencia para cuando el usuario desea conectar íconos al espacio de trabajo, primero el usuario da click en el ícono compuesto en la parte que para conectar, ya sea en la entrada o en la salida, entonces se hace activo ese ícono y se espera por el click de usuario en el ícono al que desea conectarlos, después se crea la conexión, se dibuja y por último se actualiza el espacio de trabajo.

En la figura 5.8 se muestra el diagrama se secuencia para cuando el usuario desea ejecutar el programa visual, primero se recorre cada ícono para ser interpretado y obtener el código REC del programa visual, después mediante el plug in de JQuey para JavaScript, se envía un mensaje HTTP de tipo post con el programa REC al controlador de la aplicación, después el controlador manda el programa al compilador para el lenguaje

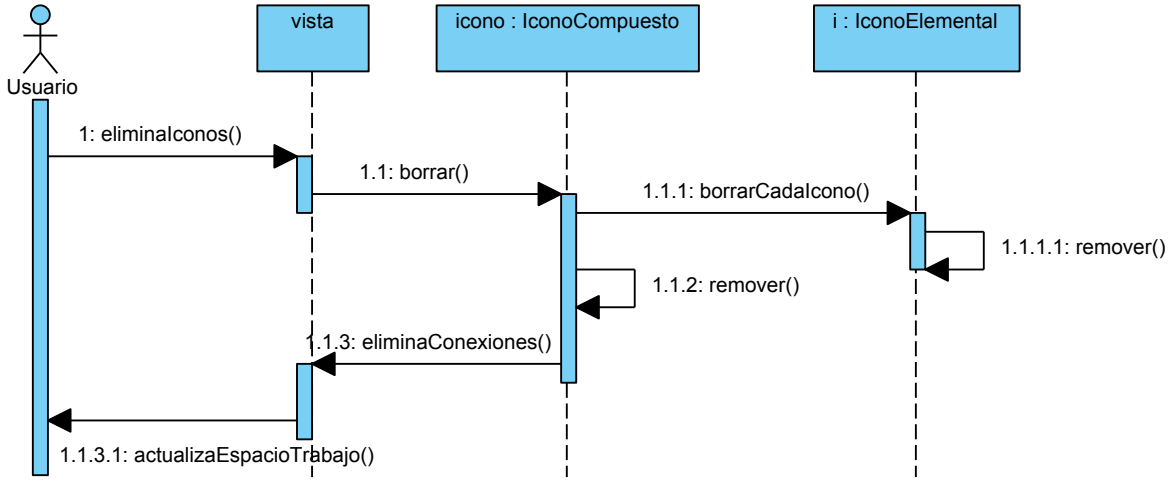


Figura 5.6: Diagrama de secuencia para borrar íconos del espacio de trabajo

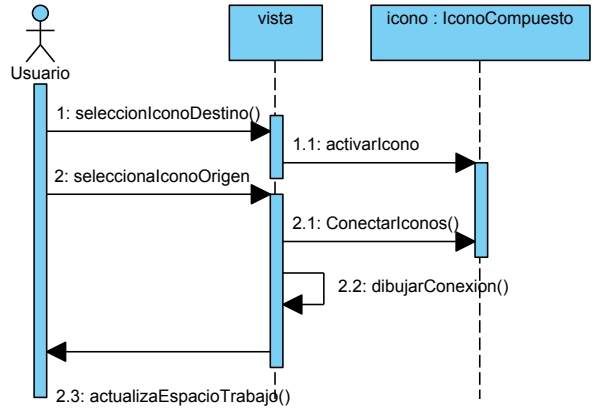


Figura 5.7: Diagrama de secuencia para conectar íconos en el espacio de trabajo

REC para poder generar el script código R, por último se ejecuta mediante la interfaz RCaller y el archivo ejecutable RScript, los resultados son devueltos a la vista para ser mostrados al usuario.

En la figura 5.9 se muestra el diagrama de secuencia para especificar las propiedades de los íconos, primero el usuario da click en algún ícono que esté en el espacio de en la parte de propiedades, en seguida se muestra sobre la vista un cuadro para que el usuario introduzca y modifique las propiedades actuales. Cuando acabe de introducirlas el usuario da click al botón .ªceptar", entonces se actualizan las propiedades del ícono, se oculta el cuadro de propiedades y se regresa al espacio de trabajo para que siga trabajando normalmente.

En la figura 5.10 se muestra el diagrama de secuencia para arrastrar los íconos a través del espacio de trabajo, se inicia cuando el usuario da click en la parte destinada a mover el ícono y sin soltar mueve el mouse a través del espacio de trabajo, entonces se le notifica

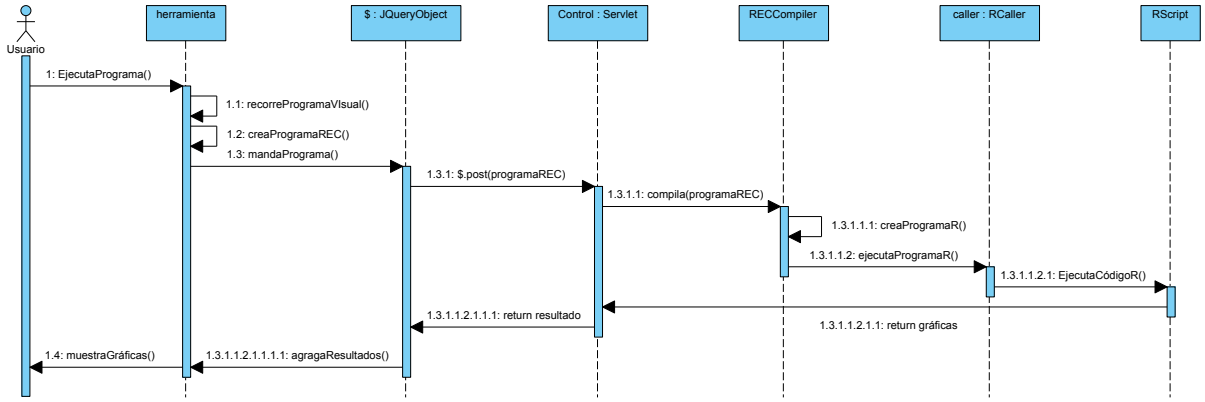


Figura 5.8: Diagrama de secuencia para ejecutar un programa visual

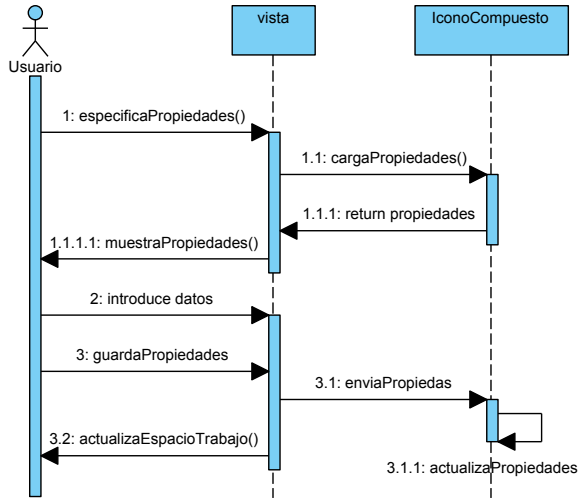


Figura 5.9: Diagrama de secuencia para desplazar los íconos en el espacio de trabajo

a la aplicación que se ha producido el evento *drag and drop*, entonces cuando esto sucede se leen los valores de la posición del mouse y el desplazamiento del mismo para recalcular la nueva posición de los íconos al igual que las conexiones. Cuando el usuario suelta se notifica de nuevo a la aplicación, se detiene el cálculo de las posiciones y se actualiza el espacio de trabajo.

En la figura 5.12 se muestra el diagrama de secuencia para cargar un archivo, primero el usuario da click a la opción abrir, aparece un cuadro de diálogo donde el usuario puede buscar su archivo con el programa visual, una vez que se selecciona se valida contra una DTD que contiene la definición y estructura que deben contener estos archivos, después se procede a leer su información la cual es pasada como parámetro para crear los íconos compuestos y las conexiones entre ellos si es que existen, después se van añadiendo al espacio de trabajo y por último se actualiza el espacio de trabajo para que el usuario

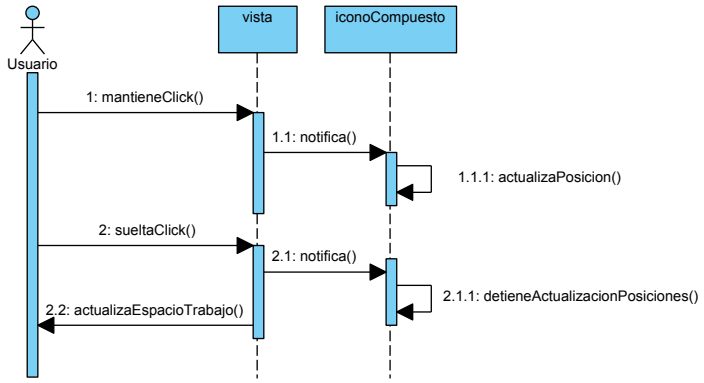


Figura 5.10: Diagrama de secuencia para especificar las propiedades de los íconos

pueda visualizar el contenido de su archivo.

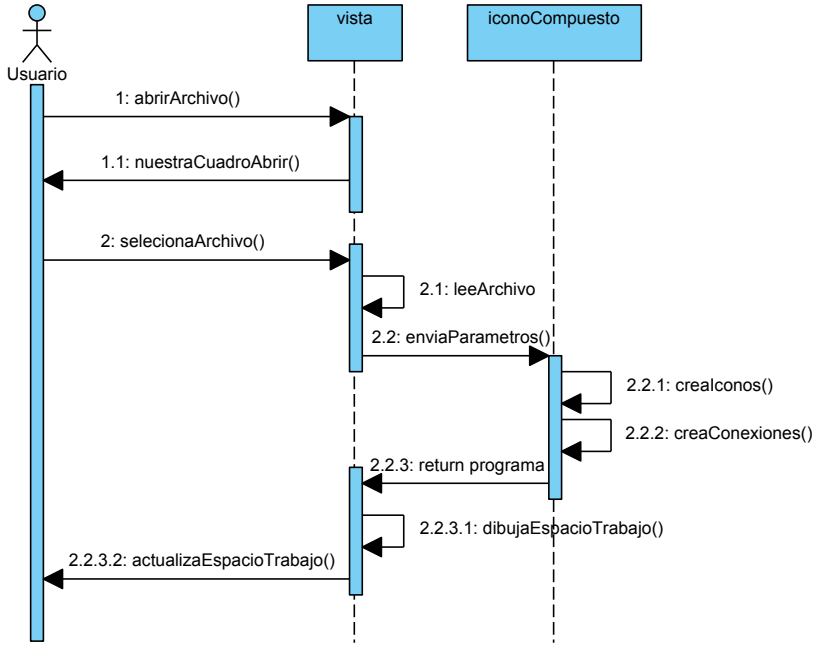


Figura 5.11: Diagrama de secuencia para cargar un archivo

5.5. Diseño del lenguaje visual

El lenguaje visual se va a componer de representaciones gráficas que se crearán mediante figuras geométricas en un espacio de dos dimensiones. La estructura de este lenguaje va a constar de un ícono compuesto, ver figura 5.12a, que será la unidad mínima en la

contrucción de las sentencias visuales, además de las conexiones entre estos íconos que indicarán el flujo de la información. Este ícono compuesto tiene un tamaño de 60px de alto por 80px de ancho, y a su vez está formado de 6 íconos base (ver figura 5.12b):

a) **Mover:** ícono que sirve para mover el ícono compuesto a través del espacio de trabajo.

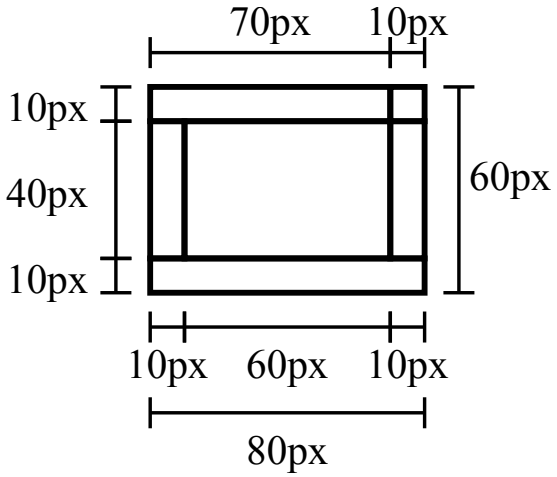
b) **Borrar:** ícono que sirve para eliminar el ícono compuesto del espacio de trabajo.

c) **Entrada:** ícono que representa la entrada de datos para el ícono compuesto.

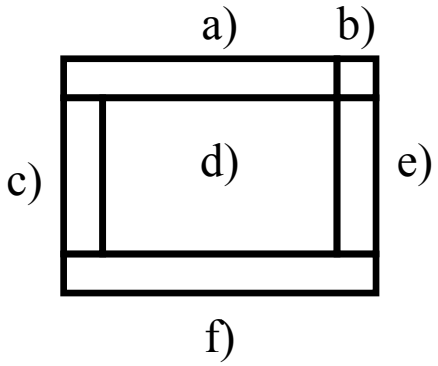
d) **Operación:** ícono que contiene la operación o función del ícono compuesto que representa dentro de la sentencia visual.

e) **Salida:** ícono que representa la salida de datos del ícono compuesto.

f) **Propiedades:** ícono que contiene las propiedades para el ícono compuesto dentro de la sentencia visual, dependerán del tipo de ícono estas propiedades.



(a) Dimensiones del ícono compuesto y de los íconos base que lo forman.



(b) Íconos base dentro del lenguaje visual: a) Mover, b) Borrar, c) Entrada, d) Operación, e) Salida y f) Propiedades.

Figura 5.12: Diseño del lenguaje visual

5.5.1. Tipos de íconos compuestos

En lenguaje visual va a tener los siguiente tipos de íconos compuesto:

1. **Variable:** representará sólo un valor, pudiendo ser un valor entero, uno flotante, una cadena de caracteres, entre otras.

2. **Arreglo:** representa un arreglo de valores de tamaño dinámico de una más dimensiones.

3. **Gráfica:** Representará a las gráficas que se pueden mostrar con los datos que tiene como entrada.

4. **Fuentes de datos:** Representará los datos que contiene en sistema R disponibles en su instalación, desde archivos de entrada en los cuales los datos se encuentran separados por algún carácter y desde sistemas gestores de bases de datos.

5. **SQL:** Representará los datos que provenientes de sistemas gestores de bases de datos para que sean leídos, poder procesarlos y opcionalmente descritos de manera gráfica.

6. **Código:** Representará partes de código puro en lenguaje R como por ejemplo sentencias *if* o ciclos, y también declaraciones de funciones.

5.6. Biblioteca Raphael para el dibujado

Raphael es una biblioteca pequeña escrita en JavaScript que simplifica el trabajo con gráficos vectoriales que incorpora la especificación oficial de HTML 5 para aplicaciones web. Como esta aplicación es visual se puede utilizar para crear los íconos, dibujar imágenes, recortarlas y girarlas.

Raphael utiliza la Recomendación de la W3C para el SVG (*scalable vector graphics*) y VML (*Vector Markup Language*) como base para crear los gráficos vectoriales. Esto quiere decir que cada objeto gráfico que se cree también será un objeto DOM, así que se le pueden asignar manejadores de eventos en JavaScript o poder modificarlos después. La meta de Raphael es proveer un adaptador que para hacer el dibujado vectorial más fácil y compatible con los navegadores web.

La forma de usarlo es bastante simple, sólo hay que descargar el archivo *raphael.js* de [52] e incluirlo en la página HTML, el siguiente código en JavaScript muestra el uso de esta biblioteca.

```
//Crea un canvas de 320 x 200 en la posición 10, 50
var paper = Raphael(10,50,320,200);
//Crea un círculo en x = 50, y = 40, con un radio de 10
var circle = paper.circle(50,40,10);
//Establece el atributo fill del círculo al color rojo (#f00)
circle.attr('fill','#f00');
//Establece el atributo stroke del círculo al color blanco
circle.attr('stroke','#fff');
```

Capítulo 6

Implementación

Este capítulo trata acerca de la implementación de la herramienta visual, algunos algoritmos que para explicar los pasos para realizar algunas tareas, detalles del código y la funcionalidad de lagunas APIs y las herramientas utilizadas.

6.1. Vista de la herramienta visual

Para lograr que la vista se ajustará a las dimensiones de la pantalla en la que se está mostrando se hizo uso de las propiedades de estilos que ofrece en el estándar CSS, principalmente las dos siguientes

- *min-width*: cuando se le asigna a un elemento dentro de un documento HTML específica el ancho mínimo que va a poseer.
- *min-heignt*: cuando se le asigna a un elemento dentro de un documento HTML específica el alto mínimo que va a poseer.

Con estas dos propiedades establecidas se logra un tamaño fijo aunque la pantalla o la ventana cambien un tamaño menor a los 1000 por 700 píxeles, la vista herramienta quedara con estas dimensiones.

Para controlar cuando la resolución o el tamaño de la ventana sea superior a la cantidad, se hace uso del evento *resize()* del objeto window que el lenguaje para aplicaciones web JavaScript ofrece y que representa una ventana abierta o un elemento iframe o fame dentro del explorador. Así entonces cuando sea el cado, el evento *resize()* nos notifica que hay un cambio en la dimensiones entonces, en las acciones a ejecutar están las de cambiar dinámicamente el tamaño del espacio de trabajo. Básicamente esto es lo que se ejecuta cuando el evento resize tiene lugar:

```
$(window).resize(function () {

        var wi = $(this).width();
```

```
        var he = $(this).height();

        $('#cajaherramienta').height(he-200);

        //Controla el alto y ancho del espacio de trabajo
        $('#espaciotrabajo').height(he-200);
        $('#espaciotrabajo').width(wi-200);
        //Controla el alto y ancho del canvas Raphael
        r.setSize(
                        $('#espaciotrabajo').width(),
                        $('#espaciotrabajo').height()
        );
});
```

Cabe mencionar que se hizo uso del plug in de JQuery para JavaScript, esta biblioteca hace más fácil el acceso a los elementos del árbol DOM que se genera, entonces la notación *$("id,clase o etiqueta del elemento")* es una versión reducida de la función nativa de las funciones *getElementBy....()* del objeto *document*.

De esta forma el problema del ajuste a cualquier tamaño de pantalla queda solucionado.

6.2. Dibujado e interacción de las representaciones graficas

Como se ha mencionado en el capítulo anterior la parte grafica corre por parte de la biblioteca Raphael para el dibujado vectorial que provee métodos que encapsulan métodos los métodos para cargar imágenes o formas geométricas, además de que facilita una interacción mediante la asignación de eventos a estos elementos.

Si bien JavaScript no es un lenguaje orientado a objetos, estos sí tienen lugar dentro de este lenguaje, existe un la forma de implementar algo que se conoce como *pseudoclases* son funciones especiales dentro del lenguaje para intentar hacer uso de este paradigma, con estas pseudo clases de pueden crear objetos con la posibilidad de añadirles atributos y funciones. La sintaxis se muestra en la en el siguiente fragmento de código.

```
var objeto = new FuncionPseudoclase();
```

Se usa la palabra reservada *new* para indicarle al lenguaje que se creará un objeto. Así fue como se crearon los íconos, cada objeto ícono tiene atributos como alto, ancho, posición y eventos como *onclick*, *ondoubleclick*, entre otros.

6.2.1. Conexión entre los íconos

Cada ícono tiene una parte para conectar con íconos de entrada y para conectar de salida, excepto el ícono de inicio que sólo tiene salida, para conseguir esto, el ícono elemental *entrada* tiene asociada una función para el evento *onclick* la cual maneja esta tarea, de

igual forma el ícono *salida* tiene asociada un función para el evento *onclick* que maneja también esta tarea.

Antes de empezar se declaran dos variables llamadas *hacia* y *desde*, las cuales hacen referencia al icono que va de entrada y al que va de salida respectivamente. Entonces cuando se hace click ya sea en la entrada o en la salida, se verifica si ya se seleccionó antes algún ícono para conectarlo, si así es se realiza la conexión agregando como referencia al arreglo que guarda las conexiones en ambos íconos y además se crea un objeto tipo *path* para mostrar en pantalla una línea que representa la conexión, si no entonces se asigna el ícono al que se le dio click en las variables que definimos (hacia o desde, según corresponda), y se espera a que el usuario seleccione el otro ícono. Ver los algoritmos 1 y2 donde se muestra el los pasos del proceso antes mencionado.

Algoritmo 1 Algoritmo para de manejar la conexión de la entrada

Entrada: Dos íconos *hacia* y *desde* que representan cuál es el ícono de entrada y cuál el de salida

Salida: Nada

1: **si** $(desde = undefined)$ **entonces**
2: **imprimir** "Selecciona la salida del ícono que desees que sea la entrada para este ícono"
3: $hacia \leftarrow this$
4: **si no**
5: **si** $(desde.oid \neq this.oid)$ **entonces**
6: $hacia \leftarrow this$
7: $hacia.iconosEntrada.push(desde)$
8: $desde.iconosSalida.push(hacia)$
9: $rutaAux \leftarrow r.connection(desde, hacia)$
10: $ruta1 \leftarrow \{oid : desde.oid, ruta : rutaAux\}$
11: $ruta2 \leftarrow \{oid : hacia.oid, ruta : rutaAux\}$
12: $hacia.conEntrada.push(ruta1)$
13: $desde.conSalida.push(ruta2)$
14: **imprimir** "Se ha hecho la conexión entre los íconos"
15: $desde, hacia \leftarrow undefined$
16: **fin si**
17: **fin si**

6.2.2. Desplazamiento los íconos

Cada ícono cuenta con una parte donde el usuario puede dar click, y mover el íconos a la posición donde desee y soltar el botón del mouse. Para lograr esto se hace uso de la función *drag(onmove, onstart, onend* que ofrece el API Raphael para el tipo *Element*, la función *onmove* maneja el movimiento del mouse, la función *onstart* maneja cuando se le da click al inicio y la función *onend* maneja cuando suelta el botón del mouse. Para este

Algoritmo 2 Algoritmo para de manejar la conexión de la salida

Entrada: Dos íconos *hacia* y *desde* que representan cuál es el ícono de entrada y cuál el de salida

Salida: Nada

1: **si** $(hacia = undefined)$ **entonces**
2: **imprimir** "Selecciona la salida del ícono que desees que sea la salida para este ícono"
3: $desde \leftarrow this$
4: **si no**
5: **si** $(hacia.oid \neq this.oid)$ **entonces**
6: $desde \leftarrow this$
7: $hacia.iconosEntrada.push(desde)$
8: $desde.iconosSalida.push(hacia)$
9: $rutaAux \leftarrow r.connection(desde, hacia)$
10: $ruta1 \leftarrow \{oid : desde.oid, ruta : rutaAux\}$
11: $ruta2 \leftarrow \{oid : hacia.oid, ruta : rutaAux\}$
12: $hacia.conEntrada.push(ruta1)$
13: $desde.conSalida.push(ruta2)$
14: **imprimir** "Se ha hecho la conexión entre los íconos"
15: $desde, hacia \leftarrow undefined$
16: **fin si**
17: **fin si**

caso se usan las funciones *onstart* y *onmove*. A continuación se muestra el código para manipular este evento. Se empieza por declarar el ícono sobre el cual va a recibir el evento drag, indicando la posición y algunos atributos de como el color y el cambio de cursor a move indicándole al usuario que puede mover el ícono. Después se le asignan las funciones necesarias para manejar el desplazamiento, la función *dragger* sirve para establecer el inicio del desplazamiento guardando x y y si la figura es un rectángulo o cx y cy si es un círculo a las propiedades ox y oy. La función *move* recibe dos parámetros dx y dy donde se guarda el desplazamiento del mouse, estos valores se le suman a los atributos ox y oy que se guardaron en la en la función anterior respectivamente, este proceso se realiza para cada uno de los íconos que conforman el íconos compuesto, y en seguida se actualizan las coordenadas de las líneas que representan las conexiones para ser mostradas, la función *up()* no realiza ninguna acción.

Código para realizar el desplazamiento de los íconos.

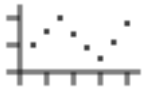
```
this.mover = raphael.rect(x, y, 80, 60,10);
this.mover.attr({fill: "white", stroke: "black", "stroke-width":1,
                      cursor: "move"});
this.mover.drag(move, dragger, up);
...
var dragger = function (){
   this.ox = this.type == "rect" ? this.attr("x") : this.attr("cx");
   this.oy = this.type == "rect" ? this.attr("y") : this.attr("cy");
};
var move = function (dx, dy){
   var mx = this.ox + dx;
   var my = this.oy + dy;

   var att = {x: mx,y: my};
   this.attr(att);

   //mover los demás íconos que lo componen
   //ícono "borrar"
   var borrar=this.borrar;
   mx_b = mx + 70;
   my_b = my + 0;

   //Se realiza lo mismo con los íconos "conexion_entrada",
   //"operacion","conexion_salida","propiedades" y "texto"

   //Ciclos para redibujar las líneas de las conexiones
   var icono = this;
```

```
   var entradas = this.super.con_entrada;
   var salidas = this.super.con_salida;
   for (var i = 0; i < entradas.length ; i++) {
      r.connection(entradas[i].ruta);
   }
   for (var i = 0; i < salidas.length ; i++) {
      r.connection(salidas[i].ruta);
   }//Se actualiza el canvas
   r.safari();
}
```

6.3. Diccionario de íconos

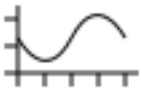
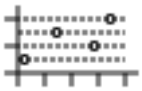
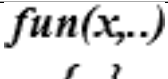
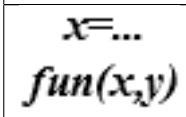
En esta sección se muestran los íconos que contienen el diccionario de íconos y que están disponibles en la herramienta para la construcción de programas visuales. La tabla 6.1 muestran a los íconos con una breve explicación.

Cuadro 6.1: Íconos disponibles en el sistema

Ícono	Descripción
INICIO	**Inicio:** Indica dónde se inicia el programa visual.
X	**Variable:** Representa una variable de un solo valor.
[1,...,n]	**Arreglo:** Representa un arreglo o una matriz.
	**Dataset:** Representa conjuntos de datos de muestra de R.
	**postgresql:** Representa datos resultantes de una conexión a PostgreSQL.
	**Plot:** Función genérica para dibujar objetos de R.

Continua en la siguiente página

Cuadro 6.1 – *Continuada desde la página previa.*

Ícono	Descripción
	**Barplot:** Crea una gráfica de barras verticales u horizontales.
	**Boxplot:** Produce una gráfica de cajas de los datos dados agrupados.
	**Histograma:** Función que calcula un histograma de los datos dados.
	**Pie:** Dibuja una gráfica de pie.
	**Curva:** Dibuja una curva correspondiente a una función en un intervalo dado. También puede representar una expresión con respecto a una variable x.
	**Dotchart:** Dibuja una gráfica de puntos Cleverland
	**Función:** Con este ícono el usuario puede crear sus funciones en R.
	**Código:** Con este íconos el usuario puede agregar sentencias en R.

6.4. Interfaz con el sistema R

La interfaz RCaller es un sorftware implementado en el lenguaje Java que provee mecanismos para hacer fácil la interacción con el sistema R, entre estas se encuentran funciones para agregar código R, arreglos enteros, booleanos, doubles, cadenas de caracteres, eso en cuanto a los datos de entrada pero también para recuperar datos resultantes o variables desde el sistema R. La forma en que estos dos lenguajes interaccionan es, por un lado desde Java a través de la clase *Process*.

Los métodos *ProcessBuilder.start()* y *Runtime.exec()* que ofrece Java crean un proceso nativo y devuelven una instancia de una subclase de *Process* que se puede utilizar para controlar el proceso y obtener información sobre él. La clase *Process* proporciona métodos para realizar la entrada desde el proceso, realizar la salida para el proceso, esperar al

proceso hasta que complete su tarea, comprobar el estado de salida del proceso, y destruir (matar) el proceso.

Todas sus operaciones de la entrada y salida estándar (es decir, stdin, stdout, stderr) serán redirigidas al proceso padre a través de tres flujos (*getOutputStream()*, *getInputStream()*, *getErrorStream()*). El proceso padre utiliza estos flujos para mandar datos a la entrada y obtener la salida del subproceso.

Ahora a estos métodos se les da como entrada varios archivos que se encuentran en el lugar donde se realizó la instalación del sistema R, estos archivos son ejecutables y son los siguientes, *Rscript.exe R.exe*, que son los encargado de leer y ejecutar fragmentos de código R, y gracias a las clases que proporciona Java, administrar la entrada, la salida y los errores que llegaran a suceder en el momento de ejecución.

Esta interfaz en Java provee funciones para agregar código en R, ejecutar este código, obtener la imagen de una gráfica en diferentes formatos y dimensiones, obtener datos y convertilos a diferentes tipos de datos de Java después de haber ejecutado algún código.

6.5. Problema con la codificación de caracteres

Los paquetes *DBI* y *RPostgreSQL* disponibles para el sistema R para la conexión y los métodos necesarios para procesar consultan tienes actualmente un problema al recuperar la información desde PostgreSQL, veamos un ejemplo a continuación en la imagen 6.1, en la cual los datos se en el campo *alumno* que están encerrados en un cuadro morado, esto no se ve para nada bien y más si se usarán para representar datos en las gráficas.

A pesar de que la codificación este correcta tanto en el sistema gestor PostgreSQL como en el cliente que se conecta a este para obtener datos, la figura 6.2 muestra la codificación actual del cliente y del servidor, las cuales están con el mismo valor de codificación.

Para resolver este problema, se hizo uso de una función que tiene el Sistema R, esa función es *Encoding*, esta función lee o establece la codificación de un vector de caracteres. Ahora el parámetro para esta función es, como se mencionó, un vector de caracteres, el tipo de dato que regresa una consulta, es del tipo *data.frame* que contiene filas y columnas, entonces dar como entrada este tipo *data.frame* a *Encoding()* produciría un error, pero la ventaja que se tiene sobre los *data.frame* es que se pueden recorrer por columnas, y cada columna se comporta como un vector, entonces se creo la siguiente función para cambiar por cada columna que sea te tipo carácter a la codificación "UTF-8" para lograr una correcta representación.

El siguiente código muestra los pasos de la función propuesta en lenguaje R, que recibe como parámetro una variable llamada *datos*, lo primero que se hace checar el tipo de esta variable , si no es del tipo *data.frame* no hay nada que hacer, ahora si es de este tipo, se recorre columna por columna checando el tipo. si es "carácter" se aplica la función de codificación nativa de R y se envía como parámetro la codificación a "UTF-8", al concluir este ciclo se regresa los datos ya codificados.

La figura 6.3 muestra el resultado de aplicar la función propuesta a los datos de la consulta para corregir la codificación.

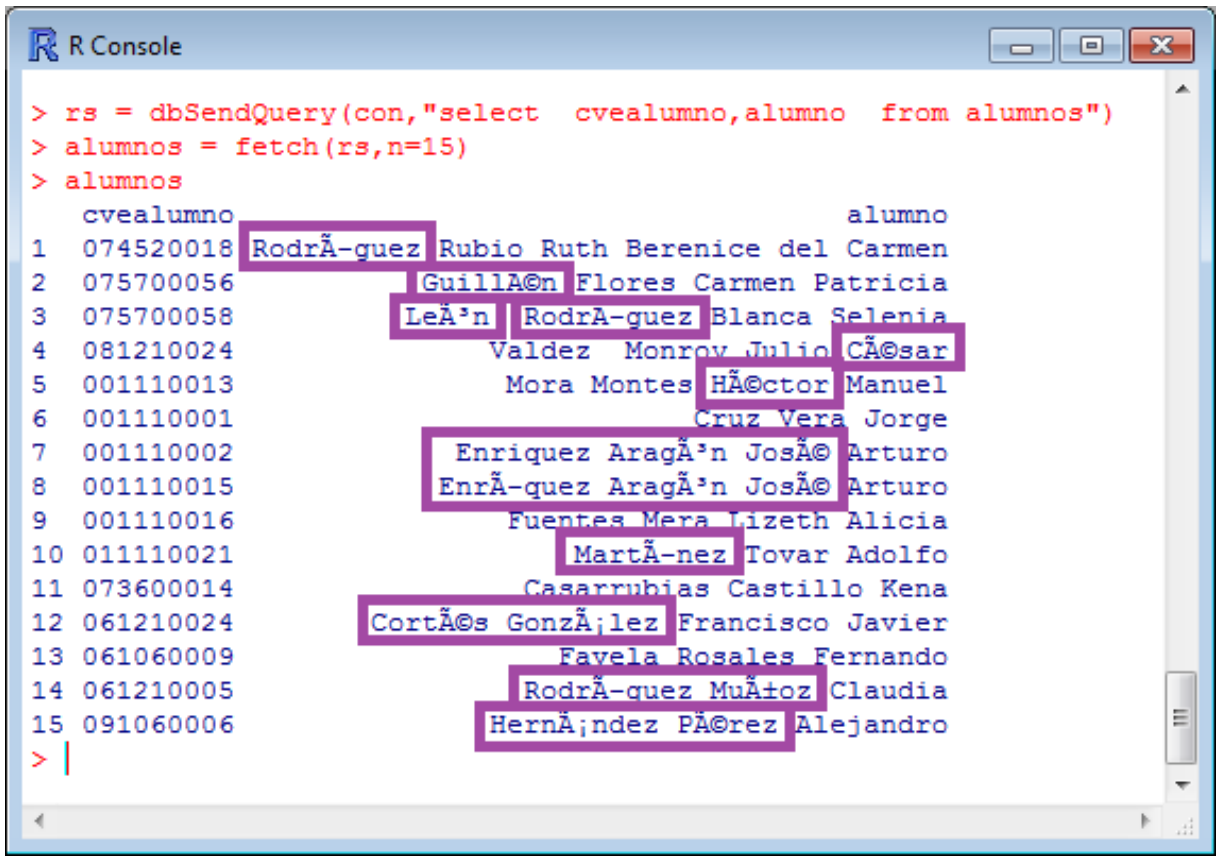


Figura 6.1: Resultado de una consulta con errores de codificación.

6.6. Formato XML para guardar los programas visuales

Una de las razones del desarrollo de esta herramienta es poder guardar los programas visuales, poder cargarlos y ejecutarlos tantas veces como el usuario lo necesite. El formato para guardar estos datos es mediante la definición de archivos XML. Ya se mencionó antes las características que XML posee ya que también son aplicables a XHTML, las cuales dicen cómo deben que ser formarse las etiquetas, los atributos y los valores de estos atributos. Lo que quiere decir que hace falta un mecanismo para validar completamente estos archivos. Una forma de realizar este proceso es mediante la definición de un archivo DTD (*Document Type Definition*), y su propósito es definir los componentes básicos legales de un documento XML. Una DTD define la estructura del documento con una lista de elementos legales y atributos [53]. Una DTD puede ser declarada en línea dentro de un documento XML, o como una referencia externa.

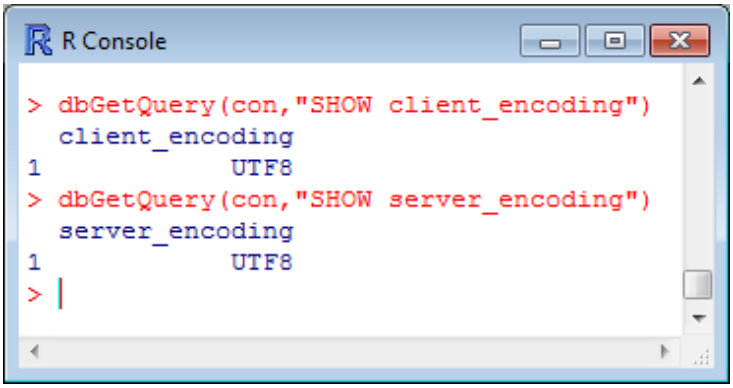


Figura 6.2: Codificación actual del cliente y del servidor, nótese que tienen el mismo valor.

Algoritmo 3 Algoritmo para codificar correctamente el resultado de una consulta

Entrada: datos: objeto de tipo "'data.frame" de R
Salida: *datos* codificada si es de tipo "data.frame" y **falso** en caso contrario.

```
1: si (tipo(datos) ≠ "data.frame") entonces
2:    devolver falso
3: si no
4:    para i = 0 hasta numerocolumnas(datos) hacer
5:       si (tipo(datos[i]) == "character") entonces
6:          Encoding(datos[i]) < −"UTF − 8"
7:       fin si
8:    fin para
9: fin si
10: devolver datos
```

Declaración de una DTD interna: Si el DTD se declara en el archivo XML, debe ser envuelto en una definición DOCTYPE con la siguiente sintaxis:

<!DOCTYPE root-element [element-declarations]>

Ejemplo de un documento XML con una DTD interna:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
```

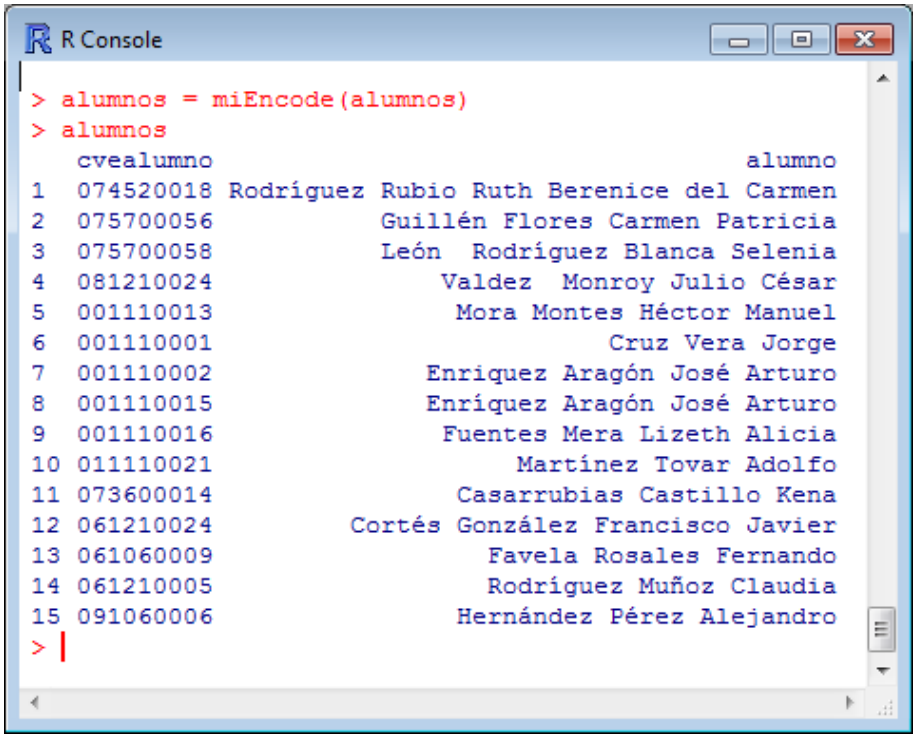


Figura 6.3: Datos después de aplicar la corrección.

<to>Tove< /to>
<from>Jani< /from>
<heading>Reminder< /heading>
<body>Don't forget me this weekend< /body>
< /note>
La DTD anterior es interpretada como sigue:

- **!DOCTYPE note** define que el elemento raíz de este documento es note.
- **!ELEMENT note** define que el elemento note contiene cuatro elementos: "to,from,heading,body".
- **!ELEMENT to,from,heading,body** se definen del tipo "#PCDATA".

Declaración externa de una DTD: Si una DTD es declara en un archivo externo, debería ser encerrado en una definición DOCTYPE con la siguiente sintaxis:

<!DOCTYPE root-element SYSTEM "nombrearchivo">

Este es el mismo documento que el de arriba, pero con una DTD externa:

<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove< /to>
<from>Jani< /from>
<heading>Reminder< /heading>
<body>Don't forget me this weekend!< /body>
< /note>
Y este es el archivo "note.dtd"que contiene la DTD:
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>

¿Por qué usar una DTD?

- Con una DTD, cada uno de los archivos XML pueden llevar a una descripción de su propio formato.
- Con una DTD, grupos independientes de personas se ponen de acuerdo para utilizar una DTD estándar para el intercambio de datos.
- La que se construya aplicación puede utilizar una DTD estándar para comprobar que los datos que recibe del mundo exterior son válidos.
- También se puede utilizar una DTD para verificar sus propios datos.

6.6.1. Bloques de Construcción XML

Los principales bloques de construcción de los dos documentos HTML y XML son los elementos. Los bloques de los documentos XML vistos desde un punto de vista DTD, todos los documentos XML (y HTML) están compuestos por los siguientes componentes básicos: Elementos, Atributos, Entidades, PCDATA y CDATA.

Elementos

Los elementos son los bloques de construcción principales de documentos XML y HTML. Ejemplos de elementos HTML son "bodyz "table". Ejemplos de elementos XML pueden ser "nota" y "message". Los elementos pueden contener texto, otros elementos , o estar vacíos. Ejemplos de elementos HTML vacíos son "hr" , "br" y "img". Por ejemplo <body>un texto< /body>, <message>un texto< /message>.

Referencia de la entidad	Carácter
&lt;	<
&gt;	>
&amp;	&
&quot;	“
&apos;	‘

Figura 6.4: Entidades predefinidas en XML

Atributos

Los atributos proporcionan información adicional acerca de los elementos. Los atributos siempre se colocan dentro de la etiqueta de apertura de un elemento. Atributos siempre vienen en pares nombre/valor. El siguiente elemento “img” tiene más información acerca de un archivo de origen: <img src=“computer.gif” / >, el nombre del elemento es “img”, el nombre del atributo es “src”, el valor del atributo es “computer.gif”. Dado que el elemento en sí es vacía que está cerrado por una “/”.

Entidades

Algunos caracteres tienen un significado especial en XML , como el signo menor que (<) que define el inicio de una etiqueta XML . La mayoría de ustedes saben de la entidad HTML : ” ”. Esta entidad ”no-breaking-space”se utiliza en HTML para insertar un espacio adicional en un documento. Las entidades se expanden cuando un documento se analiza mediante un analizador XML. Las siguientes entidades en la tabla 6.4 están predefinidos en XML:

PCDATA

PCDATA significa datos de caracteres analizados .

Piense en los datos de caracteres como el texto que se encuentra entre la etiqueta inicial y la etiqueta final de un elemento XML. PCDATA es el texto que va a ser analizado por un analizador. El texto será examinado por el analizador para las entidades y las marcas. Las etiquetas en el interior del texto serán tratadas como marcas y entidades y se ampliarán.

Sin embargo , los datos de caracteres analizados no deben contener los caracteres &, < o >, los cuales tienen que ser representados por las entidades & < y >, respectivamente.

CDATA

CDATA significa datos de carácter.

CDATA es texto que no va a ser analizada por un analizador . Etiquetas en el interior del texto no se tratan como marcas y entidades no se expandirán .

6.6.2. Implementación de la DTD para validad los archivos

A continuación se muestra la DTD creada para validad los archivos XML que contengan el programa visual creado con la herramienta desarrollada, se escogió la opción de mantener la dtd en un archivo externo, así se maneja un archivo para cualquier cantidad de archivos XML, o de lo contrario cada archivo XML aumentaría tu tamaño y sería ineficiente.

La dtd tiene como raíz el elemento *programa*, y este contendrá dos elementos, los iconos que representan las construcciones gráficas significativas que formarán las expresiones visuales; y las conexiones que se tienes entre ellos, mediante el signo "? se indica que puede o no aparecer en el archivo XML. A continuación se muestra el fragmento del archivo

<!ELEMENT programa (iconos,conexiones?)>

El elemento *iconos* podrá anidar cualquier tipo de ícono, especificado con el signo "—", y en cualquier cantidad, esta cardinalidad se representa mediante el asterisco *, que significa 0 o más ocurrencias de estos elementos, que se encuentren disponibles en el diccionario de íconos de la herramienta. A continuación el fragmento de código:

```
<!ELEMENT iconos (
  inicio|plot|pie|curva|barplot|histograma|boxplot|dataset|
  funcion|postgresql|codigo|variable|arreglo|dotchart
)*>
```

En la siguiente parte de código se muestra la definición de los elementos que contienen una etiqueta *iconos*, la palabra EMPTY significa que el elemento no contendrá otros elementos entre la etiqueta de apertura y la etiqueta de cierre.

```
<!ELEMENT inicio EMPTY>
<!ELEMENT plot EMPTY>
<!ELEMENT pie EMPTY>
<!ELEMENT curva EMPTY>
<!ELEMENT barplot EMPTY>
<!ELEMENT histograma EMPTY>
<!ELEMENT boxplot EMPTY>
<!ELEMENT dotchart EMPTY>
<!ELEMENT dataset EMPTY>
<!ELEMENT funcion EMPTY>
<!ELEMENT postgresql EMPTY>
<!ELEMENT codigo EMPTY>
<!ELEMENT variable EMPTY>
<!ELEMENT arreglo EMPTY>
```

Además de especificar que elementos contendrá la DTD, se les especificarán atributos que servirán para construir los íconos con la información que venga en el archivo XML. Esto se especifica mediante la palabra reservada ATTLIST seguido del nombre de elemento

al que se le quieren especificar es tos atributos seguido del nombre y tipo del atributo.

Todos los atributos son de tipo CDATA, sólo 4 atributos son requeridos, los cuales son el nombre, el oid (*object identifier*) y las posiciones x y y para saber dónde colocar el icono. Entonces a estos atributos se les agrega la palabra REQUIRED, mientras que a los demás atributos que no son requeridos se les agrega la palabra IMPLIED.

Atributos de *inicio*

```
<!ATTLIST inicio
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
>
```

Atributos de *plot*

```
<!ATTLIST plot
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  x_fun CDATA #IMPLIED
  y_fun CDATA #IMPLIED
  type CDATA #IMPLIED
  main CDATA #IMPLIED
  sub CDATA #IMPLIED
  xlab CDATA #IMPLIED
  ylab CDATA #IMPLIED
>
```

Atributos de *pie*

```
<!ATTLIST pie
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  x CDATA #IMPLIED
  labels CDATA #IMPLIED
  edges CDATA #IMPLIED
  radius CDATA #IMPLIED
  clockwise CDATA #IMPLIED
  init.angle CDATA #IMPLIED
  density CDATA #IMPLIED
  angle CDATA #IMPLIED
  col CDATA #IMPLIED
  border CDATA #IMPLIED
  lty CDATA #IMPLIED
  main CDATA #IMPLIED
>
```

Atributos de *curva*

```
<!ATTLIST curva
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  expr CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  n CDATA #IMPLIED
  add CDATA #IMPLIED
  xlim CDATA #IMPLIED
  type CDATA #IMPLIED
  xname CDATA #IMPLIED
  ylab CDATA #IMPLIED
  main CDATA #IMPLIED
  log CDATA #IMPLIED
>
```

Atributos de *barplot*

```
<!ATTLIST barplot
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  var CDATA #IMPLIED
  height CDATA #IMPLIED
  width CDATA #IMPLIED
  space CDATA #IMPLIED
  names.arg CDATA #IMPLIED
  legend.text CDATA #IMPLIED
```

beside CDATA #IMPLIED
horiz CDATA #IMPLIED
density CDATA #IMPLIED
angle CDATA #IMPLIED
col CDATA #IMPLIED
border CDATA #IMPLIED
main CDATA #IMPLIED
sub CDATA #IMPLIED
xlab CDATA #IMPLIED
ylab CDATA #IMPLIED
xlim CDATA #IMPLIED
ylim CDATA #IMPLIED
xpd CDATA #IMPLIED
log CDATA #IMPLIED
axes CDATA #IMPLIED
axisnames CDATA #IMPLIED
cex.axis CDATA #IMPLIED
cex.names CDATA #IMPLIED
inside CDATA #IMPLIED
plot CDATA #IMPLIED
axis.lty CDATA #IMPLIED
offset CDATA #IMPLIED
add CDATA #IMPLIED
args.legend CDATA #IMPLIED
>

Atributos de *histograma*

<!ATTLIST histograma
nombre CDATA #REQUIRED
oid CDATA #REQUIRED
x_pos CDATA #REQUIRED
y_pos CDATA #REQUIRED
var CDATA #IMPLIED
x CDATA #IMPLIED
breaks CDATA #IMPLIED
freq CDATA #IMPLIED
probability CDATA #IMPLIED
include.lowest CDATA #IMPLIED
right CDATA #IMPLIED
density CDATA #IMPLIED
angle CDATA #IMPLIED
col CDATA #IMPLIED
border CDATA #IMPLIED
main CDATA #IMPLIED
xlim CDATA #IMPLIED
ylim CDATA #IMPLIED
xlab CDATA #IMPLIED
ylab CDATA #IMPLIED
axes CDATA #IMPLIED
plot CDATA #IMPLIED
labels CDATA #IMPLIED
nclass CDATA #IMPLIED
warn.unused CDATA #IMPLIED
>

Atributos de *boxplot*

<!ATTLIST boxplot
nombre CDATA #REQUIRED
oid CDATA #REQUIRED
x_pos CDATA #REQUIRED
y_pos CDATA #REQUIRED
var CDATA #IMPLIED
x CDATA #IMPLIED
formula CDATA #IMPLIED
data CDATA #IMPLIED
subset CDATA #IMPLIED
na.action CDATA #IMPLIED
range CDATA #IMPLIED
width CDATA #IMPLIED
varwidth CDATA #IMPLIED
notch CDATA #IMPLIED
outline CDATA #IMPLIED
names CDATA #IMPLIED
boxwex CDATA #IMPLIED
staplewex CDATA #IMPLIED
outwex CDATA #IMPLIED
plot CDATA #IMPLIED
border CDATA #IMPLIED
col CDATA #IMPLIED
log CDATA #IMPLIED
pars CDATA #IMPLIED
horizontal CDATA #IMPLIED
add CDATA #IMPLIED
at CDATA #IMPLIED
>

Atributos de *dataset*

```
<!ATTLIST dataset
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  nombreDataset CDATA #IMPLIED
  nombreSimple CDATA #IMPLIED
  paquete CDATA #IMPLIED
  descripcion CDATA #REQUIRED
  esDefinido CDATA #REQUIRED
>
```

Atributos de *funcion*

```
<!ATTLIST funcion
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  nombref CDATA #IMPLIED
  parametros CDATA #IMPLIED
  cuerpo CDATA #IMPLIED
>
```

Atributos de *postgresql*

```
<!ATTLIST postgresql
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  host CDATA #IMPLIED
  puerto CDATA #IMPLIED
  usuario CDATA #IMPLIED
  pass CDATA #IMPLIED
  base CDATA #IMPLIED
  tabla CDATA #IMPLIED
  consulta CDATA #IMPLIED
  esDefinido CDATA #IMPLIED
>
```

Atributos de *codigo*

```
<!ATTLIST codigo
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  lineas CDATA #IMPLIED
>
```

Atributos de *variable*

```
<!ATTLIST variable
nombre CDATA #REQUIRED
oid CDATA #REQUIRED
x_pos CDATA #REQUIRED
y_pos CDATA #REQUIRED
valor CDATA #IMPLIED
>
```

Atributos de *arreglo*

```
<!ATTLIST arreglo
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  valor CDATA #IMPLIED
>
```

Atributos de *dotchart*

```
<!ATTLIST dotchart
  nombre CDATA #REQUIRED
  oid CDATA #REQUIRED
  x_pos CDATA #REQUIRED
  y_pos CDATA #REQUIRED
  x CDATA #IMPLIED
  labels CDATA #IMPLIED
  groups CDATA #IMPLIED
  gdata CDATA #IMPLIED
  cex CDATA #IMPLIED
  pch CDATA #IMPLIED
  gpch CDATA #IMPLIED
  bg CDATA #IMPLIED
  color CDATA #IMPLIED
  gcolor CDATA #IMPLIED
  lcolor CDATA #IMPLIED
  xlim CDATA #IMPLIED
  main CDATA #IMPLIED
```

xlab CDATA #IMPLIED >
ylab CDATA #IMPLIED

A continuación se muestra fragmento de la DTD que se encarga de guardar las conexiones entre los iconos, la etiqueta *conexiones* contiene 0 o más etiquetas *conexion*, esta etiqueta *conexion* tiene un atributo *oid* que representa al elemento, y tiene las elementos *entradas* y *salidas* que tienen los oid's con los demás elementos que tiene conexiones.

<!ATTLIST conexion
oid CDATA #REQUIRED
>
<!ELEMENT conexiones (conexion)*>
<!ELEMENT conexion (entradas?,salidas?)>
<!ELEMENT salidas (salida+)>
<!ELEMENT salida (#PCDATA)>
<!ELEMENT entradas (entrada+)>
<!ELEMENT entrada (#PCDATA)>

6.6.3. Conversión de entidades y caracteres

Como ya es menciona anteriormente algunos caracteres no pueden ser escritos directamente en un archivo XML, si no que se tienen que usar entidades definidas. Los atributos de los íconos que contienen algunas veces contienen saltos de línea o en las expresiones contienen operadores de comparación (< o >), entonces se hizo uso de la función *replace* disponible en Javascript para solucionar este problema. Las figuras 6.5 y 6.6muestra el uso de esta función, tanto para generar la cadena cambiada como para hacer este proceso inverso. Ambas funciones reciben como parámetro una cadena y regresan otra cadena reemplazando los caracteres indicados, es importante indicar el orden porque si no se podrían reemplazar los caracteres que ya se han reemplazado causando errores. EL modificador /g indica que reemplazará todas las ocurrencias dentro de la cadena.

```
function sCharXML(cadena){
   return cadena.replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&
       gt;').replace(/\"/g,'&quot;').replace(/\'/g,'&apos;').replace(/\r\n/g
       ,'&#13;&#10;').replace(/\n/g,'&#10;');
}
```

Figura 6.5: Funcion creada para codificar correctamente las cadenas de texto par aun archivo XML.

```
function sCharXMLInverso(cadena){
   return cadena.replace(/&#10;/g,'\n').replace(/&#13;&#10;/g,'\r\n').
       replace(/&apos;/g,'\'').replace(/&quot;/g,'\"').replace(/&gt;/g,'>').
       replace(/&lt;/g,'<').replace(/&amp;/g,'&');
}
```

Figura 6.6: Función creada para codificar inversamente los cadenas de texto desde un archivo XML.

6.6.4. Procesamiento del archivo XML

En esta sección se describen las dos operaciones básicas con los archivos XML generados por la herramienta visual, estas operaciones son la de "Guardarz "Abrir". En la subsección 6.6.4 se muestran los pasos y las funciones requeridas para poder guardar correctamente el archivo XML y en la subsección 6.6.4 se muestran los pasos y funciones necesarias para poder cargar un archivo XML de entrada.

Guardar el archivo XML

Antes de guardar el archivo se tienen que construir las cadenas de texto que representan a cada íconos. Para conseguir esto, la herramienta cuenta con una variable de tipo *Array* en la que pada ícono que se crea se agrega como referencia a este arreglo. Cada ícono tiene una función para obtener su representación que consiste en la concatenación de atributo más el valor correspondiente, si algún atributo no está definido no se concatena, además también se obtiene un la cadena te texto correspondiente a sus conexiones, entonces cuando el usuario selecciona la opción "Guardar", se realiza un recorrido de este arreglo concatenando el resultado de la función para obtener la representación. Cuando se ha terminado de recorrer todo el arreglo, ya se tiene completamente la cadena que representa al programa visual en formato XML.

Hasta este punto sólo se encuentra en memoria la cadena XML, este proceso se realizó en el lado del cliente en el navegador en el lenguaje JavaScript, falta ahora crear el archivo y agregar la cadena XML, este tipo de operaciones es imposible crear desde JavaScript ya que representa un problema de seguridad al tratar de leer los archivos y directorios del usuario. Así que esta cadena se mandó en un mensaje HTTP de tipo POST hacia el servidor, ahí sí es posible crear un archivo al cual se le graba la cadena. El plug in de JQuery ofrece funciones para hacer el envío de estos mensajes HTTP.

El algoritmo 4 muestra el uso del mensaje POST, y los parametros son la dirección de que procesará esta petición seguido de parámetros en formato JSON que serán procesados en el servidor, y por último se especifica una función para procesar la respuesta del servidor. Si todo es correcto, se abre una nueva ventana mostrando el archivo XML para que el usuario pueda guardarlo.

Algoritmo 4 Algoritmo para enviar la cadena XML al servidor y mostrar la ventana para que el usuario guarde el archivo.

Entrada: iconos: Arreglo que contiene las referencia de los íconos
Salida: **falso** si no hay íconos, **cierto** si todo se realizó correctamente

```
1: si (iconos.length < 1) entonces
2:    devolver falso
3: si no
4:    nombreArchivo ← leeNombre()
5:    xml ← generaXML(iconos)
6:    post("http://.../VIDAESApplication/Control",
7:    {opcion : "guardar", nombre : nombreArchivo, datos : xml})
8:    muestraVentana()
9:    devolver cierto
10: fin si
```

Abrir el archivo XML

Lo otra operación importante es la de abrir y leer un archivo xml para poder cargar el programa visual dentro del espacio de trabajo. Para validar el archivo XML, se lee el contenido mediante la API que proporciona JavaScript y HTML5, ofrece una forma estándar de interaccionar con archivos locales a través de la especificación del API de archivos[54].

A continuación se indican las interfaces que ofrece la especificación para acceder a archivos desde un sistema de archivos "local".

- **File:** representa un archivo individual que proporciona información de solo lectura (por ejemplo, el nombre, el tamaño del archivo, el tipo MIME y una referencia al control del archivo).
- **FileList:** representa una secuencia de conjunto de objetos File (tanto la secuencia <input type="file" multiple> como arrastrar un directorio de archivos desde el escritorio se consideran ejemplos de esta interfaz).
- **Blob:** permite fragmentar un archivo en intervalos de bytes.

Cuando se utiliza junto con las estructuras de datos anteriores, la interfaz de FileReader se puede utilizar para leer un archivo de forma asíncrona mediante el control de eventos de JavaScript. Por lo tanto, se puede controlar el progreso de una lectura, detectar si se han producido errores y determinar si ha finalizado una carga.

Para poder leer un archivo se tiene que validad si el explorador web soporta esta API, esa sencilla prueba se realiza usando el siguiente código en JavaScript

Hay dos formas de abrir un archivo, una forma sencilla de cargar un archivo es utilizar un elemento <input type="file"> estándar. JavaScript devuelve la lista de objetos File seleccionados como una secuencia FileList. La otra forma de carga de archivos consiste

```
// Comprobar el soporte de la API
if (window.File && window.FileReader && window.FileList && window.Blob) {
   // ¡Éxito! Todas las API para archivos son soportadas
} else {
   alert('La API de archivos no es soportada por este explorador.');
}
```

Figura 6.7: Fragmento de código para checar el soporte de la API para archivos.

en arrastrar archivos nativos desde el escritorio y soltarlos en el navegador, algunos navegadores tratan los elementos <input type="file"> como destinos donde soltar archivos nativos.

Se utilizó la primer técnica para esta herramienta para seleccionar los archivos mediante en control input.

Cómo leer archivos. Después de obtener una referencia de File, crea instancias de un objeto FileReader para leer su contenido en memoria. Cuando finaliza la carga, se activa el evento onload del lector y se puede utilizar su atributo *result* para acceder a los datos del archivo.

A continuación se indican las cuatro opciones de lectura asíncrona de archivo que incluye FileReader.

- **FileReader.readAsBinaryString(Blob|File):** la propiedad result contendrá los datos del archivo/objeto BLOB en forma de cadena binaria. Cada byte se representa con un número entero comprendido entre 0 y 255, ambos incluidos.
- **FileReader.readAsText(Blob|File, opt_encoding):** la propiedad result contendrá los datos del archivo/objeto BLOB en forma de cadena de texto. De forma predeterminada, la cadena se decodifica con el formato "UTF-8". Utiliza el parámetro de codificación opcional para especificar un formato diferente.
- **FileReader.readAsDataURL(Blob|File):** la propiedad result contendrá los datos del archivo/objeto BLOB codificados como una URL de datos.
- **FileReader.readAsArrayBuffer(Blob|File):** la propiedad result contendrá los datos del archivo/objeto BLOB como un objeto ArrayBuffer.

Una vez que se ha activado uno de estos métodos de lectura en el objeto FileReader, se pueden utilizar los eventos *onloadstart*, *onprogress*, *onload*, *onabort*, *onerror* y *onloadend* para realizar un seguimiento de su progreso.

A continuación en el algoritmo 5 que se muestra se muestra los pasos para poder leer el archivo, se lee con un formato de texto, y una vez que está en memoria se envía con un mensaje POST al servidor, para que ahí se haga la validación con las clases de Java especializadas para esta tarea, una vez hecha la validación del texto, se contesta con un

"OK" si todo está correcto, y si la respuesta es diferente, se asume que hay errores en el archivo. Si la respuesta es correcta, se procede a interpretar el contenido para poder crear el programa visual.

Algoritmo 5 Algoritmo para leer el contenido del archivo y enviarlo al servidor para validarlo,

Entrada: *evt*: variable que maneja el evento de seleccionar archivo
Salida: Nada si el archivo es correcto y **falso** en caso contrario.

```
1: archivo ← evt.target.file
2: si (!archivo.type.match('text.xml')) entonces
3:    devolver  falso
4: si no
5:    reader ← newFileReader()
6:    reader.readAsText(archivo, 'ISO − 8859 − 1')
7:    datos ← reader.onload(archivo)
8:    respuesta ← post("http : //.../V IDAESApplication/Control",
9:    {opcion : 'xml', file : escape(theFile.name), datos_xml : e.target.result})
10:   si (procesaRepuesta(respuesta) ≠ 'OK') entonces
11:      devolver  falso
12:   si no
13:      leeXML(respuesta)
14:   fin si
15: fin si
```

Después de haber cargado en memoria la cadena de texto necesitamos aplicar un parser para poder navegar en la estructura XML. Para conseguir esto se hace uso de la función *$.parseXML()* que proporciona el plug in de JQuery, esta función recibe una cadena de texto en formato XML, incluso HTML, y devuelve una referencia de un objeto parseado. Para poder navegar en él, se necesita crear otro objeto DOM mediante la función *$(parsedObject)*que recibe como parámetro el objeto devuelto por *$.parseXML()*. Ahora sí, se pueden usar todos los métodos para obtener los elementos de las etiquetas y sus atributos de JQuery para objetos DOM como son, *find()*, *contents()*, *text()*, *child()*, *attr()*, entre otros. A continuación el algoritmo 6 muestra cómo se hace uso de estas funciones para obtener la información del programa visual contenido en el archivo XML.

6.7. Ventajas y desventajas de la herramienta visual

El herramienta visual desarrollada puede ser utilizada por cualquier persona, ya sea por personas que tienen poca experiencia en computación, ya que el uso de lenguaje visual es muy intuitivo y basta con sólo ver las contrucciones visuales para suponer qué hace el programa visual; y por usuarios expertos en computación que requieran hacer análisis más refinados y expecializados ya que es posible definir funciones y agregar fragmentos de código ya que el diccionario de íconos cuenta con representaciones gráficas para poder

Algoritmo 6 Algoritmo para obtener información de la cadena XML y dibujarla en el espacio de trabajo.

Entrada: *cadena*: Representan una estructura XML
Salida: Nada

```
1: xmlDoc ← parseXML(cadena)
2: xml ← $(xmlDoc)
3: programa ← xml.find("programa")
4: si (programa.length > 0) entonces
5:    iconos ← programa.find("iconos")
6:    si (iconos.length > 0) entonces
7:       para todo icono en iconos hacer
8:          nuevoIcono ← newIconos()
9:          para todo atributo en icono.atributos hacer
10:            nuevoIcono.agregarAtributo(atributo)
11:         fin para
12:         agragaEspacioTrabajo(nuevoIcono)
13:      fin para
14:   si no
15:      imprimir "No hay íconos. Nada que hacer"
16:   fin si
17:   conexiones ← programa.find("conexiones")
18:   si (conexiones.length > 0) entonces
19:      para todo conexion en conexiones hacer
20:         creaConexion(conexion)
21:         agragaEspacioTrabajo(conexion)
22:      fin para
23:   si no
24:      imprimir "No hay conexiones. Nada que hacer"
25:   fin si
26: si no
27:    imprimir "No hay programa. Nada que hacer"
28: fin si
```

alojar este código nativo en R, y cuando se ejecute el programa completo, también se ejecuten estos fragmentos de código.

Esta herramienta formará parte de un proyecto más grande por lo que todavia posee limitantes que no son presisamente desventajas, sino son detalles que por cuestiones temporales ya no se llevaron acabo, algunas de ellas son que no se puede cargar archivos del del usuario, si no que las fuentes de datos que se pueden cargar estám directamente almacenadas en la instalación de R como datasets que estan en los paquetes, sólo se puede crear un programa a la vez,

La herramienta ofrece la conexión al sistema gestor de bases de datos PostgreSQL, una vez que se ingresan los datos de la conexión se hace una prueba para conectarse, si están correctos estos datos, se muestran las bases de datos en forma de lista para que mediante un click se seleccione una bases, después de muestran las tablas que tiene dicha base, y se repite el mismo proceso para cada tabla, al darle click se muestran los campos que la forman, esto ayuda poder construir una consulta más fácilmente, o simplemente seleccionando una tabla completamente, esto puede servir como una opción que resuelve el problema de los archivos externos, el usuario puede elegir insertar el contenido del archivo en una tabla, y así poder tener acceso a ellos.

Además de las ventajas que ofrece un entonrno visual y un lenguaje de programación visual, una ventaja importante es que gracias a la interfaz implementada con R, la herramienta visual tiene y con consiguiente los usuarios tienen acceso toda la funcionalidad del sistema R, esto quiere decir que sólo basta con instalar la última vesión de R en el servidor para quedar completamente al día.

Capítulo 7

Ejemplos y Caso de estudio

En este capítulo se muestran algunos ejemplos de programas visuales describiendo el funcionamiento de cada uno de los conos, los ejemplos van a ser obtenidos de [7]y de la documentación de R, además de una aplicación para el SINAC de visualización de algunos de sus datos que se encuentran en su base de datos, a través de consultas. La figura 7.1 muestra la pantalla de trabajo desde el explorador web Chrome, a la derecha se encuentra la caja de herramientas, en la parte superior la barra de menús, a la izquierda el espacio de trabajo dónde se crea el programa visual y en la parte inferior el espacio de resultados.

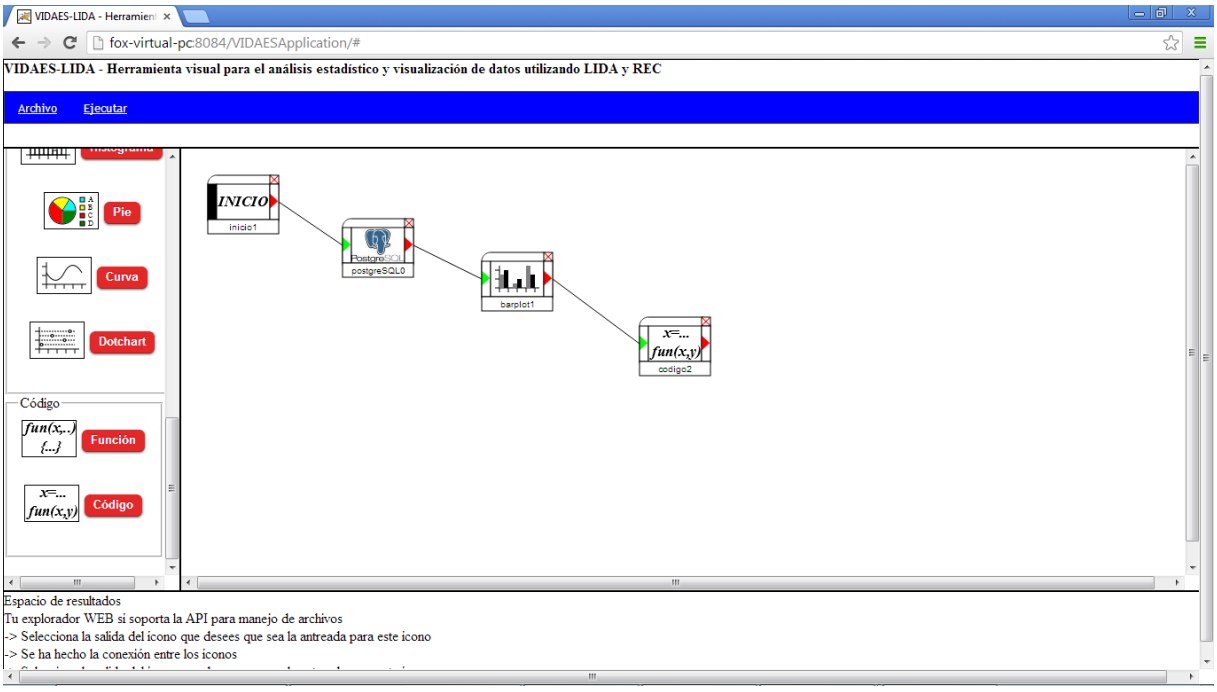


Figura 7.1: Programa de una sesión de trabajo en VIDAES-LIDA

7.1. Programas de ejemplo

En esta sección se mostraran algunas construcciones se programas ejemplificando el uso de los iconos y sus propiedades para poder generar gráficas.

El siguiente ejemplo en la figura 7.2 muestra el uso del ícono "'Plot", en él de declaran un ícono te tipo "Código", en el cual el usuario puede agregar código nativo de R, en este caso se especifica que se desplegarán dos graficas al mismo tiempo, si no se especificase de este forma sólo de desplegará la última gráfica declarada, se declara un ícono de tipo "Arreglo" al cual se le asignará un vector que va del 1 al 50, después de agregan dos íconos de tipo "Plot", y se les especifica como función x este arreglo, y como función y una función de densidad de probabilidad binomial y otro con una función de los cuartiles con una probabilidad de éxito de 1/3 ambas, ver figura 7.3. La fugura 7.4 muestra el resultado

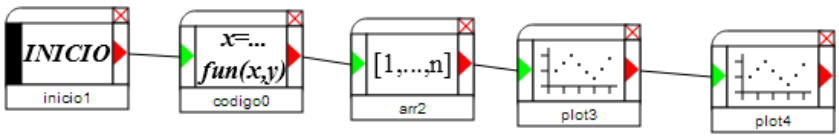


Figura 7.2: Programa visual del uso del ícono plot.

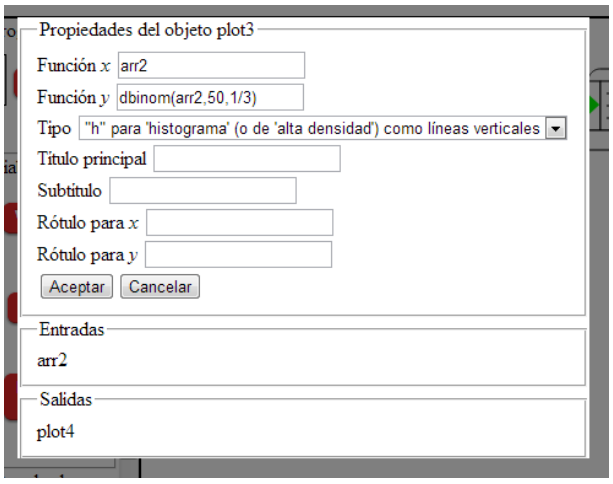


Figura 7.3: Especificación de las propiedades del objeto plot4.

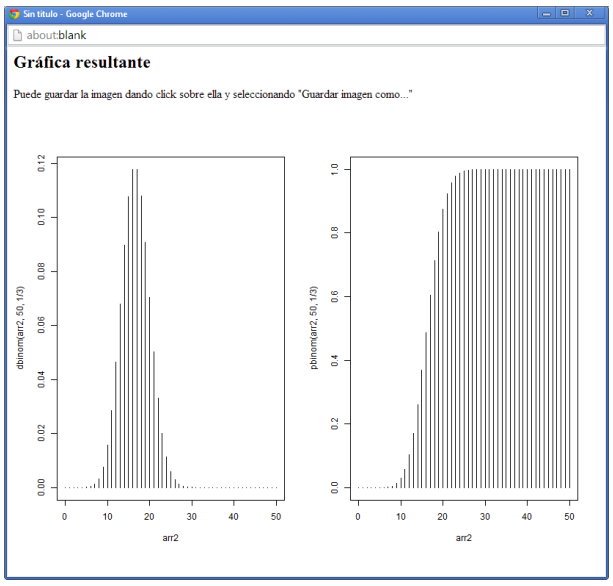


Figura 7.4: Gráfica resultante del programa .

El siguiente programa 7.5 muestra el uso del ícono curva, que recibe como parámetro una función en términos de x, y un intervalo en el que será evaluada, la figura muestra 7.6 muestra las propiedades de los íconos en este programa y en la figura 7.7 se muestra el resultado del programa.

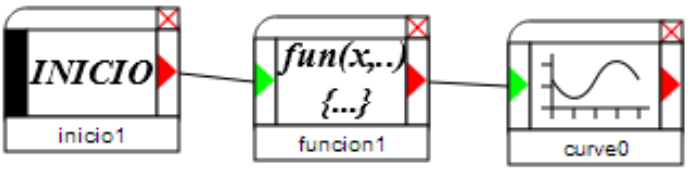


Figura 7.5: Programa visual del uso del ícono curva.

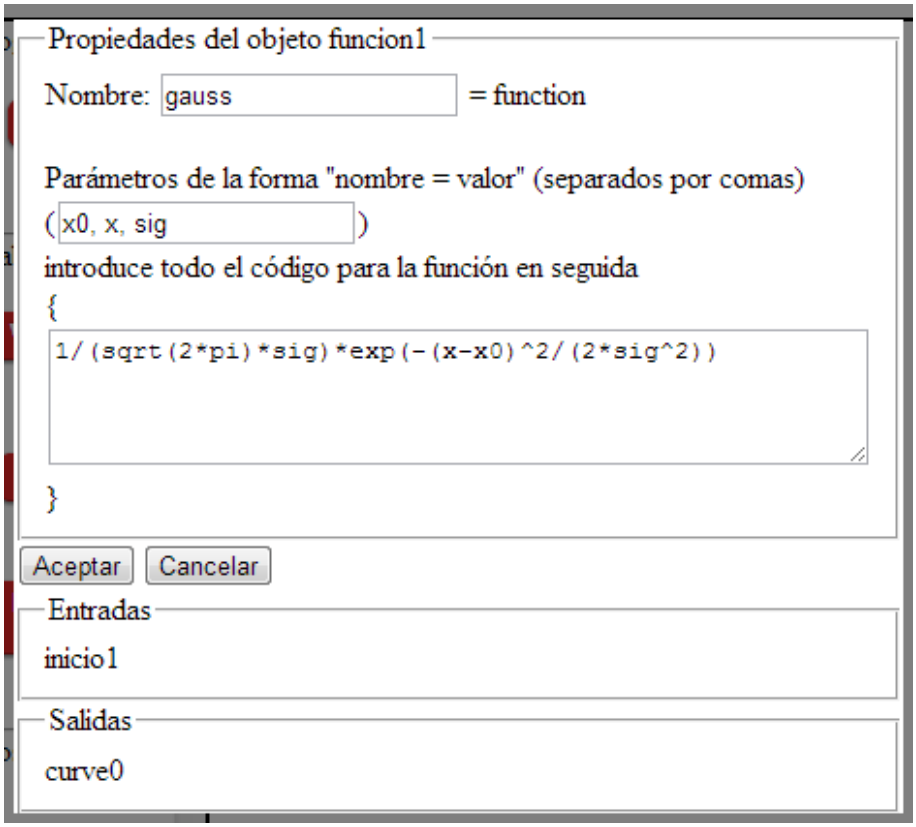


Figura 7.6: Propiedades del ícono función.

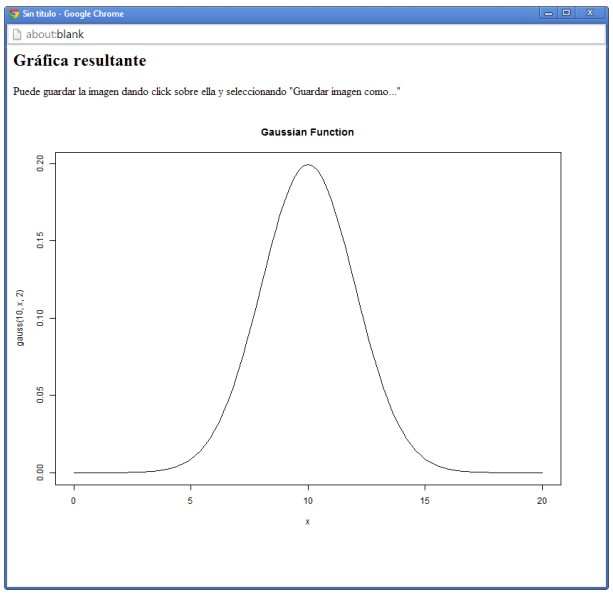


Figura 7.7: Gráfica resultante del programa.

El siguiente ejemplo muestra el uso del ícono dotchart y dataset, la figura 7.8 muestra el programa visual para crear un gráfica de tipo dotchart, el ícono dataset0 representa un conjunto de datos de muestra que se vienen con la instalación del sistema R, el dataset se llama VADeaths y representa las Tasas de mortalidad en Virginia en 1940. La figura 7.9 muestra la gráfica resultante

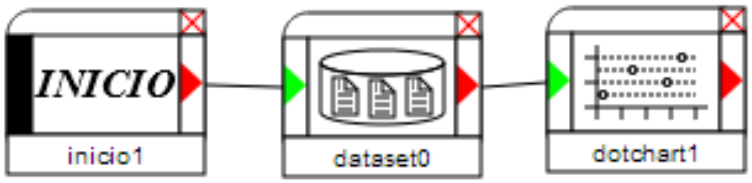
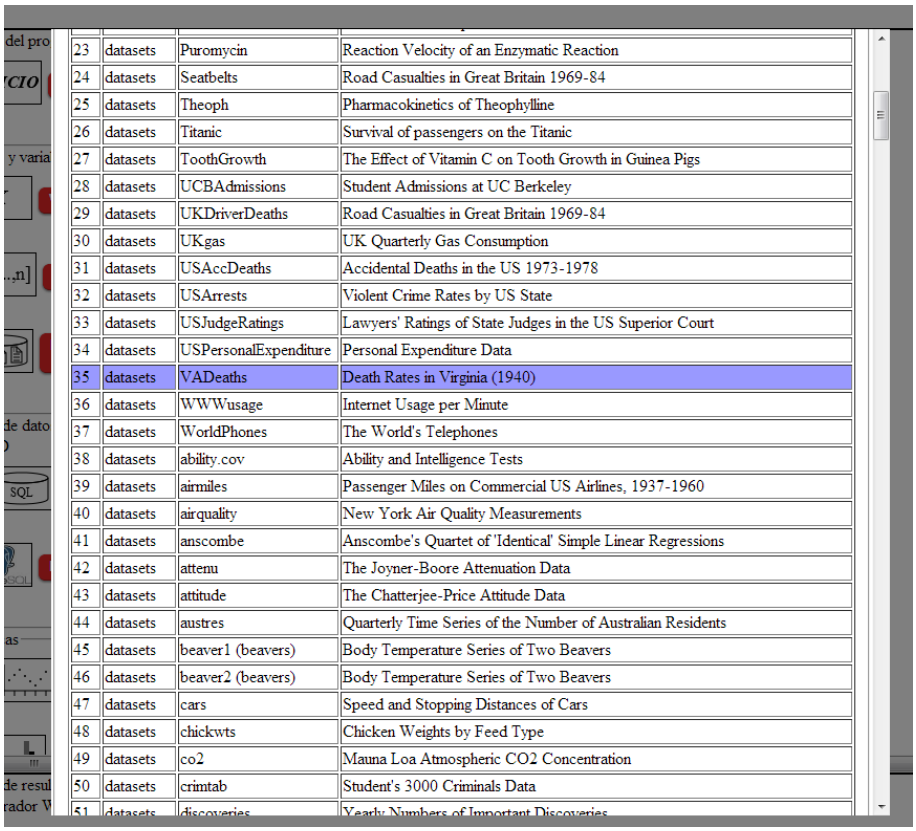


Figura 7.8: Programa visual del uso del ícono dotchart.



23	datasets	Puromycin	Reaction Velocity of an Enzymatic Reaction
24	datasets	Seatbelts	Road Casualties in Great Britain 1969-84
25	datasets	Theoph	Pharmacokinetics of Theophylline
26	datasets	Titanic	Survival of passengers on the Titanic
27	datasets	ToothGrowth	The Effect of Vitamin C on Tooth Growth in Guinea Pigs
28	datasets	UCBAdmissions	Student Admissions at UC Berkeley
29	datasets	UKDriverDeaths	Road Casualties in Great Britain 1969-84
30	datasets	UKgas	UK Quarterly Gas Consumption
31	datasets	USAccDeaths	Accidental Deaths in the US 1973-1978
32	datasets	USArrests	Violent Crime Rates by US State
33	datasets	USJudgeRatings	Lawyers' Ratings of State Judges in the US Superior Court
34	datasets	USPersonalExpenditure	Personal Expenditure Data
35	datasets	VADeaths	Death Rates in Virginia (1940)
36	datasets	WWWusage	Internet Usage per Minute
37	datasets	WorldPhones	The World's Telephones
38	datasets	ability.cov	Ability and Intelligence Tests
39	datasets	airmiles	Passenger Miles on Commercial US Airlines, 1937-1960
40	datasets	airquality	New York Air Quality Measurements
41	datasets	anscombe	Anscombe's Quartet of 'Identical' Simple Linear Regressions
42	datasets	attenu	The Joyner-Boore Attenuation Data
43	datasets	attitude	The Chatterjee-Price Attitude Data
44	datasets	austres	Quarterly Time Series of the Number of Australian Residents
45	datasets	beaver1 (beavers)	Body Temperature Series of Two Beavers
46	datasets	beaver2 (beavers)	Body Temperature Series of Two Beavers
47	datasets	cars	Speed and Stopping Distances of Cars
48	datasets	chickwts	Chicken Weights by Feed Type
49	datasets	co2	Mauna Loa Atmospheric CO2 Concentration
50	datasets	crimtab	Student's 3000 Criminals Data
51	datasets	discoveries	Yearly Numbers of Important Discoveries

Figura 7.9: Propiedades del ícono dataset0 mostrando todos los conjuntos de datos disponibles y seleccionando VADeaths.

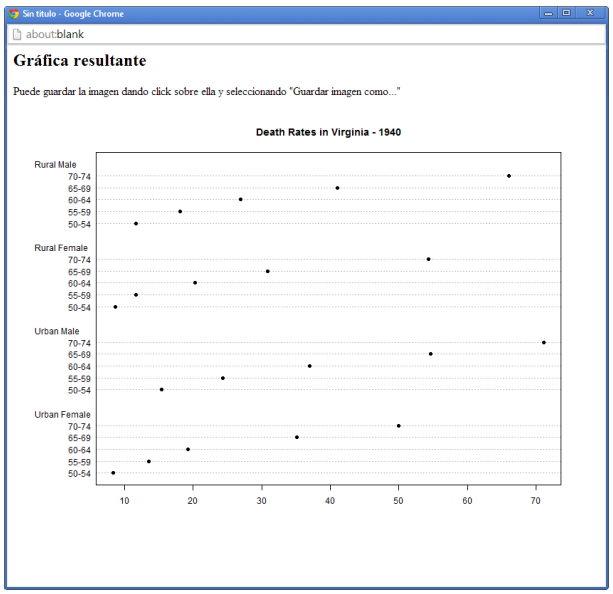


Figura 7.10: Gráfica resultante del programa.

7.2. Ejemplos del caso de estudio

En esta sección se muestra una aplicación de la herramienta para visualizar los datos del sistema SINAC mediante consultas a estos, los datos están alojados en el servidor de Postgresql, el modelo relacional tiene 321 tablas y 8 vistas sobre estos datos que ayudan a facilitar las consultas.

El siguiente ejemplo en la figura 7.11 muestra el grado académico de los investigadores que trabajan en el CINVESTAV, la figura 7.12 muestra las propiedades del objeto postgreSQL0 para conectar con el sistema gestor y la consulta en SQL. La figura 7.13 muestra el resultado de la consulta anterior.

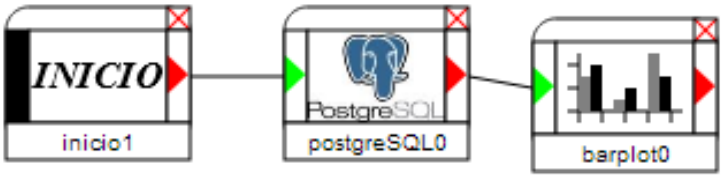


Figura 7.11: Programa visual para mostrar el .

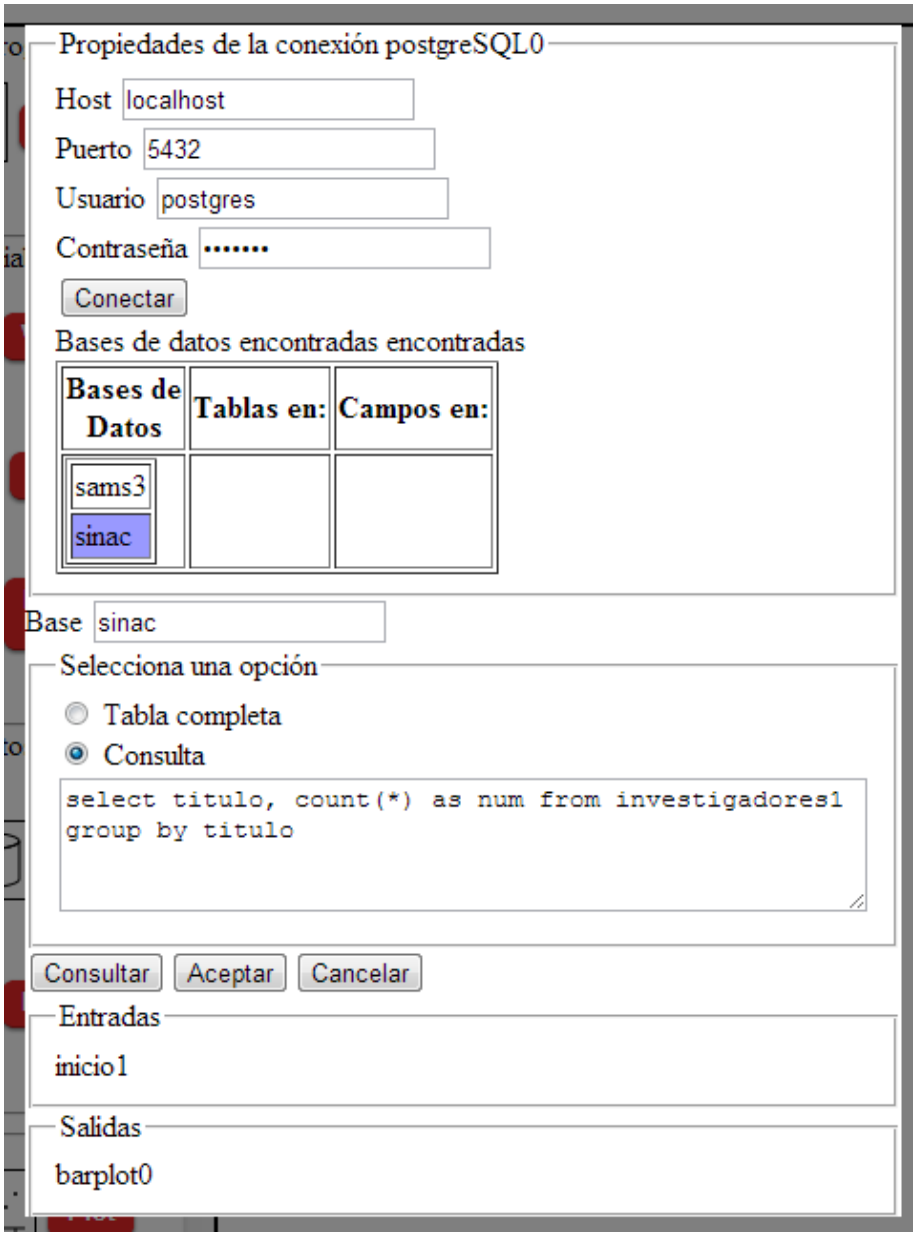


Figura 7.12: Propiedades del ícono postgreSQL0 mostrando los datos de la conexión y bases de datos disponibles, la base sinac seleccionada se resalta en azul.

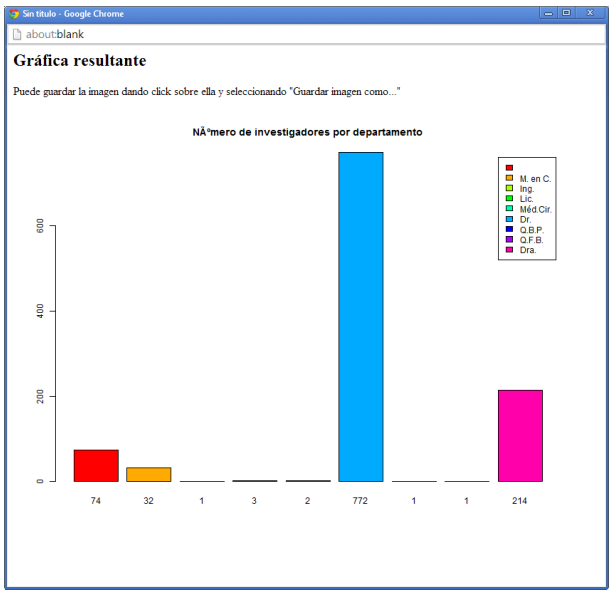


Figura 7.13: Gráfica resultante del programa.

El siguiente programa visual en la figura 7.14 muestra cuántos investigadores trabajan en cada departamento en el CINVESTAV. La figura refproppost2 muestra la cosnsulta en SQL y la figura 7.16.

Figura 7.14: Programa visual para mostrar el número de investigadores por departamento.

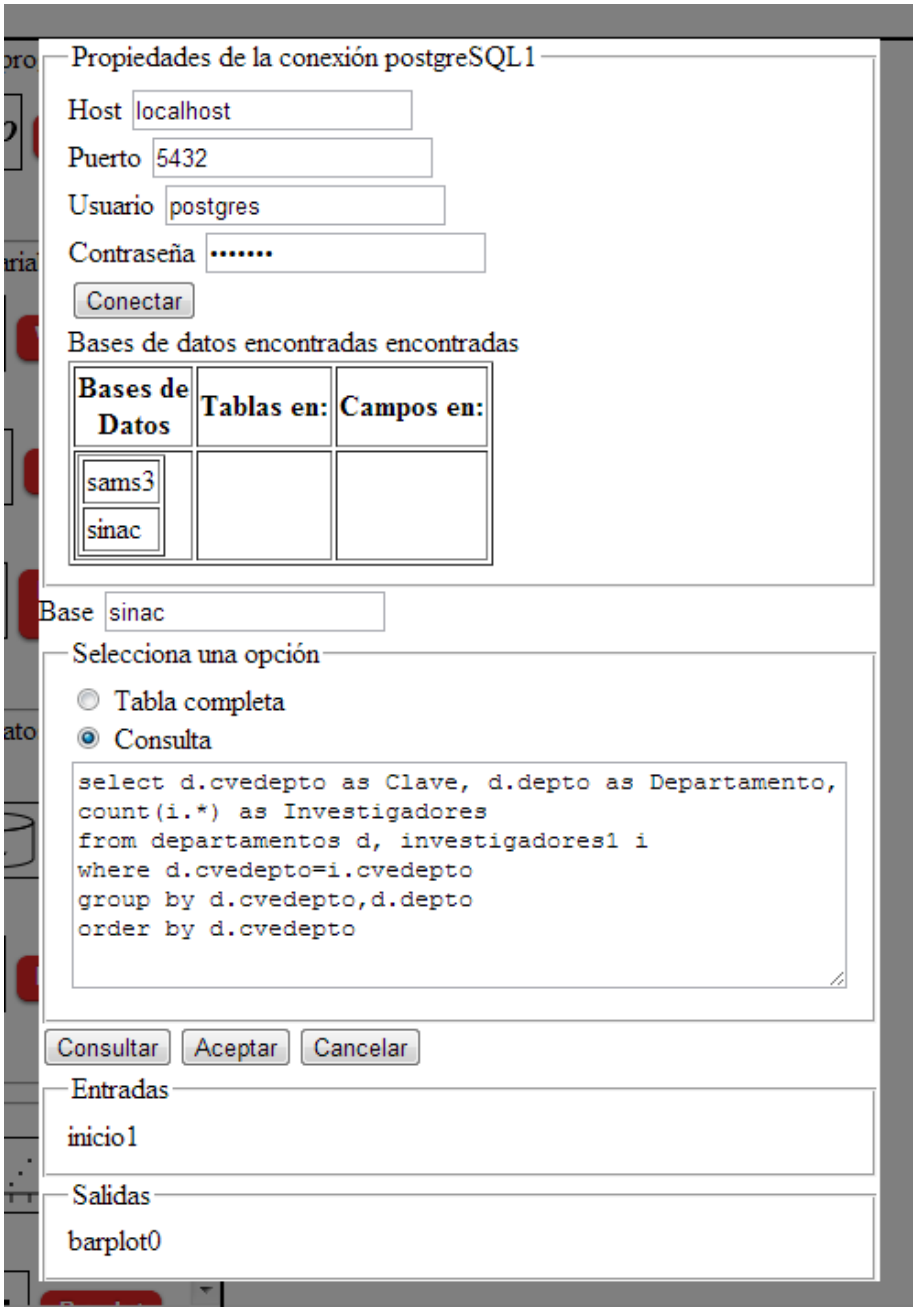


Figura 7.15: Propiedades del ícono postgreSQL1 mostrando los datos de la conexión, bases de datos disponibles y la consulta en SQL para obtener la información

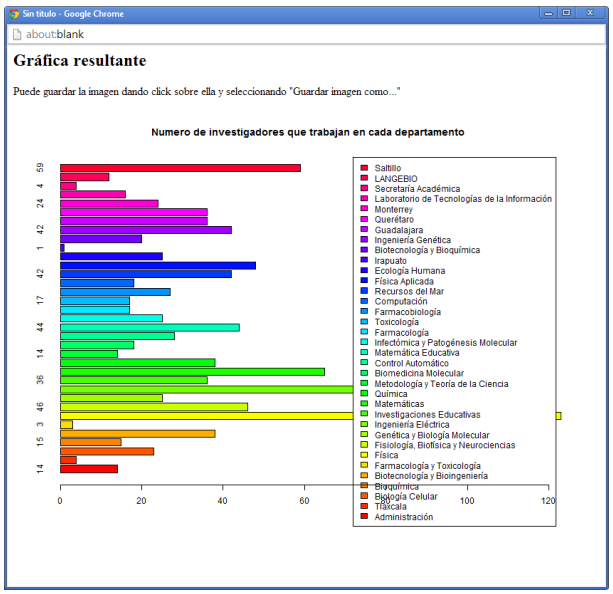


Figura 7.16: Gráfica resultante del programa.

El siguiente programa visual en la figura 7.22 muestra el comportamiento a través de los años de la obtención del nivel 3 de los investigadores del CINVESTAV de SNI. La figura refproppost3 muestra la consulta en SQL y la figura 7.19.

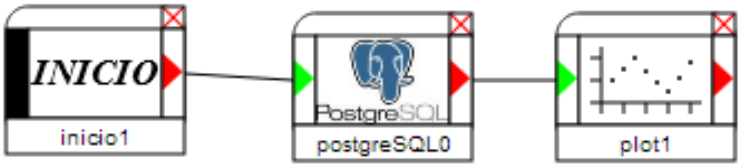


Figura 7.17: Programa visual para mostrar el número de investigadores que obtuvieron el nivel 3 en el SNI.

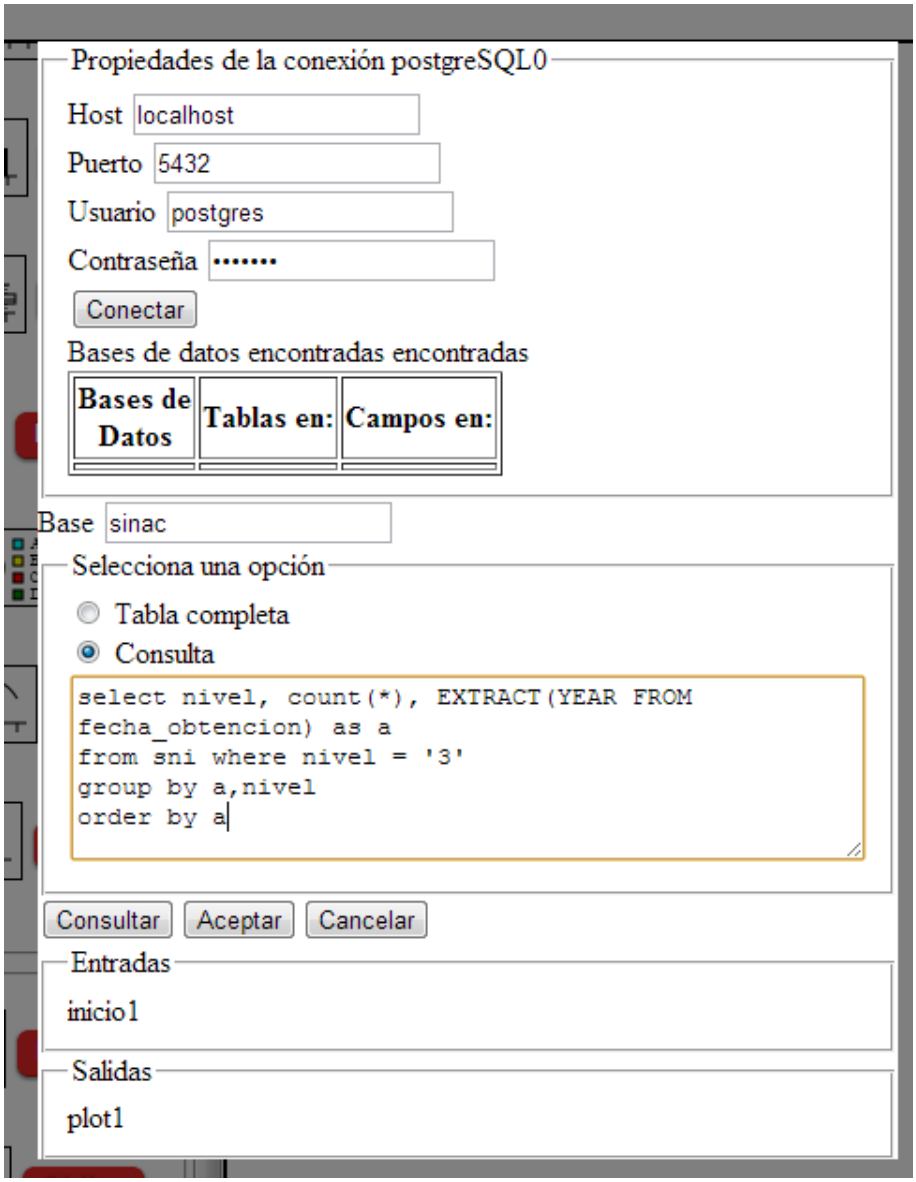


Figura 7.18: Propiedades del ícono postgreSQL1 mostrando los datos de la conexión, bases de datos disponibles y la consulta en SQL para obtener la información

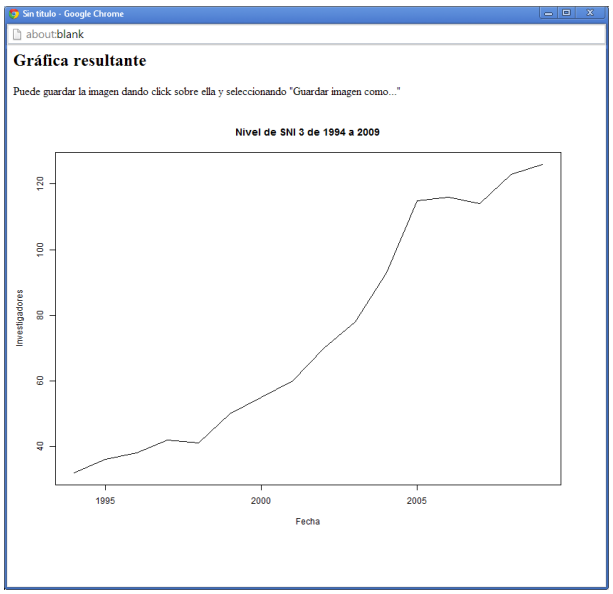


Figura 7.19: Gráfica resultante del programa.

Los programas pueden llegar a ser tan complejos como el usuario así los requiera, estos ejemplos sólo son para mostrar el uso de la herramienta. El usuario tiene una puerta hacia toda la funcionalidad de R.

7.3. Comparación con Matlab

Actualmente Matlab ha sido ampliamente usado para el análisis de datos, principalmente para matrices ya que ofrece mecanismos para manipular más fácilmente. En el departamente de computación, ente sonftware se utliza en el área de Optimización. Un compañero del departamente facilitó algunos ejemplos que realiza en Matlab para poder implementarlos con la herramienta y comparar cómo son los resultados.

El siguiente ejemplo sirve para seleccionar una muestra de un frente de pareto lineal de 300 puntos, dada la propiedad de convexidad de la norma para cada nuevo punto se calcula de la siguiente forma, se divide cada entrada del punto del frente entre la norma p y con esto se obtiene un frente cóncavo para valores mayores a 1 hasta infinito y un frente convexo para valores menores de 1 hasta 0. La siguiente figura muestra el programa que resuleve el ejemplor anterior.

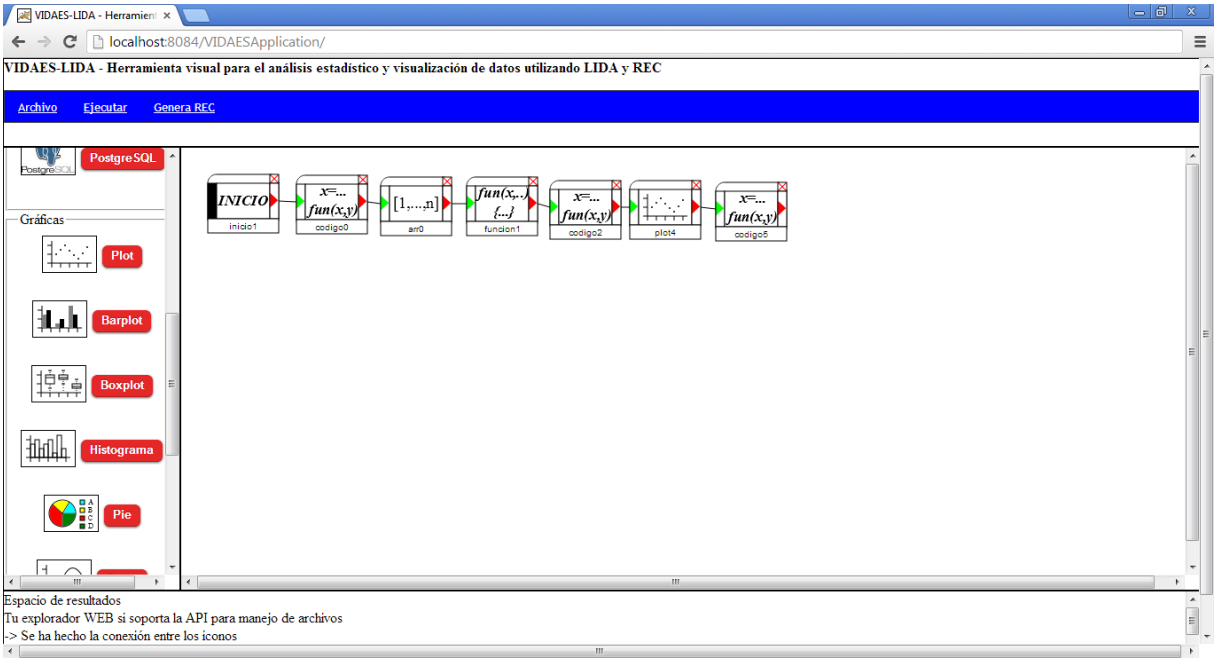


Figura 7.20: Programa visual para seleccionar una muestra de un frente de pareto, en este caso es convexa.

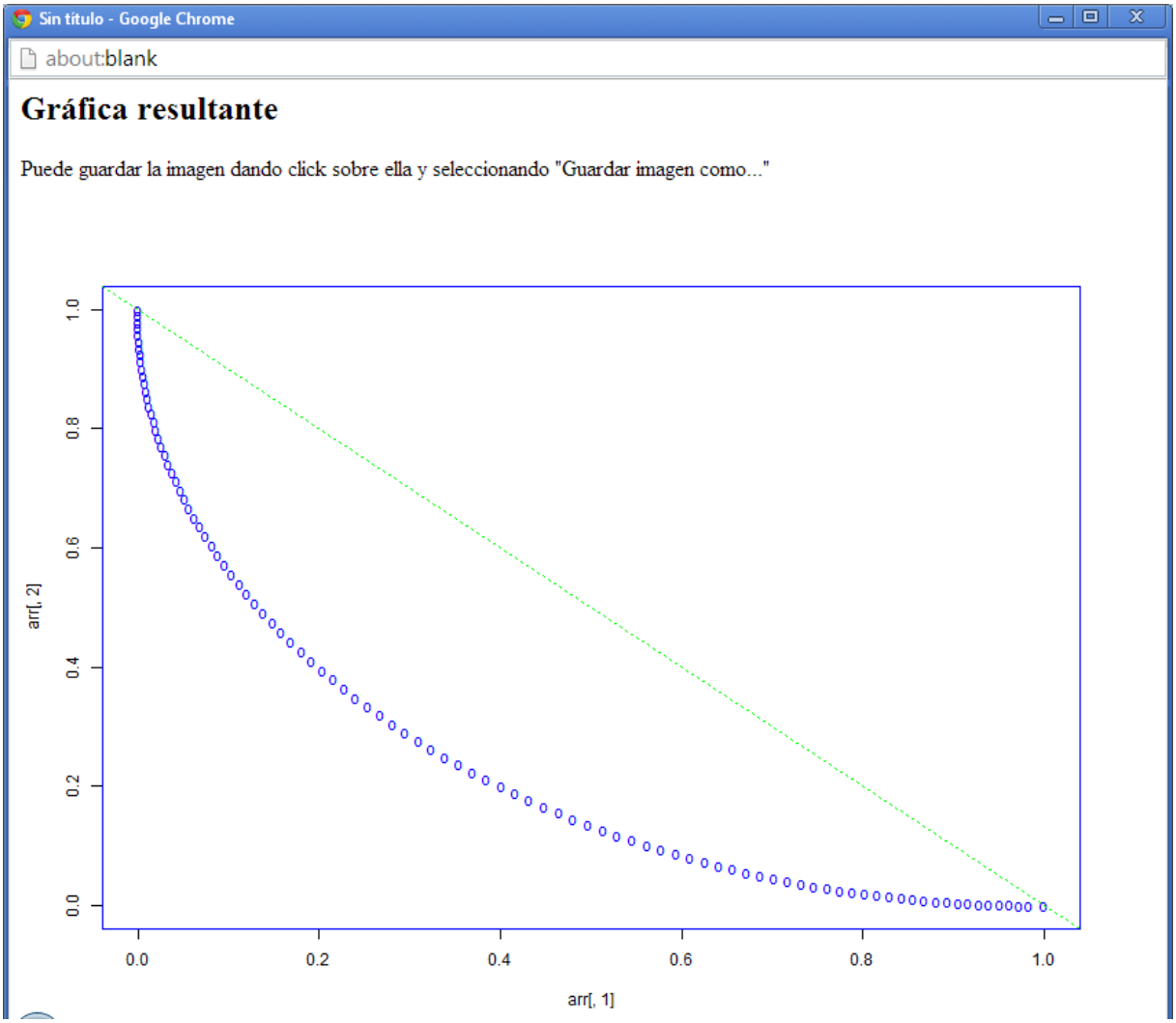


Figura 7.21: Gráfica resultante de la ejecución con la opción convex.

El siguiente ejemplo es una comparación del performance de un algoritmo evolutivo basado en el hipervolumen y de su versión memética que consiste en integrar una estrategia de búsqueda local. Lo que se busca es mostrar que el enfoque memético obtiene mejores resultados que el enfoque clásico, este experimento se hizo en 3 funciones diferentes y está hecho para 20 ejecuciones. Las siguientes figuras muestran el ejemplo anterior

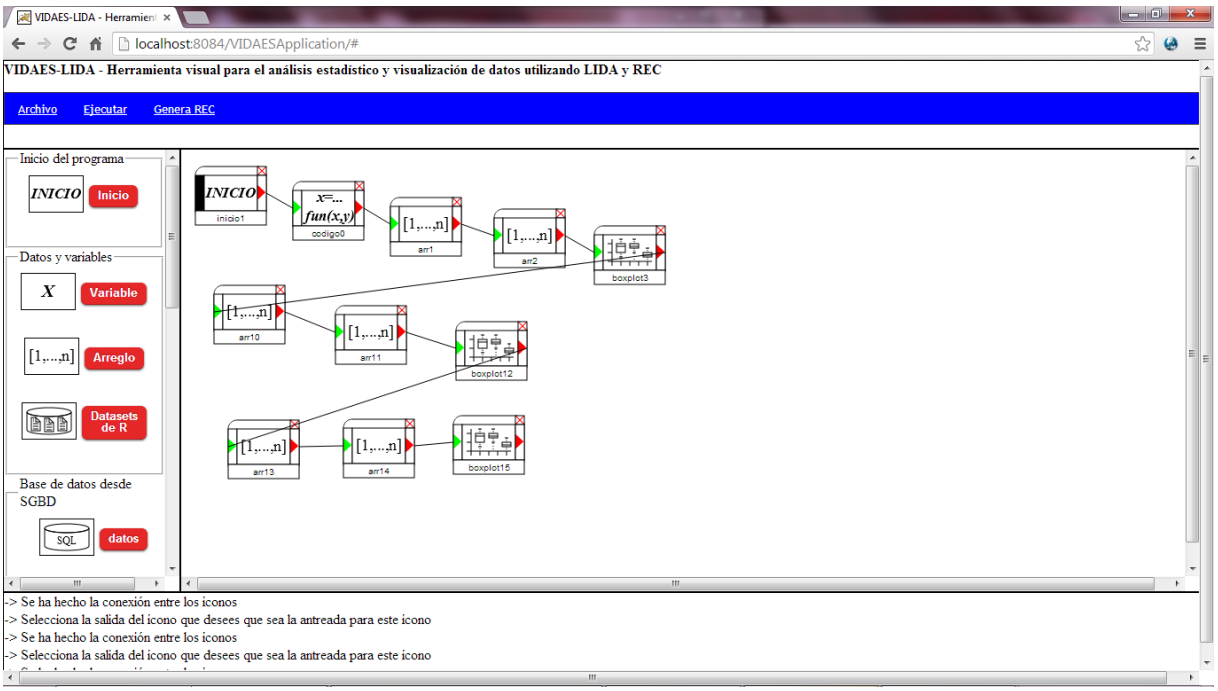


Figura 7.22: Programa visual para el desempeño de dos algoritmos.

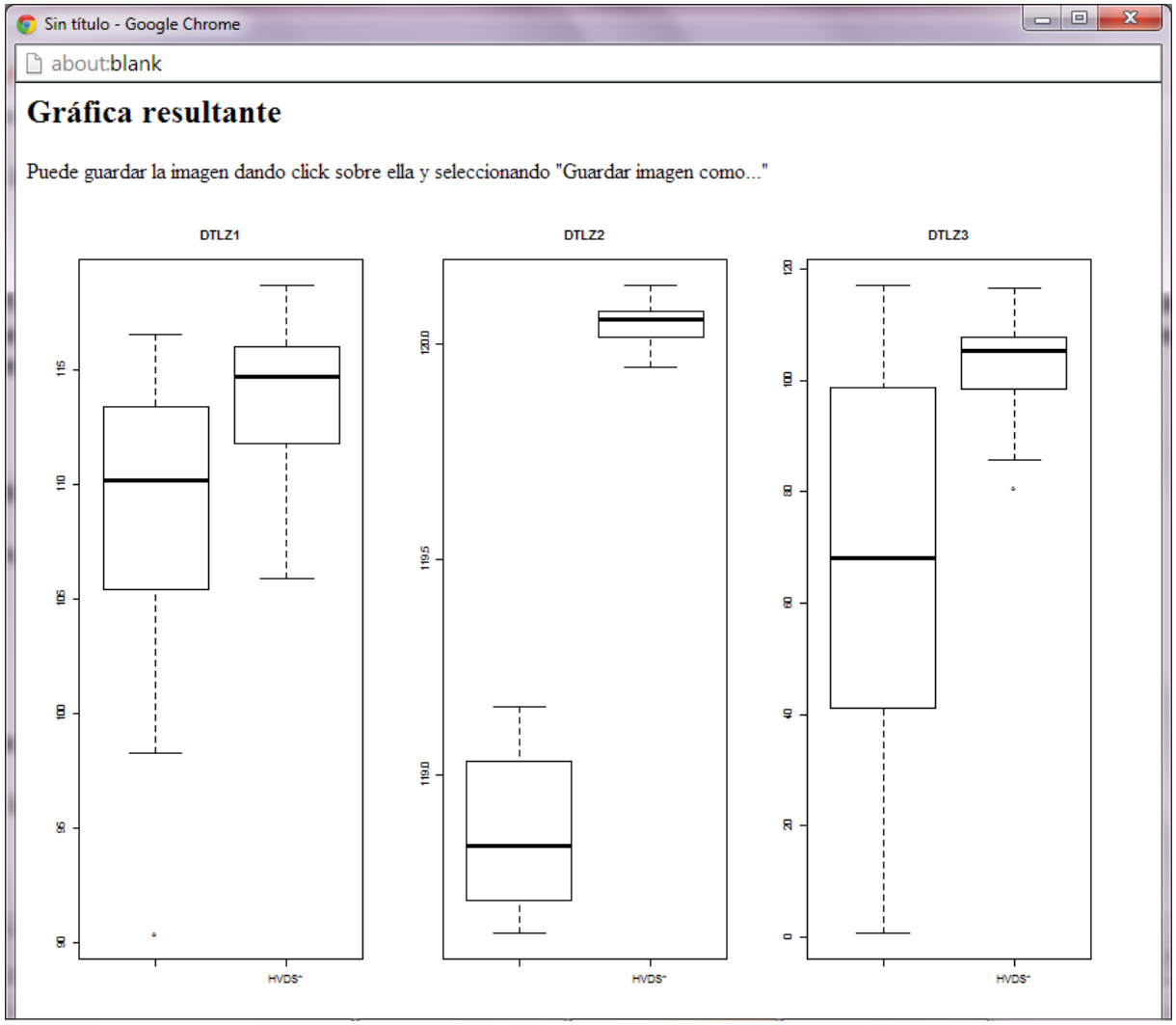


Figura 7.23: Gráfica resultante de la ejecución del programa, uno básado en el hipoervolumen (lado izquierdo de la gráfica) y el segundo integrando una busqueda local (lado derecho de la gráfica).

Capítulo 8

Resultados, conclusiones y trabajo a futuro

En este capítulo de muestran los resultados que se obtuvieron con el desarrollo de esta tesis, se discuten algunos puntos importantes para poder a las conclusiones importantes sobre la implementación de esta herramienta visual, y por último se muestra el trabajo a furuto que se puede hacer teniendo como base la herramienta visual documentada en esta tesis.

8.1. Resultados

- Una herramienta visual para el análisis estadístico y visualización de datos implementada en HTML5 y JavaScript.
- Un compilador de lenguaje intermedio REC implementado en Java agregando las siguiente características:
  - declaración de variables,
  - nombres de rutinas de más de una carácter de longitud,
  - número de parámetros ilimitado al llamar las rutinas.
- Una interfaz más robusta entre R y en lenguaje Java.

8.2. Discusión y conlusiones

La interfaz de usuario anterior a este proyecto de tesis se encuentra en su totalidad en Java y se mostraba en una página web mediante la etiqueta $< applet >$, y esto requiere complementos adicionales para poder ser ejecutado, además no se soporta ningún evento disponible en HTML, y lo más importante es que esta etiqueta $< applet >$ esta descontinuada en la versión 4.01 y ya no es soportada en la versión 5 de HTML, por lo

tanto continuar la evolución de LIDA por este camino ya no es productivo ni prometedor. En su lugar de implementó utlizando completamente HTML 5 y JavaScript, aprovechando las nuevas etiquetas que ofrece para poder dibujar gráficos vectoriales y la inteacción que se le pueden agregar, lo que nos da como resultado una herramienta más ligera para ejecutar en la parte del cliente.

Uno de las razones primordiales de proyecto es poder guardar el proceso de análisis del usuario y porder ejecutarlo las veces que lo necesite, entonces se necesita un formato para poder almacenar estos procesos. Hubieron varias opciones como utilizar formatos ya existente para almacenar datos o crear uno propio. Por cuestiones de tiempo se decidió utilizar uno ya existente, entonces se encontraron dos, XML y JSON, de entre estos dos se escogió XML ya que se puede validar la estructura interna de los elementos que lo contienen, con JSON no.

También se tuvieron bastantes complicaciones en cuanto a la codificación de caracteres, ya que a pesar de que se mostró en el caítulo 6 de que estaban en el mismo conjunto de caracteres PostgreSQL y el cliente en R, no se mostraban correctamente, por lo tanto se tuvo que hacer una función para poder codificar correctamente estos caracteres y poder agregarlos como leyendas o equietas a las gráficas de las cosultas.

Las tareas de visualización y análisis de datos corren por cuenta del lenguaje R, lo que hace necesario realizar algún tipo de conexión o interfaz entre la herramienta visual y el lenguaje R. Se encontró en un ejemplo secillo de interfaz entre R a la cual se le fueron haciendo mejoras durante el desarrallo de esta herramienta para hacerlo más robusto. Esta interfaz esta implementa en Java, por una parte utlizando las clases que ofrece para la manipulación de procesos, y por otro lado la instalación de R viene que dos archivos ejecutables (RScript.exe y R.exe) los cuales procesan código nativo, por eso se continuó el desarrollo de esta interfaz.

La versión de REC originalmente se encuentra implementado en el lenguaje C, así no era de utilidad, ya que la interfaz con R, como se mencionó arriba, se encuentra implementada en Java, por lo tanto se tuvo que implementar una nueva versión de REC en Java, agregando ciertas características para lograr la intergración de este lenguaje REC. Además también se conluye que se puede prescindir de este lenguaje ya que es sencillo pero R a su vez también es muy sencillo de usar, estonces no hay una razón lo suficientemente fuerte para continuar usándolo en futuras versiones del a herramienta.

Se realizó una comparación con Matlab realizando los mismos problemas implementados su lenguaje m y en la herramienta implementada, el análisis y las gráficas resultantes fueron similares, entonces el uso de esta herramienta puede ser una opción alternativa a Matlab.

Por lo tanto, el desarrollo de una herramienta visual para resolver el problema de visualización de datos es una buena solución.

8.3. Trabajo a futuro

- Trabajar en mejorar los estilos de la vista de la aplicación.

- Incorporar nuevas fuentes de datos de entrada al sistema, como pueden ser desde otros sistemas gestores de bases de datos a parte de PostgreSQL.
- Integrar la nueva funcionalidad que se vaya incorporando a R, ya que se encuentra en constante evolución y actulización, a la herramienta visual desarrollada.
- Agregar el manejo de multiarchivos en una sesión de trabajo, para mantener multiples programas visuales al mismo tiempo.
- Dar la posibilidad que el usuario pueda envia archivos desde la herramienta al servidor para que se puedan ser leídos desde el sistema R, ya que como la herramienta se ejecuta del lado del cliente y R se ejecuta en el servidor no se puede tener acceso a los archivos locales de forma directa en el servidor.
- Manejar cuentas de usuario para que el usuario pueda manejar sus archivos cargados, y pueda eliminarlos o cargar más, a fin de que los tenga disponibles al momento de usar la herramienta.

Bibliografía

[1] H. Dreyfuss. *Symbol Sourcebook: An Authoritative Guide to International Graphic Symbols.* -. McGraw-Hill Companies, 1972.

[2] S.-K. Chang and S. Levialdi. Journal of Visual Languages and Computing - Elsevier. `http://www.journals.elsevier.com/journal-of-visual-languages-and-computing/`, febrero 2013.

[3] N. Sierra Hernández. HEVICOP: HErramienta VIsual para la COnstrucción de Programas. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Mayo 1995.

[4] S. G. Ju. *Visualización de una Dase de Datos Espacial.* Tesis de doctorado, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, 1996.

[5] S. V. Chapa Vergara. *Programación Automática a través de descriptores de Flujo de Información.* Tesis de doctorado, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Marzo 1991.

[6] S.K. Chang. *Visual languages and visual programming.* Languages and information systems. Plenum Press, 1990.

[7] V. Bloomfield. *Computer Simulation and Data Analysis in Molecular Biology and Biophysics: An Introduction Using R.* Biological and Medical Physics, Biomedical Engineering. Springer, 2009.

[8] M. L. Berenson, D. M. Levine, and T. C. Krehbiel. *Estadística para administración.* Pearson Educación, 2006.

[9] G. Cisneros. Configurable rec. *SIGPLAN Not.*, 29(5):7–16, May 1994.

[10] H. V. McIntosh and G. Cisneros. The programming languages rec and convert. *SIGPLAN Not.*, 25(7):81–94, July 1990.

[11] P. Alday Echeverría. Diseño de Base de Datos con EVEX Entidad Vínculo Extendido para xwindows. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Febrero 1997.

[12] R. Hernández Stefanori. Sistema para el Diseño de Base de Datos: EVE. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Marzo 1994.

[13] J. Martinez Cruz and S. V. Chapa Vergara. Informe Final: Proyecto Colección de datos Cultivos Microbianos. Bases de Datos, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Abril 1996.

[14] N. Murray, N. Paton, and C. Goble. Kaleidoquery: a visual query language for object databases. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '98, pages 247–257, New York, NY, USA, 1998. ACM.

[15] C. Bonhomme, C. Trépied, M. A. Aufaure, and R. Laurini. A visual language for querying spatio-temporal databases. In *Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, GIS '99, pages 34–39, New York, NY, USA, 1999. ACM.

[16] S. Hibino and E. A. Rundensteiner. User interface evaluation of a direct manipulation temporal visual query language. In *Proceedings of the fifth ACM international conference on Multimedia*, MULTIMEDIA '97, pages 99–107, New York, NY, USA, 1997. ACM.

[17] L. Paolino, G. Tortora, M. Sebillo, G. Vitiello, and R. Laurini. Phenomena: a visual query language for continuous fields. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, GIS '03, pages 147–153, New York, NY, USA, 2003. ACM.

[18] A. Hernández Montoya. LIDA/REC Lenguaje Visual para Bases de Datos. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Agosto 2005.

[19] J. Lin, M. Thomsen, and J. A. Landay. A visual language for sketching large and complex interactive designs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, pages 307–314, New York, NY, USA, 2002. ACM.

[20] F. Goetz, R. Borau, and G. Domik. An xml-based visual shading language for vertex and fragment shaders. In *Proceedings of the ninth international conference on 3D Web technology*, Web3D '04, pages 87–97, New York, NY, USA, 2004. ACM.

[21] N. Liu, J. Grundy, and J. Hosking. A visual language and environment for composing web services. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 321–324, New York, NY, USA, 2005. ACM.

[22] Katherine Howland, Judith Good, and Judy Robertson. A learner-centred design approach to developing a visual language for interactive storytelling. In *Proceedings of the 6th international conference on Interaction design and children*, IDC '07, pages 45–52, New York, NY, USA, 2007. ACM.

[23] S. Gauvin and O. Banyasad. Transparency, holophrasting, and automatic layout applied to control structures for visual dataflow programming languages. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 67–75, New York, NY, USA, 2006. ACM.

[24] N. Liu, J. Hosking, and J. Grundy. A visual language and environment for specifying user interface event handling in design tools. In *Proceedings of the eight Australasian conference on User interface - Volume 64*, AUIC '07, pages 87–94, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[25] Na Liu. Visual languages for event integration specification. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 969–972, New York, NY, USA, 2006. ACM.

[26] P. Díaz, I. Aedo, M. B. Rosson, and J. M. Carroll. A visual tool for using design patterns as pattern languages. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, pages 67–74, New York, NY, USA, 2010. ACM.

[27] G. Tekli, R. Chbeir, and J. Fayolle. Xcdl: an xml-oriented visual composition definition language. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '10, pages 134–143, New York, NY, USA, 2010. ACM.

[28] L. Méndez Segundo. DALI: Herramienta para la representación de gráfica de información. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, Diciembre 1998.

[29] D. Rojas Davis. Contrucción de un Editor de Gráficas de Propósito General. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Electrica, Sección de Computación, 1999.

[30] G. Alor Hernández. Sistema de Consulta Estadística con un Ambiente Gráfico utilizando Tecnología Dinámica. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Computación, Agosto 2001.

[31] L. Wu, M. Li, Z. Li, W. Y. Ma, and N. Yu. Visual language modeling for image classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, MIR '07, pages 115–124, New York, NY, USA, 2007. ACM.

[32] C. T. Johnston, P. Lyons, and D. G. Bailey. User evaluation and overview of a visual language for real time image processing on fpgas. In *Proceedings of the 10th International Conference NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction*, CHINZ '09, pages 85–92, New York, NY, USA, 2009. ACM.

[33] Chunxi Liu, Qingming Huang, Shuqiang Jiang, and Changsheng Xu. The third eye: mining the visual cognition across multi-language communities. In *Proceedings of the international conference on Multimedia*, MM '10, pages 431–440, New York, NY, USA, 2010. ACM.

[34] G. Costagliola, V. Deufemia, F. Ferrucci, and C. Gravino. Using extended positional grammars to develop visual modeling languages. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, SEKE '02, pages 201–208, New York, NY, USA, 2002. ACM.

[35] P. T. Cox and S. Gauvin. Controlled dataflow visual programming languages. In *Proceedings of the 2011 Visual Information Communication - International Symposium*, VINCI '11, pages 9:1–9:10, New York, NY, USA, 2011. ACM.

[36] S. Fitrianie and L. J. M. Rothkrantz. A grammar-free visual language-based communication. In *Proceedings of the 13th International Conference on Computer Systems and Technologies*, CompSysTech '12, pages 95–102, New York, NY, USA, 2012. ACM.

[37] N. C. Shu. *Visual programming*. Van Nostrand Reinhold, 1988.

[38] S.K. Chang, T. Ichikawa, and P.A. Ligomenides. *Visual languages*. Management and information systems. Plenum Press, 1986.

[39] Stephen Few. *Data Visualization for Human Perception*. The Interaction Design Foundation, Aarhus, Denmark, 2013.

[40] W3C. World Wide Web Consortium (W3C). `http://www.w3.org/`, febrero 2013.

[41] CERN. CERN - the European Organization for Nuclear Research. `http://www.cern.ch/`, febrero 2013.

[42] W3C. Tags used in html. `http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html`, febrero 2013.

[43] IETF. Internet Engineering Task Force (IETF).htm. `http://www.ietf.org/`, febrero 2013.

[44] Apple, Mozilla, and Opera. Web Hypertext Application Technology Working Group. `http://www.whatwg.org/`, febrero 2013.

[45] W3C. Html5. `http://www.w3.org/TR/html5/`, marzo 2013.

[46] W3C. W3C relaunches html activity. http://www.w3.org/2007/03/html-pressrelease/, marzo 2013.

[47] W3C. HTML 4.01 specification. http://www.w3.org/TR/html401/, febrero 2013.

[48] W3C. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). http://www.w3.org/TR/xhtml1/, febrero 2013.

[49] W3C. Introducción a XHTML. http://www.w3schools.com/html/html_xhtml.asp, marzo 2013.

[50] J. Franganillo. Html5: el nuevo estándar básico de la web. *Anuario ThinkEPI*, 5:261–265, September 2010.

[51] R Core Team. The Comprehensive R Archive Network. http://cran.r-project.org/, febrero 2013.

[52] Dmitry Baranovskiy. Raphaël-javascript library. "http://raphaeljs.com/, febrero 2013.

[53] W3C. DTD Tutorial. http://www.w3schools.com/dtd/, marzo 2013.

[54] E. Bidelman. Cómo leer archivos locales en JavaScript - HTML5 Rocks. http://www.html5rocks.com/es/tutorials/file/dndfiles/, marzo 2013.