UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DIAGRAMACIÓN DE REDES DE PETRI CON CONTROL LOCAL DE VERSIONES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SEBASTIÁN FERNANDO RODRÍGUEZ BUSTAMANTE

PROFESOR GUÍA
JOSE ALBERTO PINO URTUBIA

MIEMBROS DE LA COMISIÓN
CLAUDIO GUTIERREZ GALLARDO
RODRIGO ANDRES ARENAS ANDRADE

SANTIAGO DE CHILE
ENERO 2008

RESUMEN

Hoy en día son escasas las herramientas en Internet que permiten crear y compartir diagramas en línea. Por eso, esta es una buena oportunidad de construir un sitio web que tenga dicha funcionalidad, ya que a pesar de las dificultades que se puedan presentar promete una serie de ventajas, por ejemplo ser accesible desde cualquier PC conectado a Internet y poder colaborar con otros usuarios alrededor del mundo.

En esta memoria se implementó un sitio web que permite realizar diagramas de redes de Petri en línea, grabarlos, almacenar distintas versiones, exportarlos a formato PNG y compartir documentos con otros usuarios. Además, se intentó proveer de herramientas que permitan la colaboración asíncrona de varios usuarios sobre un mismo documento. Para ello utilizamos *Stick-Ons* aplicados a diagramas (lo cual permite tener un control local de versiones). También se implementó una funcionalidad poco común: es posible “entrar” a un elemento del diagrama y obtener un sub diagrama, por ejemplo para tener mayor detalle. Esta funcionalidad se usó para poder representar sub-procesos en las redes de Petri.

En el trabajo realizado fueron revisadas las posibilidades gráficas que tiene el browser, con que herramienta podemos implementar los servicios necesarios y que base de datos vamos a utilizar. Se eligió *Flash* (*Flex*) para el desarrollo de la interfaz, *PHP* para los servicios web y *MySql* como base de datos. Posteriormente diseñamos de qué manera deberíamos aplicar los Stick-Ons en el caso de diagramas, en donde proponemos la forma en que se pegarán y el *feedback* visual que tendremos al despegarse el Stick-On. Finalmente se implementa el sitio web y el editor de diagramas, incorporando los Stick-Ons como parte de su funcionalidad.

Los resultados finales se ilustran creando algunas redes de Petri tomadas del libro "*Workflow Managment*" en el sitio web. Además se muestra un ejemplo del uso de Stick-Ons hecho también dentro de la aplicación creada. Concluimos que los Stick-Ons son una herramienta adecuada para llevar un control de cambios local dentro de un diagrama y que si se mantiene dentro de rangos normales de operación, el proyectar una sombra cuando se despega un Stick-On da una buena indicación visual del elemento despegado.

"*no es difícil hacer lo correcto, lo difícil es saber qué es lo correcto*"

En el camino recorrido muchas personas me ayudaron para realizar esta memoria: el personal administrativo de la U, me ayudaron en todos los tramites que tuve que realizar para esta segunda oportunidad, mi profesor guía que siempre estuvo cuando lo necesité, mis compañeros de trabajo que supieron dejarme tiempo libre y aceptaron mis ausencias, a todos, gracias totales.

… y especialmente a Hernán, mi padre, a Patricia, mi madre, y a Karina, mi polola, gracias, jamás lo habría logrado sin su constante apoyo y comprensión.

Un siete para todos ustedes en mi libro de notas.

Índice

1. INTRODUCCIÓN Y MOTIVACIÓN 6

INTRODUCCIÓN 6
JUSTIFICACIÓN 7
OBJETIVOS 9
Objetivo General 9
Objetivos específicos 9

2. REVISIÓN BIBLIOGRÁFICA 10

STICK-ONS. 10
Consistencia de los Stick-Ons. 11
REDES DE PETRI 14
Definición formal de una red de Petri. 15
Ejemplos de redes de Petri 16
Extensiones a Redes de Petri 21

3. INVESTIGACIÓN SITIO WEB 25

REVISIÓN HERRAMIENTAS PARA MANEJAR GRÁFICOS DISPONIBLES. 26
VML 26
SVG 26
Canvas tag 26
Librería gráfica basada en DIVs 27
Java 28
Flash. 28
Participación de Mercado 29
Conclusión herramientas gráficas 30
FLEX 31
Arquitectura de Flex 31
Flex es Open Source 32
Flex es orientado a objetos 32
Ejemplos de aplicaciones Flex 33
TECNOLOGÍAS WEB 35
Web Services 35
Ruby on Rails 35
PHP 37
ASP.Net 38
J2EE 38
EJB 38
Definición de EJB 39
AJAX 39
ColdFusion 39
Conclusiones tecnología web 40
BASE DE DATOS 41
EJEMPLOS USADOS PARA SITIO WEB 42

4. SITIO WEB 44

MODELO DE DATOS 45
Discusión del modelo 46
LOGIN, REGISTRO DE USUARIOS Y MÁS. 47
SEGURIDAD ¿QUÉ OCURRE AL INGRESAR AL SITIO? 54
DENTRO DEL SITIO 55
SECCIÓN DE CONTROLES 57

Nuevo diagrama *59*
Refrescar diagramas *59*
Desbloquear diagramas *59*
Cerrar sesión *60*
SECCIÓN DE NAVEGACIÓN 61
Utilización de espacio disponible *62*
Foco y opciones sobre elemento *65*
SECCIÓN DETALLE 69
Versiones: *72*
FILTROS 73
Funciones de filtrado *75*

5. STICK-ONS 80

JUSTIFICACIÓN DE LOS STICK-ONS 80
DISEÑO DE STICK-ONS PARA DIAGRAMAS 80
Diferencias entre texto y diagramas *80*
¿Qué se selecciona cuando se selecciona? *84*
Casos de uso básicos *86*
Múltiples Stick-Ons *88*
Sombras *92*

6. EDITOR DE DIAGRAMAS 98

ELEMENTOS DIAGRAMACIÓN 98
Diagramas de clases *98*
BaseComponent *98*
Handles *99*
Anchors *99*
Figure *100*
LinearC *100*
RectangularC *101*
Interfaces Selectable y Dockable *101*
SelectionManager *101*
DockManager *102*
Ejemplo *102*
Diseños alternativos del modelo *103*
DISEÑO DE UN DIAGRAMA 104
EDITOR 105
BARRA DE HERRAMIENTAS 107

7. DISCUSIÓN Y CONCLUSIONES 116

RESULTADOS 116
Manejo de quejas *117*
Notación para construcciones comunes *118*
Resolver Falla *119*
Ejercicio 2.8. Manejo de quejas *120*
Ejercicio 2.7 Fábrica de bicicletas *121*
Sombras nada más *122*
Stick-Ons *126*
CONCLUSIONES 132
POSIBLES MEJORAS 134

8. REFERENCIAS 135

1. Introducción y motivación

Introducción

Un diagrama es una representación visual simplificada y estructurada de una idea. Posibilidades de diagramas hay tantas como conceptos e ideas existan. Sin embargo, en el área de la computación y en áreas relacionadas de la ingeniería se han estandarizado algunos tipos de diagramas que representan un tema en común. Algunos ejemplos muy conocidos pueden ser un modelo de datos (diagrama entidad-relación), diagramas de flujo, diagramas de circuitos en el área de electricidad, un organigrama en una empresa (representación de los cargos y dependencias dentro de la organización), etc.

Herramientas para crear estos diagramas existen varias en el mercado. Posiblemente la mejor y más conocida sea *Visio* de *Microsoft*, que tiene un precio acorde a la calidad y potencia del producto. Además de su alto precio, tiene algunas debilidades inherentes a su diseño. Al ser una aplicación que corre localmente en un PC limita las posibilidades de que varias personas colaboren en el mismo diagrama. Además carece de funcionalidad para el manejo de versiones de los documentos, debiendo esto quedar a criterio del usuario y saber grabar periódicamente y con distintos nombres las versiones globales del documento.

Nosotros deseamos crear --y este es el tema principal de la presente memoria-- un sistema en *web* que permita construir diagramas en línea. Es decir, que usando un navegador de internet estándar como *Internet Explorer* o *Firefox* podamos conectarnos a un sitio *web* a ser implementado por nosotros y ser capaces de generar diagramas desde cualquier parte en donde tengamos conexión a Internet. Nos concentraremos en diagramar *Redes de Petri* [3], que pueden ser usadas para modelar *Workflows* entre otros usos. Además, queremos que nuestro sistema ofrezca facilidades para que diferentes individuos puedan trabajar en un mismo diagrama, y por consiguiente debemos ofrecer algún tipo de control local de cambios o de versiones. Para esto usaremos el concepto de los *Stick-Ons* [1,2], que aunque en la referencia [1] es usado para control de cambios en texto, nos parece apropiado su uso en un contexto principalmente visual como son los diagramas debido a la naturaleza gráfica de los *Stick-Ons*.

Justificación

Hoy en día es cada vez más común el uso de *workflows* para coordinar la ejecución de tareas y sus dependencias, las cuales no necesariamente están orientadas sólo a datos, si no que pueden involucrar la participación de varias personas. Por eso surge la necesidad de disponer de una herramienta que permita apoyar el uso de *workflows* y que disponga de facilidades para la colaboración entre distintos individuos, ya que todos los involucrados en el grupo de trabajo pueden participar, aportar ideas, tener visiones distintas, etc.

Para resolver este problema proponemos la creación de un sistema *web* que permita diagramar Redes de Petri, ya que como muestra *Van der Aalst* [5], estos diagramas son una herramienta adecuada para modelar *workflows.* Agregando a la funcionalidad de diagramación un control de versiones locales basado en *Stick-Ons* [1] (ya han sido aplicados en otras áreas como modelado de sólidos [5] y diagramas UML [6]), pensamos que se puede dar solución a la necesidad de un grupo para colaborar en el diseño de un *workflow*, donde las personas puedan hacer aportes y luego se genere una versión final integrando estos aportes.

No hemos encontrado ninguna herramienta que entregue los servicios que deseamos crear. El desarrollo presenta algunos desafíos, como la forma en que se implementarán los elementos gráficos de una Red de Petri para que sean de fácil uso para los clientes, o la manera en que se informará al usuario de los cambios que han ido ocurriendo en el documento (*awareness*), ya sean hechos por él o por otras personas. A continuación se detallan algunas ventajas que tendría nuestro sistema sobre un software tradicional de PC:

Multi-Plataforma. Al ser accesible desde un *browser* común y corriente, podrán trabajar personas que usan *Linux*, *Windows*, *MacOS*, etc. No será necesario tener que ingresar a *Windows* para usar *Visio* si es que se usa otro sistema operativo.

Distribución del uso. Dado que el sistema estará en Internet y no es necesario contar con un software especial en el cliente más que un navegador, se podrá acceder y trabajar con el sitio desde cualquier lugar con conexión a Internet, desde un café virtual o desde la casa por ejemplo.

Asíncrono. No se necesita que el grupo se coordine a un cierto momento en el tiempo para avanzar en el trabajo. Una persona puede hacer su aporte, y este será visible para los demás cuando éstos se conecten al sistema.

Fácil Colaboración. Ya que los documentos se almacenarán en forma centralizada, los usuarios podrán acceder y revisar fácilmente las distintas versiones de un documento, teniendo acceso inmediato a los últimos cambios realizados lo cual facilitará el trabajo colaborativo.

Control de versiones. Como esperamos que varias personas puedan modificar un documento, es razonable que algunos cambios no sean considerados apropiados por otros participantes del proyecto. Por ello se implementará un sistema de control de versiones, manteniendo almacenados en una base de datos los cambios de los usuarios y mediante el uso de *Stick-Ons* se ofrecerá una representación visual de los cambios.

Bajo costo de instalación. No es necesario instalar en cada PC de los participantes un software para realizar diagramas evitándose el costo de soporte que esto requiere. Además, en caso de una actualización del sistema con alguna corrección de *bugs* o una nueva funcionalidad, basta con actualizar el servidor y con eso todos los usuarios tendrán disponible la nueva versión.

Mayor seguridad de los datos. Como la información estará centralizada en el servidor, basta una política de respaldo rigurosa en esa máquina para evitar pérdidas de información, a diferencia de que cada integrante trabaje localmente lo cual requeriría un respaldo de la información de cada PC por separado.

Objetivos

Objetivo General

Se desarrollará un sistema en *web* que permita resolver el problema de creación de diagramas de Redes de Petri por un grupo de trabajo con control local de versiones.

Objetivos específicos

- El sistema deberá permitir a los usuarios dibujar diagramas de Petri básicos y extendidos, según la caracterización de Van der Aalst[4].
- El sistema deberá permitir a los usuarios almacenar y visualizar versiones parciales o totales de los diagramas de redes de Petri.
- El sistema deberá permitir también la especificación, almacenamiento y visualización de redes de Petri en distintos niveles de granularidad.
- El sistema deberá proveer facilidades de generar versiones “en limpio” de diagramas definitivos y poder exportarlas a otros sistemas.
- Estudiar usabilidad del sistema generado.

2. Revisión Bibliográfica

Stick-Ons.

Los *Stick-Ons* [1] son una herramienta de control de versiones local originalmente desarrollados en el marco de un software de escritura colaborativa llamado SHADOWS. Esta herramienta puede ser extendida a otras áreas, como diseño de sólidos [8] o diagramas UML [9], por nombrar algunas aplicaciones.

Usando una metáfora para ilustrar los *stick-ons*, supongamos que tenemos un papel con texto en él y hay alguna palabra, frase o párrafo que deseamos modificar. No es recomendable modificar directamente el texto ya que perderíamos la información previa. No sólo eso, es posible que en realidad el texto previo sea mejor que nuestra propuesta o que nuestro cambio sea solo tentativo. Una posibilidad para manejar esta situación en este ejemplo físico es pegar sobre el texto otro papel con nuestra versión revisada. Bajo este papel pegado aún se encuentra la información anterior. El concepto de estos papeles que parchan el texto pero que no pierden información son los *Stick-Ons*. Las analogías con un papel real no terminan aquí. Podemos usar papeles de color para indicar ciertas características, como el autor del cambio (el que pego el papel en la metáfora) o ponernos de acuerdo en ciertos colores para indicar el orden en que fueron agregados los *Stick-Ons*. Nada impide tampoco que se peguen *Stick-Ons* sobre *Stick-Ons*.

Junto con el concepto de *Stick-On* hay que entender la idea de la sombra del *Stick-On*. Cuando se están revisando los cambios dentro del documento y “despegamos” o quitamos un *Stick-On*, debemos dar una noción visual de que sobre este sector había un *Stick-On*. A este hint visual se le llama sombra o *shadow* en inglés.

Consistencia de los Stick-Ons.

Un requerimiento básico de los *Stick-Ons* es que no haya pérdida de información al "pegar" y "despegar" distintos *Stick-Ons*. En [1] y [2] se analizan cuatro casos básicos que describimos a continuación:

Caso 1. Supongamos que pegamos un *Stick-On* B sobre la sombra de otro *Stick-On* A, como se ilustra en la figura 1. El Stick-On B cubre completamente la sombra de A. Si se despega ahora el *Stick-On* B surge una situación que debe ser analizada. Si el sistema muestra las sombras de A y de B (figura 1a), el usuario no sabrá si se trata de 3 sombras continuas o de una sombra dentro de otra sombra. Por otro lado, si el sistema muestra en este caso solo la sombra de B, el usuario nunca podrá ver el contenido del *Stick-On* B de nuevo (figura 1b). Una solución simple es hacer que el sistema reponga automáticamente el *Stick-On* A antes de que se pegue el *Stick-On* B en el texto. Si el usuario ahora remueve B ahora vera el *Stick-On* A y la sombra de B (figura 1c).

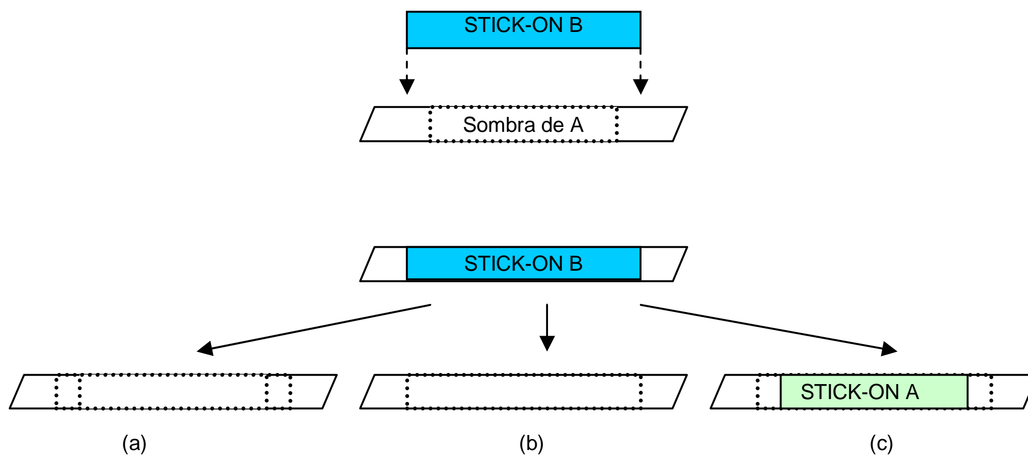


Ilustración 1. Despegado de STICK-ON B, puesto sobre sombra de A

Caso 2. Hay que tener cuidado con los solapamientos parciales. Supongamos, como ilustra la figura 2, que ponemos un *Stick-On* B y este cubre parcialmente una sombra de otro *Stick-On* A. Si se permite este solapamiento y el usuario quita la sombra de A (parte media de figura 2), llegamos a una situación en que el usuario no podrá volver a poner el *Stick-On* A, ya que al quitar el *Stick-On* B quedarán dos sombras continuas, lo que tendría un significado diferente. Para manejar este caso se proponen dos soluciones. La primera requiere del conocimiento del usuario y es simplemente que el sistema no permita solapamientos parciales. Para poder agregar una modificación el usuario deberá crear un *Stick On* que cubra completamente la sombra de A, llevando la situación a lo discutido en el caso 1. La segunda solución consiste en que el sistema agrande automáticamente el *Stick-On* que se está agregando y copie los contenidos de la sombra que cubriría el nuevo *Stick-On*. Con este enfoque el usuario deberá estar alerta a que los *Stick-Ons* que esta agregando serán modificados en algunas situaciones.

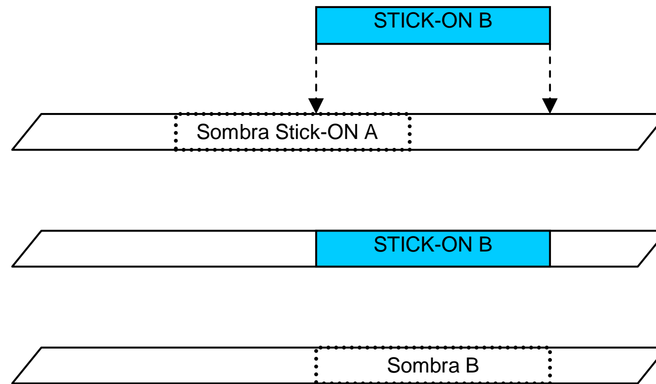


Ilustración 2. Solución falsa de solapamiento parcial

Caso 3. El caso 3 consiste de un *Stick-On* que está completamente contenido dentro de una sombra. En esta situación el cuidado que se debe tener es poder distinguir visualmente un *Stick-On* sobre una sombra de una sombra sobre un *Stick-On*. Esto se consigue al aplicar una textura diferente a las sombras, como se ilustra en la figura 3.

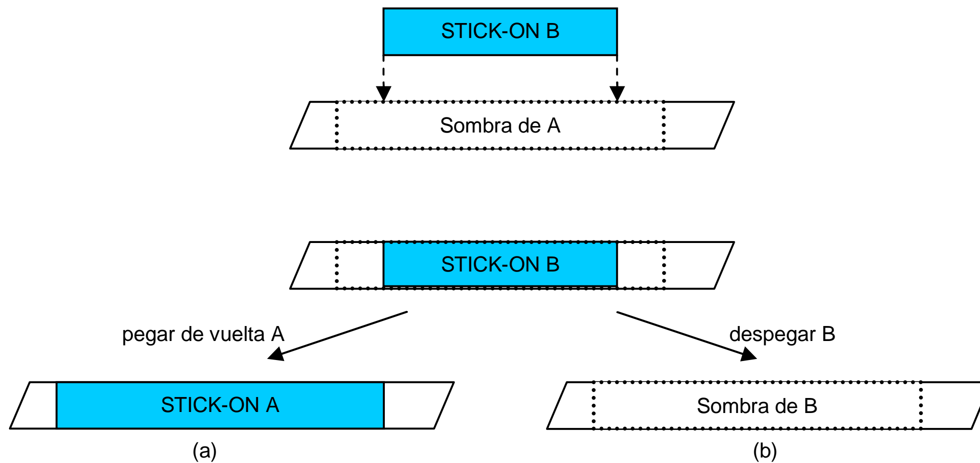


Ilustración 3. Pegado de un Stick-On dentro de una sombra

Caso 4. El cuarto caso, discutido en [2], considera la situación en que existe una sombra A y sobre ella se agrega un *Stick-On* B. En la figura 4 se ilustra esta situación. Si se despega el *Stick-On* B, el modelo indica que se debe mostrar la última sombra, de otro modo seria confuso visualmente para el usuario (figura 4b). Supongamos que se agrega un *Stick-On* C que cubre parcialmente el texto (invisible en estos momentos) de la sombra A (figura 4c). Si se permite esto llegamos a una situación en que no podemos recuperar el Stick-On A ya que se ha perdido toda indicación de su existencia. Una solución para este problema se muestra en la figura 4d. El *Stick-On* C se agranda hasta cubrir la sombra de A. Más aún, el sistema pega de vuelta los *Stick-*

Ons anteriores antes de transformar el *Stick-On* C. Ahora si el usuario despega C*, encontrará de vuelta el *Stick-On* B en su posición. Con esto no hay pérdida de información.

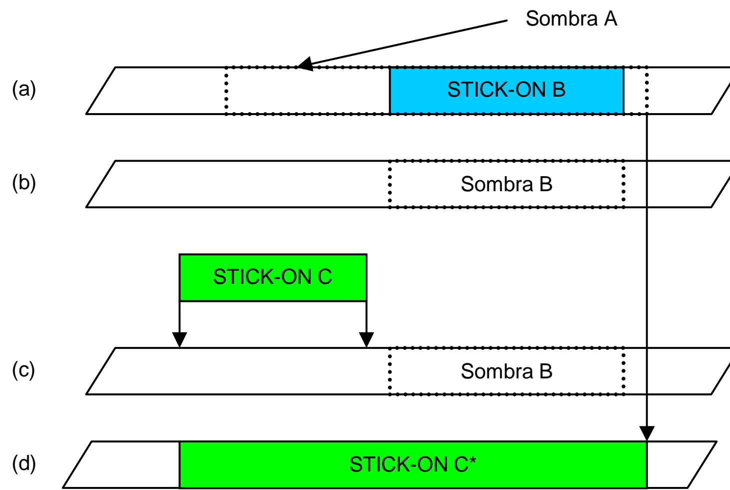


Ilustración 4. Potencial caso de pérdida de información

Redes de Petri

Las redes de Petri [3] son una herramienta de modelado gráfico y matemático. Fueron inventadas el año 1962 por Carl Adam Petri [4] en su tesis doctoral y desde esa fecha han estado en permanente desarrollo.

En [5] se da una descripción simple de las Redes de Petri, caracterizándolas como un grafo dirigido con un estado inicial, llamado *marcación inicial*. Los componentes principales de la Red de Petri son los *sitios* (también conocidos como *estados*) y las *transiciones*. Gráficamente, los sitios son representados por círculos y las transiciones como barras o rectángulos. Las aristas del grafo son conocidas como arcos y tienen pesos específicos representados por un número entero no negativo, o bien como n arcos paralelos. El peso por defecto de un arco es 1.

El estado del sistema modelado por la Red de Petri es representado por una cantidad entera en los sitios. Dicha asignación de valores es llamada una *marcación*, y se representa gráficamente con pequeños círculos dentro de los sitios, conocidos como fichas o el término original en inglés: *tokens*. Cuando el número de *tokens* es demasiado grande se reemplaza por un número no negativo dentro del sitio.

Normalmente los estados representan alguna condición del sistema y una transición representa un evento. Lo que los *tokens* representan puede variar dentro de un amplio rango, pudiendo representar cuando una condición es verdadera o cierta cantidad de recursos, por ejemplo el número de *clicks* con el Mouse.

Una transición *t* está habilitada si cada sitio de entrada tiene al menos la cantidad de *tokens* requerida por los arcos que llevan a *t*. Cuando más de una transición está habilitada, se selecciona una de ellas de manera no determinística dependiendo del modelo empleado.

Definición formal de una red de Petri.

Una definición formal para una red de Petri se encuentra en [5], caracterizándola como una 5-tupla (S, T, F, M_0, W) donde:

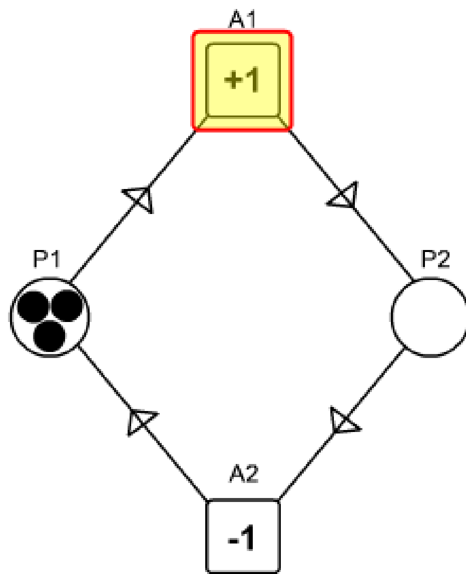
- S es un conjunto de *lugares*.
- T es un conjunto de *transiciones* o *actividades*.
- F es un conjunto de arcos conocido como *flujo*. El conjunto F esta bajo la condición de que ningún arco debe conectar dos lugares o dos transiciones, o bien que: $F \subseteq (S \times T) \cup (T \times S)$.
- M_0: $S \rightarrow \mathbb{N}$ es un estado inicial donde en cada lugar $s \in S$ existen $n_s \in \mathbb{N}$ fichas.
- W: $F \rightarrow \mathbb{N}^+$ es un conjunto de pesos que le asignada a cada arco $f \in F$ algún $n \in$ a $\mathbb{N}^+$ que representa la cantidad fichas que se consumen desde un lugar por una transición, o alternativamente, cuantas fichas son producidas por una transición y puestas en un lugar.

Ejemplos de redes de Petri

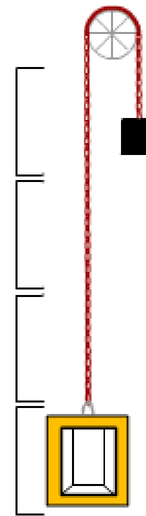
Es difícil entender una Red de Petri sin ver un sistema real modelado. Por ello revisaremos dos modelos que usan redes de Petri tomados de [6].

Elevador

El siguiente ejemplo modela un elevador que pude moverse por 3 pisos. Veamos la situación inicial:



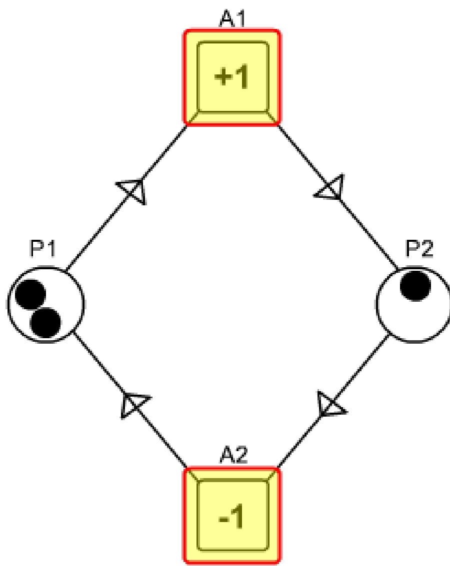
A. Estado inicial



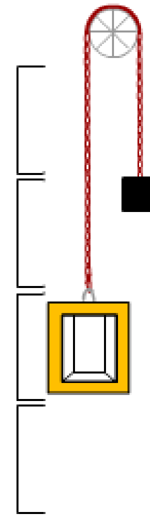
B. Estado representado

Ilustración 5. Elevador de tres pisos, estado inicial

En la figura 5A se puede apreciar una Red de Petri representando a un elevador que está en el primer piso (figura 5B). En la situación diagramada, solo es posible la transición A1 (marcada en rojo) que representa subir un piso. La transición A2, que representa bajar un piso, no es posible en estos momentos ya que no existen *tokens* en P2. Esto está de acuerdo con la realidad, porque cuando el ascensor esta en el primer piso no es posible bajar en el caso representado. En este ejemplo los *tokens* representan la cantidad de movimientos verticales (hacia arriba o hacia abajo) que puede hacer el ascensor en un estado dado.



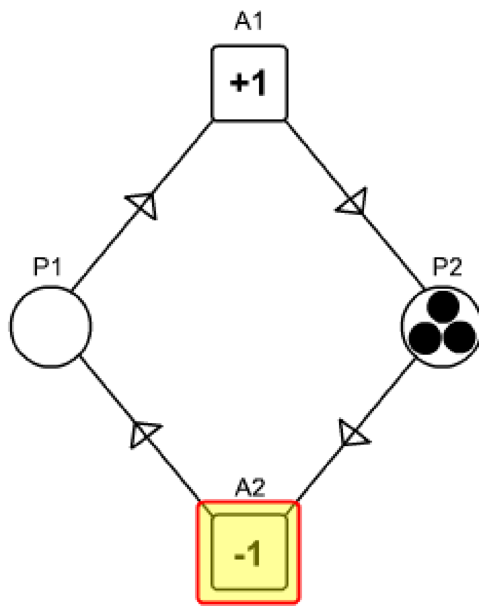
A. El ascensor ha subido un piso



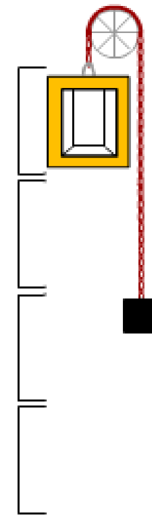
B. Estado representado

Ilustración 6. Elevador de tres pisos, ha subido un piso

En la figura 6A se muestra el sistema luego de que ha ocurrido la actividad A1: Subir un piso. En la figura 6B se muestra la situación real, el ascensor subió al segundo piso. Revisemos las similitudes entre la realidad y lo modelado por la Red de Petri. En primer lugar, al estar en el segundo piso ahora hay dos actividades posibles, subir (hasta 2 pisos más) o bajar (un piso). Lo mismo ocurre en la Red de Petri ya que son posibles las actividades A1 y A2 (marcadas en rojo) por tener *tokens* disponibles en los lugares P1 y P2. Al ser posibles A1 y A2, se puede producir cualquier de estas dos actividades en forma no determinística. Nosotros continuaremos el ejemplo haciendo subir el ascensor dos pisos.



A. El ascensor ha subido 3 pisos.



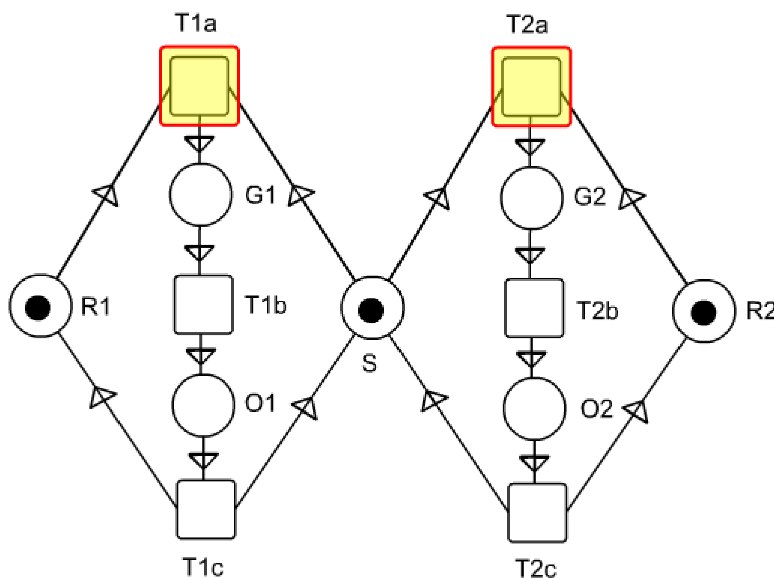
B. Estado representado

Ilustración 7. Elevador de tres pisos, estado final

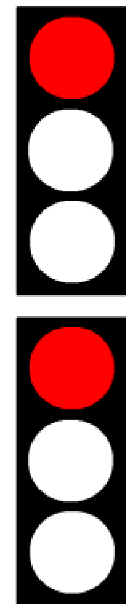
En la figura 7 se representa el sistema luego de haber subido 3 pisos. Se puede ver que la única actividad posible en estos momentos es bajar, o la actividad A2 (en rojo) ya que es la única con *tokens* disponibles para consumir. Los *tokens* muestran que se pueden bajar 3 pisos en estos momentos.

Semáforo

Un ejemplo algo más complicado es una red de Petri representando dos semáforos en donde no es posible que se prendan dos luces que permitan el paso simultáneamente, por ejemplo verde y verde, o verde y amarillo. Revisemos la solución:



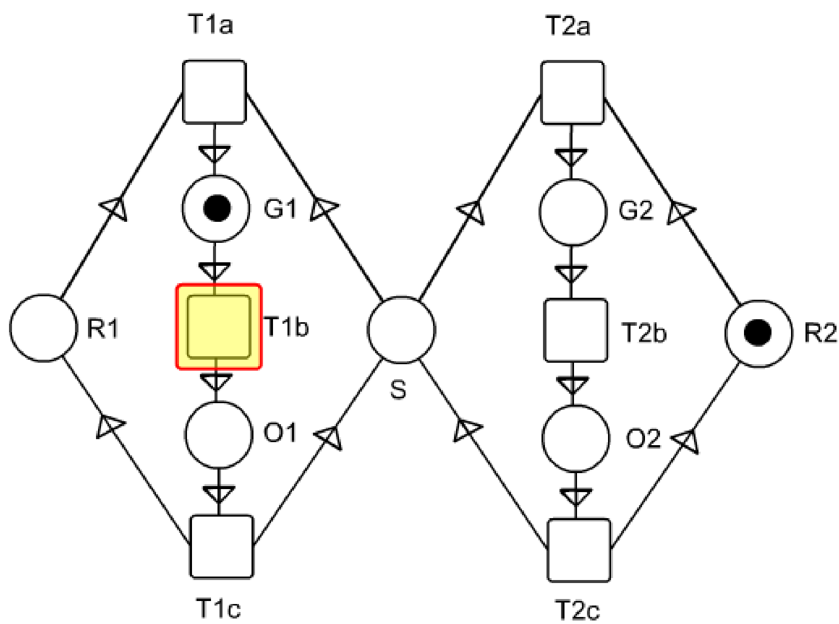
A. Modelo para dos semáforos

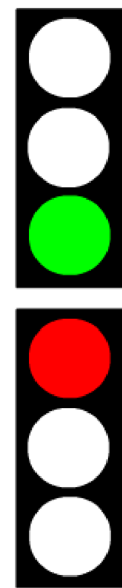


B. Estado inicial

Ilustración 8. Modelo de dos semáforos, estado inicial.

Tenemos la marcación inicial representada en la figura 8. Esta situación representa los dos semáforos en rojo. Cualquiera de las dos actividades (T1a y T1b) en amarillo es posible y representan el cambio de uno de los semáforos a verde. Esta transición es no determinística, pero lo interesante es que una vez tomada, la situación se vuelve determinística en donde el semáforo que quedo en rojo ya no podrá salir de ese estado porque le faltará el *token* que estaba en S. Veamos gráficamente la situación:

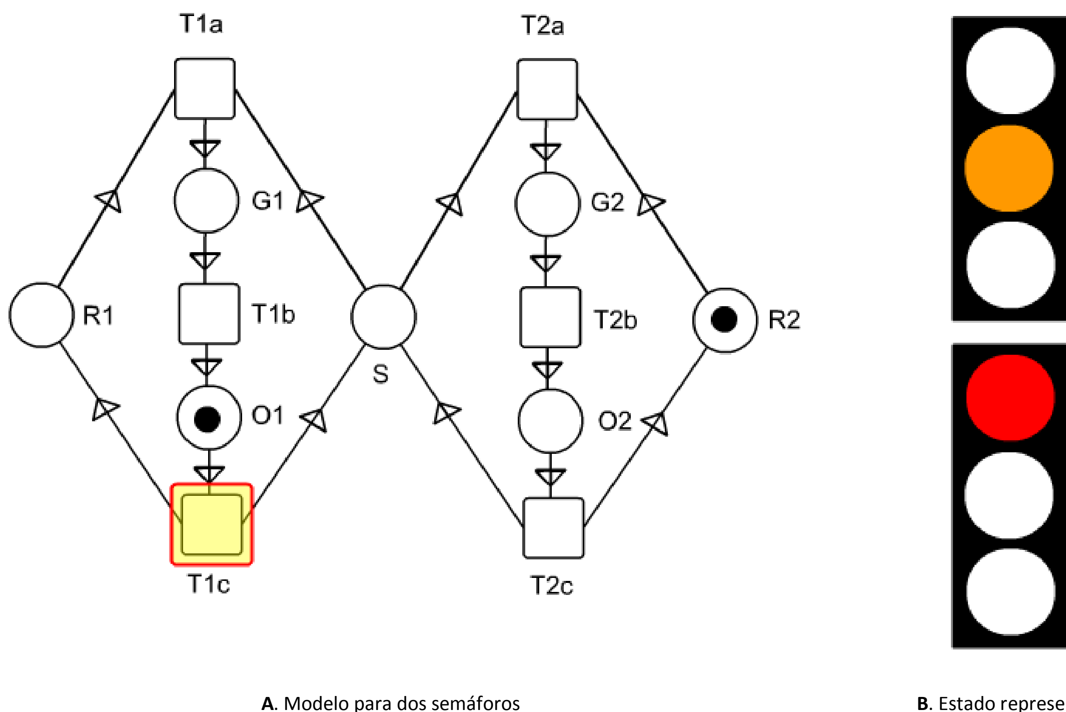




A. Modelo para dos semáforos **B**. Estado representado

Ilustración 9. Modelo para dos semáforos, estado intermedio.

La transición T1a fue la que se siguió en la figura 8 resultando el estado de la figura 9. Hemos pasado a verde en el semáforo 1 y ahora ya no es posible que el semáforo 2 salga del color rojo porque le falta un *token* en S para poder hacerlo. Vemos que la cantidad de *tokens* no necesariamente es constante en una Red de Petri ya que ahora solo hay dos *tokens*. En este estado, la única posibilidad es la actividad T1b que lleva al semáforo 1de verde a amarillo (o naranjo).



A. Modelo para dos semáforos

B. Estado representado

Ilustración 10. Modelo para dos semáforos, estado intermedio.

En la figura 10 tenemos un semáforo en rojo y otro en amarillo. La actividad siguiente (T1c) volverá al primer semáforo a rojo e instalará una ficha en S, con lo que volvemos al estado inicial y ahora podría el segundo semáforo cambiar a estado verde (G2).

Extensiones a Redes de Petri

Entre las aplicaciones en donde se usan las Redes de Petri se cuenta el modelado de *Workflows*. En [12] se analizan *workflows* usando un enfoque formal basado en Redes de Petri. Las ventajas de esto son variadas, permitiendo tener una representación gráfica del proceso y eliminando ambigüedades y contradicciones. También se describen algunas extensiones a las Redes de Petri, detalladas en los puntos siguientes:

A. Redes de Petri coloreadas.

Como ya se había mencionado, los *tokens* o fichas son usadas para modelar una amplia variedad de cosas. Pueden representar reclamos de seguros en un caso, y en otra situación pueden representar el estado de las luces de un semáforo. Sin embargo, en el modelo original, los *tokens* son indistinguibles unos de otros por definición. En general esto puede ser no deseable. Por ejemplo podríamos querer distinguir entre dos reclamos de seguros, o querer incluir la naturaleza del reclamo, el número de seguro, etc. Por ello esta extensión propone asignar un valor o color a cada *token*. Una transición produce *tokens* basados en los valores de los *tokens* de entrada. Esta extensión permite que se establezcan condiciones sobre los *tokens* a ser consumidos. En este caso una transición solo ocurre si existe un *token* para ser consumido y si se cumplen las precondiciones.

Un resultado de esta extensión a las Redes de Petri es que la representación gráfica ya no contiene toda la información. Ahora se debe especificar para cada transición si es que tiene precondiciones (y de haberlas definirlas), el número de *tokens* producidos por salida y los valores de los *tokens* producidos, los cuales pueden depender en los valores de entrada.

Ejemplo de *tokens* o fichas coloreadas:

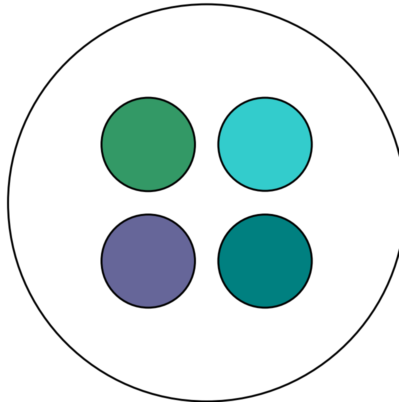


Ilustración 11. Extensión de fichas coloreadas

B. Extensión de tiempo.

Dado un proceso modelado por una Red de Petri, a menudo podríamos querer tener conclusiones respecto al desempeño esperado. Por ejemplo, si se modela un semáforo en un cruce de caminos, nos podría interesar la cantidad de autos que pasan por este cruce cada hora. Esta respuesta no se puede obtener usando una Red de Petri clásica, por lo que se extiende la definición de una Red de Petri para que contenga esta nueva dimensión. Bajo esta definición, los *tokens* reciben una marca de tiempo (*timestamp*) y un valor. Esto indica el momento a partir del cual estará disponible el *token*. Un *token* con *timestamp* 14 está disponible para una transición solo desde el momento 14. Una transición está habilitada solo cuando todos los *tokens* para ser consumidos tienen un valor del *timestamp* mayor o igual al momento actual. Los *tokens* son consumidos en un orden FIFO (el primero en llegar es el primero en salir).

Un ejemplo de cómo se representa gráficamente esta extensión:

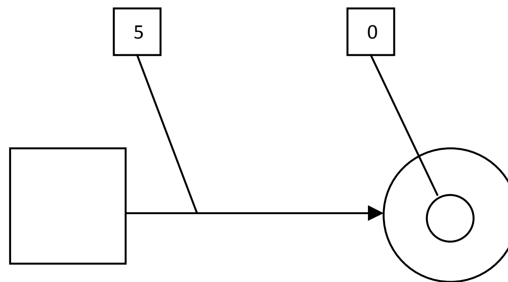


Ilustración 12. Extensión de tiempo

C. Extensión Jerárquica.

Esta extensión está pensada para ayudar a representar más fácilmente la estructura del modelo representado, permitiendo encapsular dentro de una figura especial llamado un *proceso* una subred compuesta de actividades, transiciones y subprocesos. Ya que un proceso puede estar compuesto a su vez de subprocesos, es posible estructurar una jerarquía compleja. Para ilustrar este nuevo elemento “proceso” usaremos de ejemplo un modelo para manejar fallas técnicas en un departamento de producción.

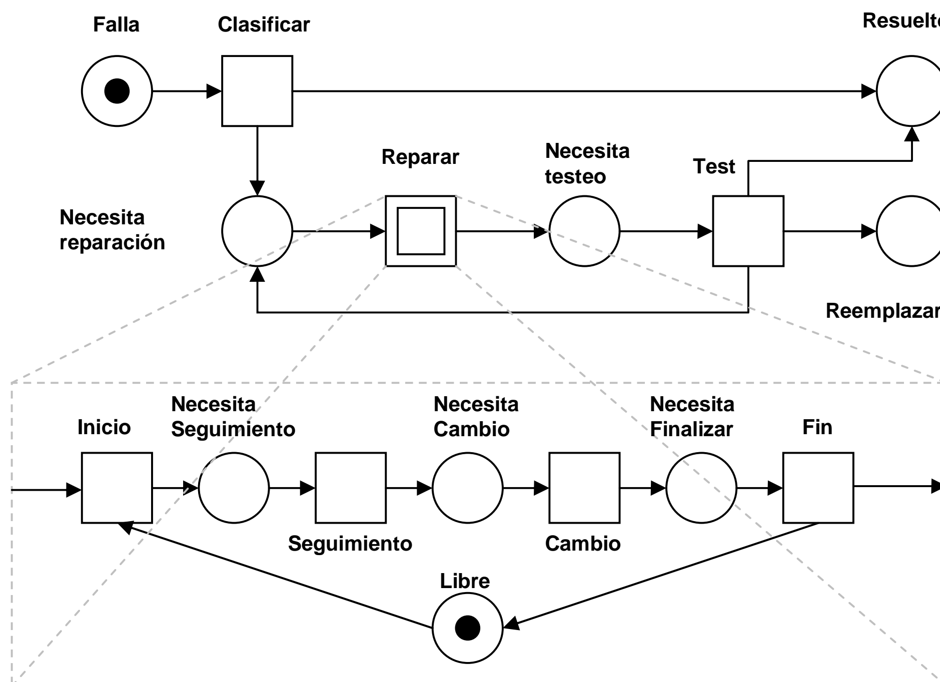


Ilustración 13. Extensión jerárquica

El diagrama previo representa un *workflow* de manejo de fallas. Podemos apreciar que la actividad “reparar” no es una única acción indivisible, si no que un subproceso que detalla las actividades y transiciones involucradas al reparar un producto.

3. Investigación Sitio Web

En este capítulo se revisan distintas opciones y tecnologías para la implementación del sitio web. Se investigan las tecnologías gráficas disponibles que nos permitan crear diagramas en un navegador de internet, como *Internet Explorer* o *Mozilla Firefox*. También revisaremos las alternativas para implementar los servicios que nos permitan grabar y recuperar la información desde un servidor web. Finalmente, se selecciona un motor de base de datos para ser usado por la aplicación.

Revisión herramientas para manejar gráficos disponibles.

El lenguaje HTML no posee funcionalidades directas para generar elementos gráficos dinámicamente. Los gráficos que se ven dentro de las páginas *web* generalmente son mapas de bits, imágenes estáticas en formatos GIF, JPG o PNG en general, pero no es sencillo proveer mayor dinamismo gráfico usando HTML estándar. Esta limitación ha hecho que se creen una serie de soluciones en forma de *plug-ins* o bien de estándares con poco soporte. En esta sección se revisarán algunas de las alternativas actuales.

VML

Vector Markup Language (VML) [13] es una extensión del lenguaje XML orientado a generar gráficos vectoriales. Es un estándar propuesto por *Microsoft*, *Macromedia* y otros en el año 1998 al consorcio W3C. La propuesta fue rechazada debido a la existencia de una propuesta alternativa conocida como PGML generada por Adobe, Sun y otros. Ambas propuestas, VML y PGML fueron fundidas y mejoradas en un solo estándar: SVG.

Aunque no fue aceptado como estándar, Microsoft sigue soportando el lenguaje VML en su navegador Internet Explorer y en la suite Office, por lo que no se puede descartar completamente. De hecho, la aplicación en Internet Google Maps usa VML para dibujar los gráficos vectoriales cuando se despliega en Internet Explorer.

SVG

Scalable Vector Graphics (SVG) [14] es un lenguaje para describir gráficos vectoriales en dos dimensiones que nace de la propuesta de VML y PGML. Está basado en XML y es un estándar del consorcio de la World Wide Web.

Tiene algunos atributos que lo hacen atractivo:

- Los gráficos SVG pueden ser dinámicos e interactivos
- Si se desea, los archivos SVG pueden ser comprimidos usando compresión gzip

Desafortunadamente este lenguaje no es soportado en forma nativa por Internet Explorer, requiriendo de un *plug-in* para funcionar. El *plug-in* más usado es el de Adobe Systems, pero Adobe ha anunciado que descontinuará el soporte para dicho plug-in en enero del 2008.

Canvas tag

Originalmente introducido en el navegador Safari de *Apple* y posteriormente adoptado por *browsers* basados en el motor *Gecko* como *FireFox* y *Mozilla*, el elemento “Canvas” es parte de la especificación de HTML5 y permite dibujar gráficos usando Javascript.

Desde su introducción ha recibido reacciones mezcladas. Por una parte se critica que viene a ocupar el rol del estándar SVG que ni siquiera ha sido adoptado universalmente. También se ha criticado su diseño. Al ser completamente procedural y carecer de una parte descriptiva, permite “pintar” en pantalla, pero los elementos dibujados no son identificables posteriormente.

Este nuevo elemento HTML no es soportado por Internet Explorer. Sin embargo, existen esfuerzos de terceros orientados a dar compatibilidad con el *tag* Canvas mediante librerías que implementan su funcionalidad. Por ejemplo Google ha mostrado interés en esto y ha liberado bajo formato open source su librería excanvas [15].

Un problema serio a nuestro gusto es que en la implementación del Canvas, aún no está finalizado el soporte para texto. Esto es lo que dice la propuesta de la WHATWG respecto a HTML5 y el *tag* CANVAS en lo referente al texto [16]:

```
// drawing text is not supported in this version of the API
// (there is no way to predict what metrics the fonts will have,
// which makes fonts very hard to use for painting)
```

Librería gráfica basada en DIVs

Aunque HTML no dispone de una forma estándar de dibujar gráficos proceduralmente, si es posible hacerlo en forma indirecta. Por ejemplo dentro de la definición de HTML existe un elemento de bloque llamado DIV, el cual si se analiza solo su parte grafica puede ser visto como un rectángulo de un color dado, en una posición especifica. Si el rectángulo se define como un rectángulo de 1x1, puede ser visto como un único píxel. Con esto ya se tiene una forma de dibujar a pantalla usando solo elementos estándares de HTML.

Esto es exactamente lo que hace la librería desarrollada por Walter Zorn para dibujar gráficos vectoriales usando javascript [17]. Esta librería puede servir de punto de partida para implementar figuras más complejas. Tiene ciertas optimizaciones que la hacen más atractiva que partir de cero. Por ejemplo, para trazar líneas usa el algoritmo de Bresenham que es conocido por su eficiencia, ya que usa sólo aritmética entera y operaciones de *shift*. También aprovecha que los DIVs que forman los pixeles son realmente rectángulos, sacando partido cuando es posible extender estos pixeles para no tener que crear varios objetos distintos con la sobrecarga que eso conlleva. Esto se ilustra en el siguiente diagrama, a la izquierda se ve el método directo en donde cada píxel es un DIV, y a la derecha se aprecia que cuando se puede usar un rectángulo la librería lo hace:

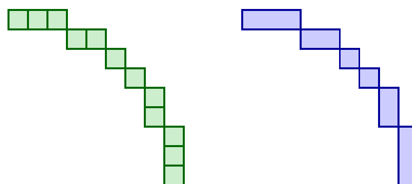


Ilustración 14. Ejemplo de dibujo usando DIVs.

Java

Una alternativa que viene inmediatamente a la mente cuando se piensa en hacer gráficos en una página *web* es usar un *applet* de Java [10]. Un *applet* es un programa que corre dentro de un *browser* mediante el uso de una maquina virtual que interpreta su código. Los *applet* se escriben usando el lenguaje de programación JAVA. El resultado de la compilación del programa resulta en un código llamado *Bytecode*, el cual es independiente de la plataforma. Por ello pueden ser ejecutados en una variedad de plataformas siempre y cuando exista una maquina virtual que lo interprete. El uso más común de los *applet* es proveer de funciones interactivas que no pueden ser manejadas por HTML. Entre las ventajas y desventajas de los applets se pueden contar:

Ventajas

- Independiente de la plataforma.
- Al correr con permisos limitados (en una *sandbox*) no es necesario que el usuario autorice su ejecución.
- Esta soportado por la mayoría de los browsers
- Corre a una velocidad comparable a C++ (generalmente un poco menos)

Desventajas

- Requiere de un plug-in que no está disponible para todos los browsers
- No parte hasta que no esté andando la maquina virtual de Java, lo cual lo hace ser lento para partir.
- Se considera que es más difícil construir buenas interfaces de usuario que en HTML

Flash.

Breve Historia de Flash

El producto conocido como Flash [17] comenzó su vida en las mentes de John Gay y Robert Tatsumi como un software llamado SmartSketch que permitía a los usuarios crear ilustraciones en un *TabletPC*. Esto fue a mediados de los años 90, cuando el primer boom de las "punto com" estaba en pleno auge.

La segunda versión del producto se llamó "FutureSplash Animator" y soportaba animaciones. Venía incluido un pequeño *plug-in* que permitía ver animaciones incrustadas en páginas web. En este punto el producto anduvo tan bien en lo que hacía que algunas compañías de medios como *Disney* y *MSN* lo incluyeron en sus páginas *web* a fines del 96. En ese tiempo Macromedia se dio cuenta del potencial del producto de Gay y Tatsumi y compró su compañía, *Future Wave Software*, y renombró el software a Flash.

Uno de los puntos más importantes en la historia de Flash vino en 1997. La guerra de los *browsers* entre Netscape e Internet Explorer estaba en su punto alto y Macromedia se veía enfrentada a un dilema al estilo del huevo y la gallina: para que los desarrolladores usaran su producto, tenía que haber una base de usuarios más o menos importante que tuviera instalado el *plug-in* y pudiera ver su contenido. Desafortunadamente para ellos había cientos de *plug-ins* en ese tiempo y los usuarios no estaban dispuestos a bajar otro *plug-in* a menos que hubiera mucho contenido disponible. ¿Que se podía hacer?

La compañía encontró una solución. Netscape estaba ganando la guerra de los browsers en ese tiempo, así que Macromedia le pago a Netscape una considerable cantidad de dinero para que incluyera su *plug-in*. Microsoft, para no verse en menos, también incluyo el *plug-in* en su instalación sin cobrarle nada a Macromedia. Gracias a este trato Flash se volvió el plug-in más común en Internet y se le puede ver casi en todos lados, llegando a una penetración de mercado [11] de aproximadamente un 98%. Algunos sitios como www.youtube.com, recientemente vendido a Google en U$1.500 millones lo usan con gran éxito.

Hoy en día, tras más de 10 años de evolución, Flash se ha vuelto una plataforma robusta y ampliamente soportada, con herramientas maduras que ayudan con el desarrollo de nuevas aplicaciones más allá de las animaciones y los videos que típicamente encontramos en Internet.

Participación de Mercado

El siguiente gráfico, tomado de [11] muestra la participación de mercado de los actores más importantes de multimedios.

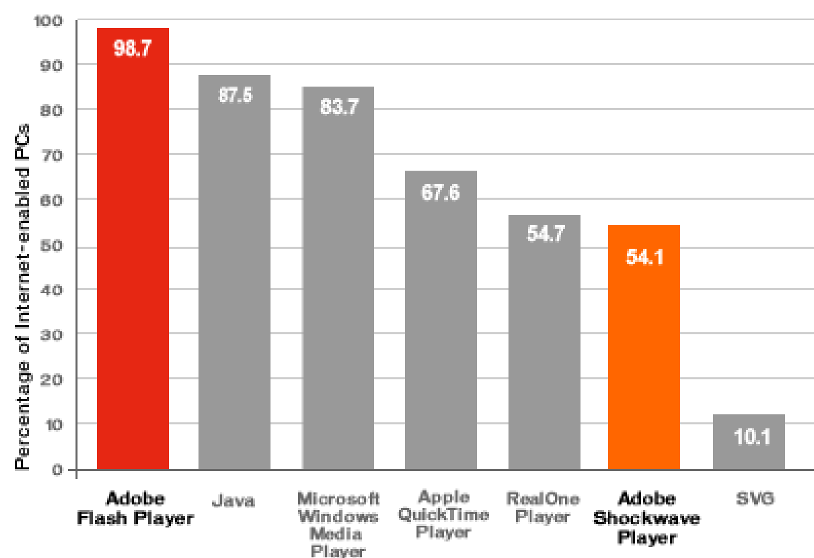


Ilustración 15. Participación de mercado.

Conclusión herramientas gráficas

Entre las opciones investigadas, las tecnologías más maduras y extendidas parecen ser *Flash* y *Java* (*applets* en el lado del cliente). Entre ellas, nos inclinamos por Flash debido a su mayor participación de mercado y a que su máquina virtual es más liviana que la de Java, permitiendo un uso con menos interrupciones y una experiencia en general más fluida. Dentro de Flash existe un producto llamado Flex, que a diferencia de Flash tradicional que está orientado al desarrollo de animaciones, está orientado a crear aplicaciones en internet. Este es el producto que se usará para la parte gráfica del sitio y lo revisaremos en detalle en la siguiente sección.

Flex

Tradicionalmente las aplicaciones disponibles en web han sido algo limitadas en lo que respecta a la interfaz de usuario. Normalmente se debe usar los controles básicos que ofrece HTML tales como *radio buttons*, *selects*, *checkboxs*, etc., todos ellos de relativa simplicidad. Con bastante esfuerzo es posible construir y extender nuevos controles, por ejemplo un control que permita ingresar fechas y que despliegue un calendario, como en [18]. A pesar de lo logrado que pueda estar el control, como es el caso del ejemplo, existen una variedad de desventajas a este enfoque como que no existe un comportamiento homogéneo entre los distintos sitios ya que en uno pudieron ocupar este control de fecha, y en otro haber usado otro distinto con una funcionalidad diferente. Si esta tendencia se amplía a las páginas completas, se comprende que cada sitio de cierta complejidad debe ser aprendido en forma independiente. Desde el punto de vista del desarrollador también se presentan dificultades. ¿Este control con que navegadores es compatible? Debo investigar hasta la implementación del control para tener alguna certeza. En este contexto se entiende la ventaja, o la necesidad hasta cierto punto, de contar con un *framework* maduro y consistente. Este espacio es el que intenta llenar Adobe Flex.

Flex permite a los desarrolladores crear aplicaciones con una interfaz de usuario comparable a la experiencia que encontrarían en una aplicación de escritorio, pero usando un *browser*. El término usado para describir esta nueva generación de aplicaciones es RIA, por *Rich Internet Applications*. En esta sección se revisarán distintos aspectos del marco de trabajo elegido para implementar la parte gráfica del desarrollo, Flex.

Arquitectura de Flex

La arquitectura de Flex puede ser descrita en base a tres componentes principales. Un lenguaje de *markup* basado en XML llamado MXML, la programación de la lógica o extensión de componentes usando el lenguaje Actionscript, actualmente en la versión 3.0, y finalmente un conjunto de clases que proveen los controles de usuario y las funcionalidades necesarias para poder desarrollar una aplicación RIA. Con estos componentes y el compilador provisto por Adobe se genera *bytecode* que es interpretado por la máquina virtual de *actionscript* (Flash player). Esta arquitectura se resume en la siguiente figura:

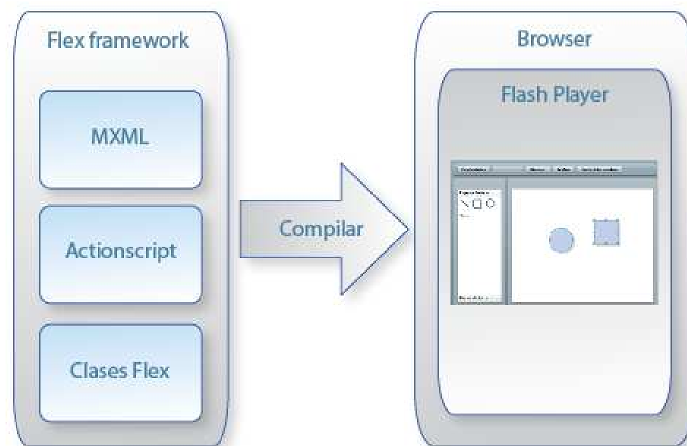


Ilustración 16. Arquitectura Flex.

Flex es Open Source

Adobe ha anunciado que el SDK de Flex y la documentación serán open source bajo la licencia de Mozilla, MPL (Mozilla Public License). Esto incluye el código en Java del compilador de Flex (compilador de Actionscript y compilador de MXML), el código Actionscript de las clases base del SDK de Flex y el debugger de Actionscript. De todos modos se podrán licenciar estas componentes en caso de que alguna empresa no desee desarrollar bajo la licencia open source.

No será open source el IDE FlexBuilder, ni las componentes de Flex para hacer gráficos estadísticos o los servicios en el servidor para Flex. Esto último podría parecer preocupante, sin embargo no es obligatorio usar los servicios de Flex (*Flex data services*), ya que se puede usar un *webservice* estándar para comunicarse con Flex.

Flex es orientado a objetos

La aplicación que deseamos desarrollar usando Flex, una herramienta de diagramación, pareciera ser el ejemplo ideal de programación orientada a objetos. Los objetos a diagramar compartirán varias propiedades por lo que nos sería muy útil una herramienta que respete el paradigma de la programación OOP. Afortunadamente, al poco andar en el desarrollo en Flex se nota que es un entorno fuertemente orientado a objetos. Por un lado la programación de una aplicación estándar en Flex involucra casi únicamente el uso de componentes (objetos) y eventos (mecanismo de comunicación entre objetos), y por otro lado el lenguaje de desarrollo Actionscript soporta desde su versión 2.0 los elementos claves de la programación orientada a objetos tales como herencia, encapsulamiento, polimorfismo, etc.

Ejemplos de aplicaciones Flex

Ya que Flex es una tecnología relativamente nueva, sería interesante saber si ha sido usado en proyectos de importancia y como ha resultado la experiencia. Podemos responder que si ha sido usado, por ejemplo para implementar la interfaz gráfica del sitio de mapas de Yahoo (*Yahoo maps*) exitosamente. Revisemos sólo algunos de los sitios que han usado Flex.

Yahoo

Como se comentó con anterioridad, Yahoo usó Flex para implementar Yahoo maps. Seguramente esta es la aplicación más importante que usa Flex.

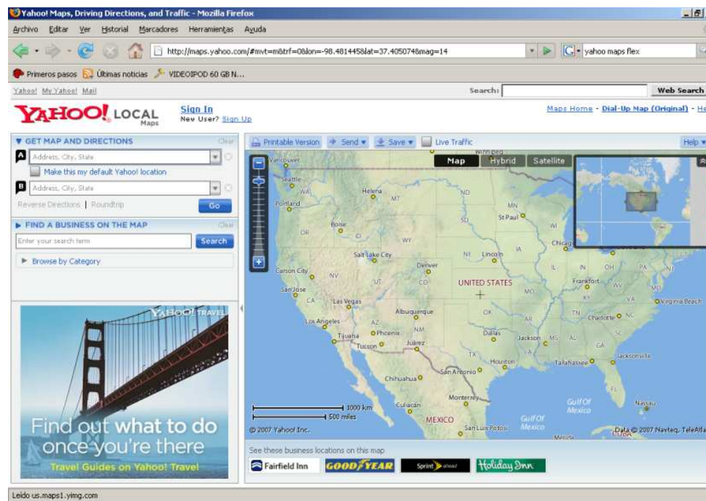


Ilustración 17. Yahoo Maps.

Volkswagen

Dentro del sitio de Volkswagen se puede encontrar una aplicación para buscar y comparar autos implementada en Flex.

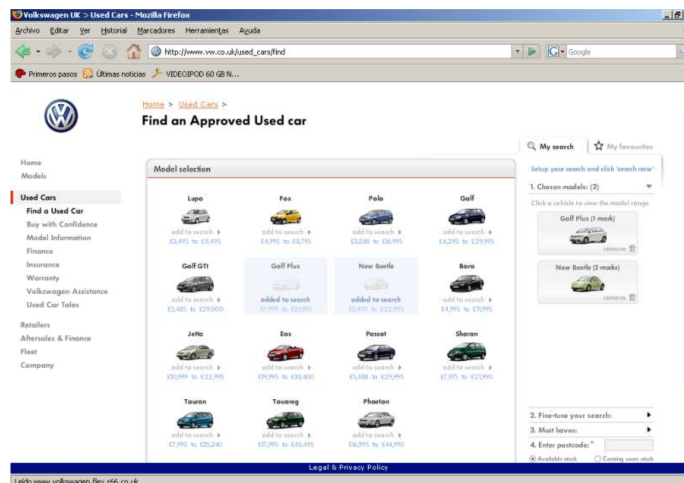


Ilustración 18. Sitio de Volkswaken.

Harley-Davidson

En la página de Harley-Davidson se puede lanzar un diseñador de motos, para poder cambiar los colores y agregar accesorios. Esta aplicación fue creada usando Flex.

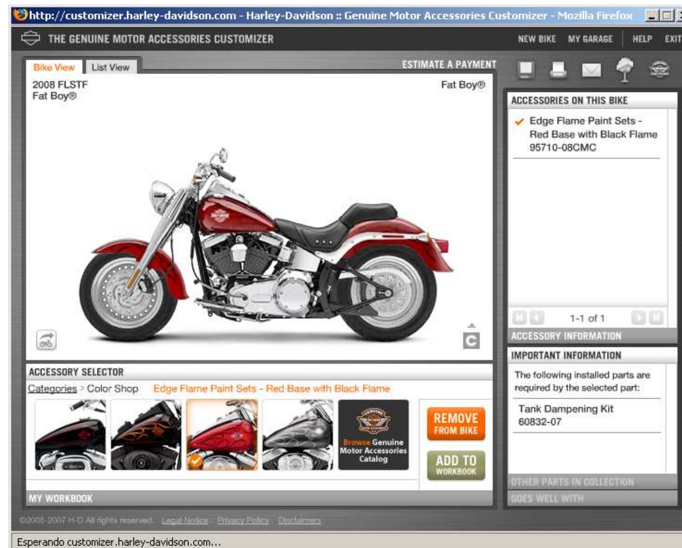


Ilustración 19. Sitio de Harley-Davidson

Tecnologías web

Si se me permite una pequeña experiencia personal, entré a trabajar hace cinco años a una empresa que tiene un producto en Internet, y me he especializado en nuestro software. Usamos una solución desarrollada internamente que consiste en una *Isapi* (extensión del servidor web) y que tiene la función de recibir requerimientos y enviarlos a objetos COM o CORBA. Ahora, al momento de revisar que tecnología podría ser apropiada para desarrollar el presente trabajo me he encontrado con una variedad de nuevas soluciones que no existían hace cinco años, que francamente pueden ser un poco intimidantes y a veces frustrantes al pensar que en cinco años más van a haber nuevas alternativas que desconoceremos. Supongo que esto siempre será así, por lo que es mejor no sufrir demasiado pensando en eso y pasar de inmediato a la revisión de las tecnologías existentes.

La siguiente selección no pretende ser una revisión exhaustiva de todas las tecnologías web existentes el año 2007, pero si al menos una revisión de algunas de las más importantes y prometedoras.

Web Services

Aunque es más una definición que una tecnología en particular que pudiéramos elegir para implementar nuestro sitio, conviene detallar esta palabra ya que será mencionada más adelante. Un Web Service se define como sistema de software que permite la interacción entre máquinas en una red [25]. En la práctica, usualmente se trata de una API Web que puede ser invocada desde internet y ejecutada en un servidor remoto. Los clientes y servidores se comunican usando XML, lo cual permite que interactúen máquinas de muy distinta naturaleza y diferentes sistemas operativos. El servidor expone sus métodos a los posibles clientes, indicando los parámetros esperados así como el formato de respuesta mediante WSDL (Web Services Description Languaje).

Ruby on Rails

Ruby on Rails es el boom del momento, y se puede decir que ha puesto de cabeza el mundo del desarrollo en web. Todo el mundo que lo ha experimentado tiene solo buenas cosas que decir de este Framework. Revisemos un poco en qué consiste.

Ruby on Rails (o Rails) es un framework programado en el lenguaje orientado a objetos "Ruby" inspirado en Perl y Phyton (Ruby=ruby, Perl=perla) y está pensado para obtener resultados rápidos, basándose en un patrón de diseño *Model View Controller* (MVC). Conceptualmente además abraza dos principios básicos: DRY y *Convención sobre Configuración*. El primer principio, DRY significa en inglés *Don´t Repeat Yourself*, o en una traducción literal "No te repitas a ti mismo" y se refiere a que cada porción de lógica o conocimiento debe estar en un solo lugar, generalmente dictado por las convenciones de una arquitectura MVC. *Convención sobre Configuración* es crucial también, y significa que Rails provee un default inteligente para casi todos los aspectos que se necesitan para hacer andar la aplicación. Si se siguen estos defaults se puede concentrar en aspectos puntuales de la aplicación a desarrollar y se la tendrá andando rápidamente.

Model View Controller

Dijimos que Rails se basa en un diseño Model View Controller, pero ¿qué es esto exactamente? Es una arquitectura en donde se desea separar el sistema en tres partes: Modelos, Vistas y Controles, como se detalla en la figura:

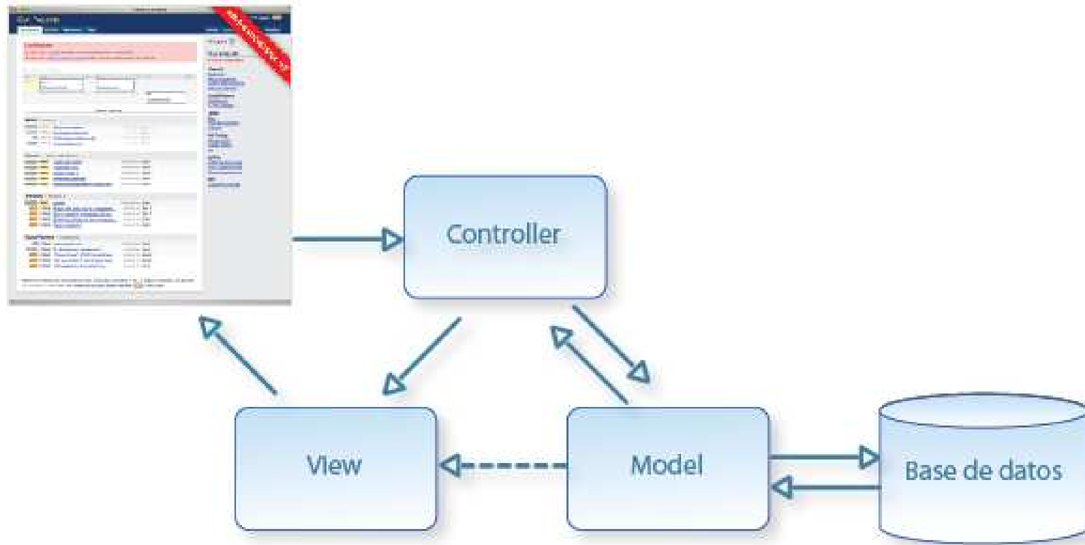


Ilustración 20 - Arquitectura MVC

El *modelo* es responsable de mantener el estado de la aplicación, ya sea por medio de interacción con la base de datos o alguna forma menos permanente. No solo contiene datos, sino que también se encarga de aplicar todas las reglas de negocio que sean necesarias.

Las *vistas* se encargan de dar formato a la salida que se le presentará al usuario. Puede ser por ejemplo una lista de libros para que el usuario seleccione alguno. Aunque puede presentar varios elementos ingreso de datos, la vista nunca maneja esta información de entrada. Eventualmente pueden obtener información para desplegar directamente desde el *modelo*. Esta acción se representa como una flecha punteada en la figura.

Los *controladores* (Controllers) sincronizan toda la aplicación. Reciben eventos del exterior del sistema, ya sea input de usuario u otra fuente, interactúan con el *modelo* y despliegan una *vista* adecuada al usuario.

La primera vez que uno se topa con la arquitectura MVC podría pensar que es sólo un nuevo acrónimo de tres letras, tan populares hoy en día en informática, pero que en realidad es el conocido modelo de tres capas, en donde teníamos separadas las capas de presentación, lógica del negocio y datos. La diferencia principal entre ambos modelos se puede decir que es topológica. En el modelo de tres capas la capa de presentación no puede interactuar directamente con los datos, debe pasar necesariamente por la capa de negocios. En la arquitectura MVC, el *modelo* puede presentar datos invocando directamente una *vista*. Es decir, el modelo de tres capas es lineal y el modelo MVC se puede decir que es triangular, como se aprecia en la figura anterior.

Otros aspectos importantes a considerar de Ruby on Rails se listan a continuación

- Soporte integrado para crear o consumir *Webservices*
- Capacidad para recibir y manejar email
- Unidad completa de testeo integrada y fácil de usar
- Separación de ambientes de desarrollo, test y producción
- Se creó a partir de una aplicación comercial real (www.basecamphq.com) por lo que al usarlo se está partiendo de la base de una buena aplicación.

Algunas críticas

A pesar de todas sus virtudes, se pueden hacer algunas críticas a Rails, no todas ellas técnicas, pero las comentaremos de todas maneras. Estas críticas han sido tomadas de [25].

Poco soporte de los proveedores de internet

No hay demasiados proveedores de internet que tengan soporte para Ruby on rails, en caso de querer contratar un servidor compartido. Se necesita que soporte Ruby, que tenga el driver para la base de datos que se necesite y que ofrezca soporte para *fastCGI* o *lighthttpd*. Un comentario extra para esta crítica… aunque se podría decir lo mismo de Java EE, no es lo mismo ya que Java EE en el servidor está orientado a empresas que seguramente tendrán un servidor propio. En el caso de Rails se está apuntando a un mercado de sitios con algunas visitas al día, en cuyo caso es más atractiva la posibilidad de PHP que está instalado en prácticamente todos los proveedores de internet.

Capacidades empresariales y escalabilidad poco claras

Este punto no es bien recibido por la gente que gusta de Ruby on Rails, y se refiere a que existen pocas aplicaciones importantes desarrolladas en Rails, y las que existen (*basecamp* por ejemplo) fueron desarrolladas por los mismos creadores de Rails. Puede ser que el problema sea sólo que Rails es demasiado joven aún, pero se puede decir que faltan aún algunas evidencias para asegurar que Rails está listo para el mundo empresarial.

No tiene IDE

Al menos al momento de escribir pareciera no haber un IDE robusto, que tenga completar código, debugger, integración con software de control de versiones, administración del proyecto, etc. Al parecer lo más cercano que existe es un plug-in para Eclipse llamado RadRails, pero no es suficientemente robusto según algunos comentarios, no he tenido el placer de usarlo personalmente.

PHP

Antes de saber que es PHP uno podría preguntar ¿Qué significa PHP? Una respuesta común, es "**H**ypertext **P**re**P**rocessor". Pero en ese caso la sigla debería ser HPP y no PHP. Investigando un poco más [26], las siglas al parecer vienen de los orígenes de PHP, que partió siendo un proyecto de Rasmus Ledmorf (1994) con el cual deseaba mantener su página personal y recabar algunas estadísticas. A estas herramientas les llamó *Personal Home Page Tools*, que cuadra más

con las siglas PHP. Olvidándonos de la semántica de PHP, PHP es un lenguaje que permite generar dinámicamente contenido HTML para ser consumido por un navegador, mediante un pre-proceso en el servidor. Allí, en base a algún criterio como usuario conectado o elemento seleccionado, se crea dinámicamente la página HTML. Otras herramientas que conceptualmente hacen lo mismo pero usando otro lenguaje, son ASP (Active Server Pages) y JSP (JAVA Server Pages). PHP lleva ya tiempo en el mercado y ha evolucionado de un proyecto personal a un lenguaje robusto que ya va en la versión 5. Sin duda es una de las soluciones más usadas en Internet, pudiendo correr en casi cualquier plataforma que deseemos, se puede usar sin tener que pagar licencias, provee conectividad con las bases de datos más comunes sin cargos extra, tiene librerías para crear imágenes dinámicamente y existe extensiva documentación debido a su popularidad.

ASP.Net

ASP.Net es la plataforma impulsada por Microsoft para programar aplicaciones web y es parte de la plataforma .NET. Es gratis desde el punto de vista de que viene incluido con Windows (que no es gratis). No es un upgrade de ASP aunque comparte el nombre y coincide en que es posterior, lo cual lleva a malos entendidos. La herramienta para desarrollar en la plataforma .NET más usada es Visual Studio que tiene distintas modalidades de licenciamiento pero tampoco es gratis. Se requiere de un servidor Windows para correr las aplicaciones desarrolladas en esta plataforma.

J2EE

Es una plataforma usada para programar aplicaciones basadas en el lenguaje JAVA. Está basada en una especificación definida por *Java Community Process* (JCP), y define una variedad de APIs como JDBC, RMI, WebServices, XML, etc. Estas aplicaciones deben correr sobre un servidor de aplicaciones J2EE (como Webshpere, JBoss, etc.) que maneja transacciones, seguridad, concurrencia, escalabilidad, etc. de tal modo que el programador puede concentrarse en la lógica de negocios del proyecto a desarrollar.

La última versión de esta plataforma ha cambiado de nombre de J2EE a Java EE.

EJB

Enterpise Java Beans. Esta sección será breve, ya que al poco tiempo de buscar información sobre esta tecnología nos encontramos con algunas historias de terror. Por ejemplo un artículo llamado "EJB's 101 Damnations" [22] (traducido significa 101 Maldiciones de EJB y en inglés es una referencia a la película de Disney, "*101 Dalmatians*") hace algunas críticas bastante severas relativas a la complejidad de la especificación, la no existencia de una aplicación de referencia hasta ya muy entrado el ciclo de vida de los EJB, de una especificación salida de la imaginación de ingenieros y sin base en problemas concretos, la existencia de un lenguaje de acceso a datos llamado EJB Query Language que no tiene mucho parecido a SQL y sin embargo cumple las mismas funciones, obligando al programador a aprender un nuevo lenguaje [23]. En fin, nada muy alentador como para considerar esta tecnología.

Definición de EJB

Cuando el punto 1.1 de la definición en *Wikipedia* respecto a los EJB dice "Rápida adopción seguido por críticas" [24], no puede ser visto como una buena señal. Sin embargo, las críticas se centraban en la versión 1.0 y 2.0 de la especificación. En la actualidad se tiene la versión 3.0 y se han atacado muchos de los mayores problemas así que tal vez si se pueda considerar a los EJB como una alternativa al momento de seleccionar las tecnologías a usar. Ahora que aclaramos que en este momento en el tiempo (2007), los EJB son una solución factible, sería bueno explicar que es un EJB. Un EJB (Enterprise Java Bean) es una componente en el lado del servidor que encapsula la lógica de negocios y provee acceso a los datos. El espíritu de los EJB y de su especificación era dar una solución común a problemas que siempre se encontraban en aplicaciones empresariales, tales como integridad en las transacciones, un manejo de seguridad común, y dejar que los programadores se concentren en otros temas más específicos al problema que deseaban solucionar. EJB es una dentro de varias APIs que forman parte de la plataforma Java EE (anteriormente conocida como J2EE). La especificación de los EJB detalla como el servidor maneja los siguientes aspectos:

- Persistencia
- Proceso de transacciones
- Concurrencia
- Eventos
- Seguridad
- Instalación de la aplicación
- Publicar Webservices

AJAX

Asynchronous JavaScript y XML. Más que una plataforma o herramienta, podemos definir este concepto como una técnica que nos permite traer información desde el servidor y desplegarla al usuario. Para traer la información en forma asíncrona se usa el objeto *XMLHTTPRequest*, disponible en la mayoría de los navegadores modernos y la información recuperada es desplegada usando JavaScript y accediendo al modelo que despliega el navegador directamente. Este enfoque contrasta con lo que eran las aplicaciones en web tiempo atrás, caracterizadas por recargar la página completa cada vez que se necesitaba desplegar nueva información al usuario. Con esta técnica que nos permite traer información de pedazos sin necesidad de abandonar la pagina actual, se puede aumentar la interactividad, velocidad y usabilidad de la página web.

ColdFusion

Con un nombre tan entrador como "ColdFusion", basta haber oído o leído sólo una vez de esta posibilidad para ya no olvidarla más. ColdFusion es el nombre de una plataforma de desarrollo de software más un servidor de aplicaciones desarrollado por Macromedia y posteriormente comprado por Adobe. La versión actual de ColdFusion (ColdFusion 8) está disponible en una variedad de sistemas operativos, entre los cuales se cuentan OSX, Linux y Windows. Se puede decir que está menos restringido a la plataforma que una aplicación ASP.NET o J2EE ya que puede correr sobre cualquiera de estos servidores de aplicaciones.

ColdFusion tiene la ventaja de que se integra bastante bien con Flash y por lo tanto con Flex, sin embargo tiene una desventaja importante, no es gratis como las otras alternativas. Su precio, dependiendo de la licencia, varía entre U$1000 a U$5000, por lo que no lo podemos considerar para el proyecto. Esto no quiere decir que siempre sea, ya que en proyectos de mayor envergadura puede ahorra tiempo de desarrollo. Hay que analizar caso a caso la situación. Durante el presente desarrollo no esperamos tener demasiada lógica en el servidor, sólo algunas operaciones de actualización o lectura de datos, por lo que privilegiamos una solución que sea económica y rápida de aprender.

Conclusiones tecnología web

A pesar del temor ante tantas nuevas palabras y tecnologías, la decisión tomada respecto a que herramienta se usará resulta en una tecnología relativamente antigua: PHP. Los motivos son su alta presencia en los proveedores de internet, su tiempo en el mercado lo hace una solución ya probada y además el proyecto que estamos desarrollando no precisará demasiada sofisticación en el servidor, además de ingresar registros en la base de datos y leerlos, por lo que necesitábamos algo que se pudiera implementar rápidamente como PHP. En particular, se ha usado la versión 5.2.4 de PHP.

Base de datos

La base de datos que se ha elegido para el desarrollo es MySQL (versión 5.0). Las otras posibilidades eran:

- Oracle
- SqlServer
- PostgreSQL
- MySQL

Entre ellas, se descartan de inmediato Oracle y SqlServer debido a que se debe pagar por su uso. Entre PostgreSQL y MySQL no hubo razones técnicas para elegir, ambas proveían las funcionalidades que necesitaba: poder ser usadas con PHP y manejo de transacciones (para la instancia posterior de bloquear un diagrama) y además ambas permiten su uso sin cobro. La razón que inclinó la balanza a favor de MySQL fue un deseo personal de conocer esta base de datos ya que había escuchado que se usaba en algunos sitios que frecuento, como *Digg* y *Wikipedia*.

Ejemplos usados para Sitio Web

No estaba muy seguro si incluir esto como sección o simplemente citarlo como bibliografía. Al final me incline por dedicar una parte de la memoria a dos ejemplos que me sirvieron muchísimo para aprender Flex (lenguaje que desconocía absolutamente) y que me sirvieron de base para muchos de los aspectos implementados en el sitio. Se trata de una tienda virtual y de un explorador de primitivas.

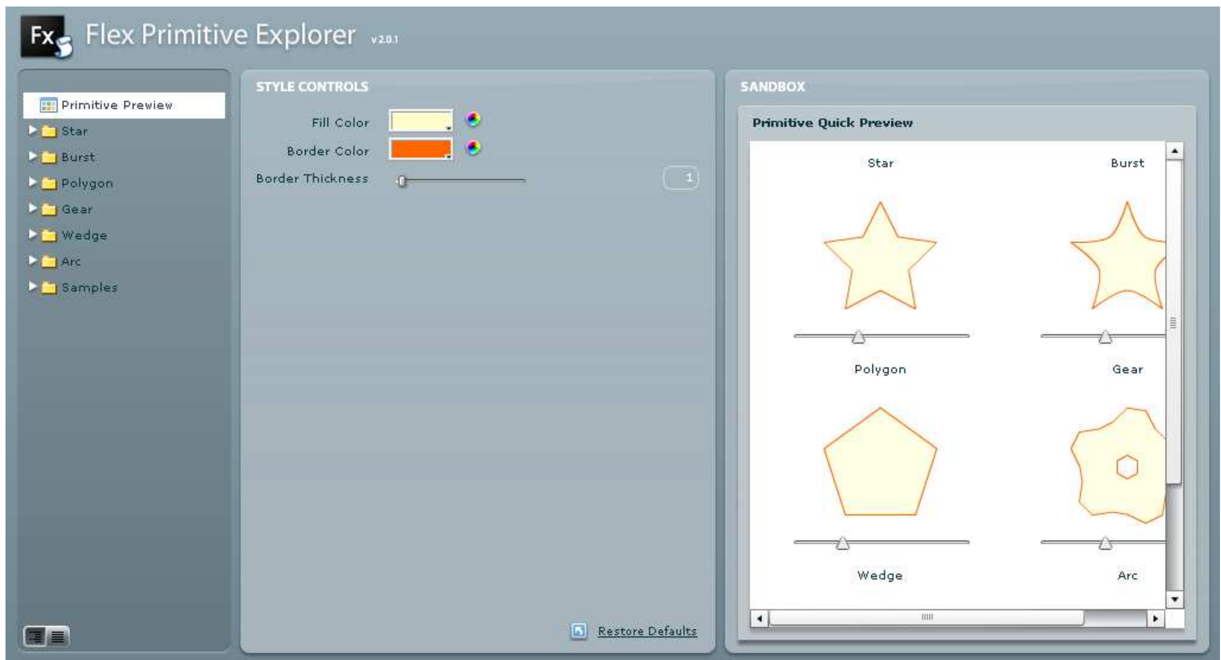


Ilustración 21. Explorador de Primitivas Flex

A decir verdad, después de ver la aplicación "Explorador de Primitivas" quede encantado con los elementos de la interfaz de Flex y además quedaba claro que era factible implementar diagramas usando este lenguaje, ya que las figuras eran exactamente del tipo que me interesaba tener. En un comienzo partí modificando esta aplicación para la sección de diagramación, aunque al poco tiempo cambie el enfoque de las figuras ya que las que necesitaba tenían más funcionalidad y preferí partir de cero. Las pantalla de dialogo para ingreso de propiedades me fueron de mucha ayuda y las propiedades básicas están tomadas de aquí. Esta aplicación está en internet a modo de ejemplo, con su código disponible y fue hecha en base a un ejemplo de Adobe por "*Flexible Experiments*" [19].

La segunda aplicación es directamente un ejemplo de Adobe, se trata de la "Tienda Flex" y es un esqueleto de una tienda virtual.

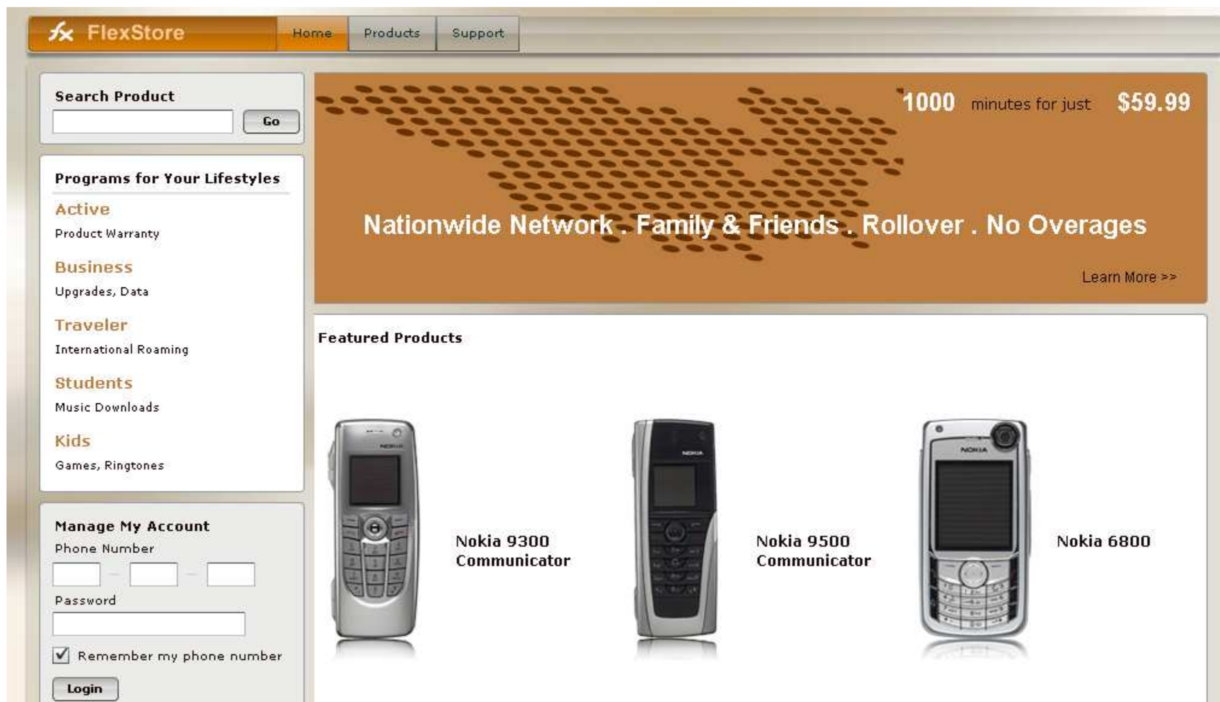


Ilustración 22. Tienda Flex.

La funcionalidad es limitada en este ejemplo, el *login* que se ve en pantalla no funciona y no existe interacción con datos dinámicos. Sin embargo, me sirvió mucho para la implementación de la parte de navegación de los diagramas y vistas en miniatura ha sido modificado a partir de este ejemplo [20].

Ambas aplicaciones son una lección de humildad para quién las estudia y revisa su implementación. Sumamente ordenadas, altamente orientadas a objetos, métodos cortos y claros, códigos escritos casi sin necesidad de comentarios y sin embargo cuando los hay, estos no son superfluos ni innecesarios sino que de gran ayuda.

Las iconografías usadas dentro del sitio web corresponden a la librería "Silk Icons" [21], a disposición del público bajo licencia *Creative Commons*.

4. Sitio Web

En este capítulo se revisará el sitio web desde un punto de vista funcional y se comentarán algunos aspectos de su diseño. En capítulo aparte se revisa la funcionalidad para editar diagramas. Resumamos un poco antes de iniciar que tecnologías se usaron y qué objetivos se perseguían.

La función del sitio web es permitir crear diagramas en línea, por ejemplo Redes de Petri, y permitir la colaboración sobre los diagramas de distintos usuarios en forma asíncrona. Se tendrá la posibilidad de tener documentos compartidos para se puedan editar por un usuario, o se podrán usar Stick-Ons como medio de colaboración. La interfaz del sitio web se construirá usando el *framework* Flex, que es una plataforma nueva para crear aplicaciones basadas en Flash. En el servidor se usarán servicios escritos en PHP para almacenar y recuperar información desde una base de datos mySQL. La arquitectura del sistema es bastante simple y se puede ilustrar de la siguiente manera:

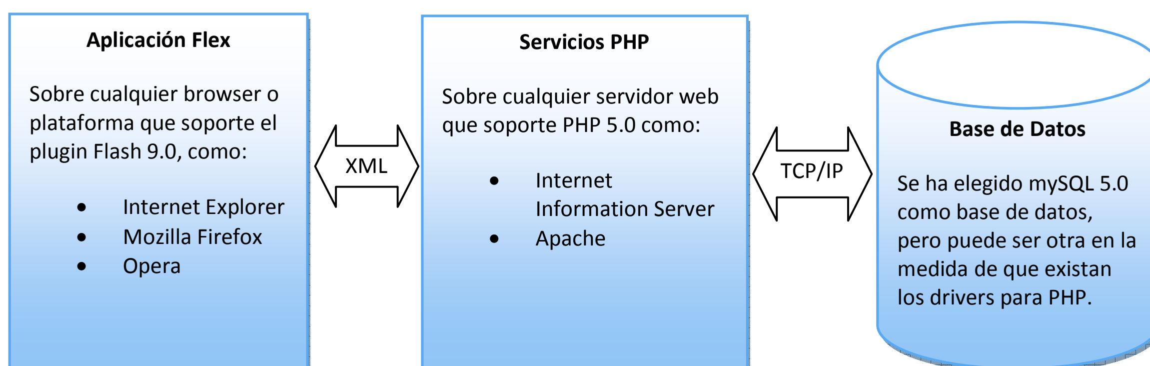


Ilustración 23. Plataforma.

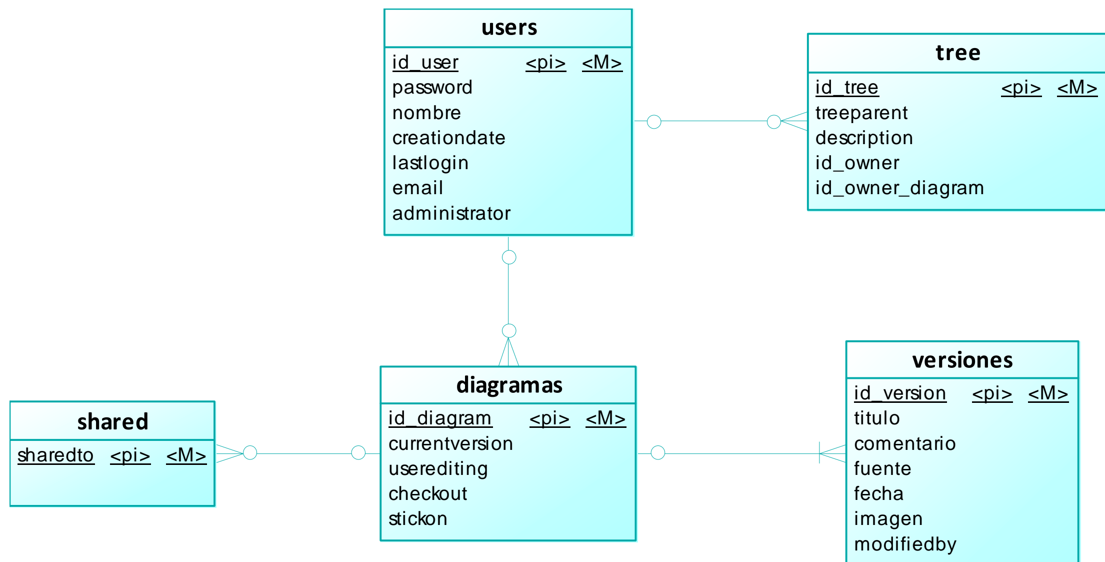
La figura representa las 3 capas lógicas que forman nuestro sitio. Inicialmente está la aplicación Flex, que es descargada localmente a nuestro browser con el plugin de Flash instalado (Internet Explorer, Firefox, etc). Esta aplicación invocara servicios web desarrollados en PHP y que existen en el servidor web hacia donde nos conectamos. La información entre cliente y servicios es transmitida en formato XML. Los servicios PHP guardan y recuperan información desde una base de datos mySQL usando el driver nativo de PHP para mySQL.

Un comentario respecto al uso de Flex (Flash) en nuestro sistema. Muchas veces he sentido que la palabra Flash es un poco mirada en menos por los desarrolladores web. Es una impresión que no comparto demasiado, sin embargo sí hay una crítica fuerte que se le puede hacer al uso de Flash en mi opinión, y es que sus contenidos quedan ocultos a los robots que indexan los contenidos en Internet (ya sean de Google, Yahoo u otro), lo cual en definitiva se traduce en que el sitio desarrollado con tanto esfuerzo no es encontrado cuando alguien busca lo que ofrecemos. Por ejemplo, hace poco tiempo atrás el diario “Las Últimas Noticias” cambió el contenido de su sitio web (www.lun.cl) de HTML estándar a Flash. No comparto dicho cambio ya que va contra el espíritu de tener el diario en internet, y eso es que más y más gente los lea. Sin

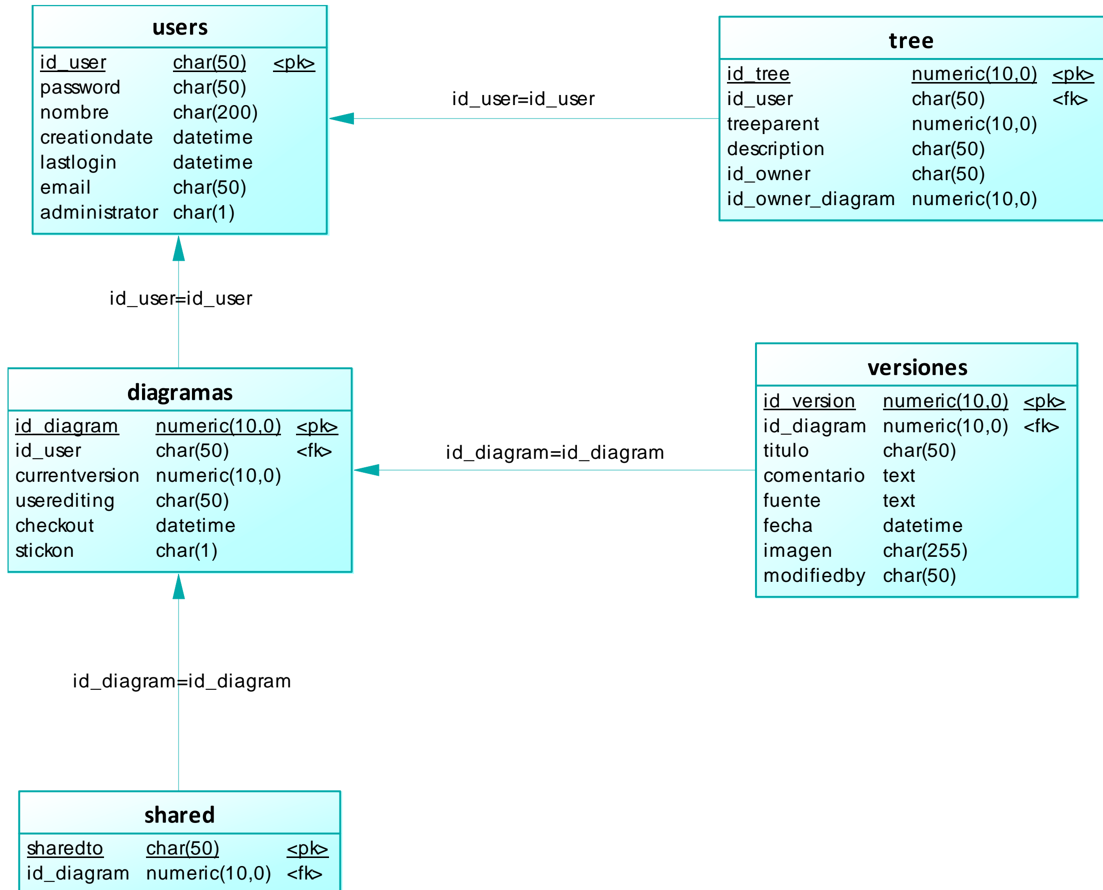
embargo, en el caso de nuestro sitio, los contenidos que desarrollaremos están tras un login de usuario, que requiere conocer un usuario y password, por lo que independiente del lenguaje o tecnología elegida para el desarrollo, estos contenidos no deberían ser visibles para buscadores como Google, y en caso de serlo sería un problema de seguridad. Por eso pienso que esta crítica que a mí me parece fundada en general, no aplica a nosotros.

Modelo de datos

A continuación se presenta el modelo de datos usado para guardar la información relativa al sitio web, sus usuarios y las distintas versiones de los diagramas en la base de datos:



El modelo físico de datos se generó automáticamente a partir del modelo lógico usando el software *PowerDesigner*:



Discusión del modelo

Veamos en palabras la idea del modelo. En primer lugar tenemos la entidad **users**, que almacena cierta información relevante de la persona como nombre, email, password, etc., y le asigna un nombre único dentro del sistema. Este usuario puede tener varios diagramas y además puede crear categorías (en **tree**). Cada uno de los diagramas tiene al menos una versión (en **versiones**) y puede ser compartido a distintos usuarios (**shared**).

El modelo de datos del sistema es bastante simple principalmente por una decisión de diseño. Cuando se está trabajando con un diagrama dentro de la aplicación podemos tener una alta cantidad de elementos, conectores, distintos niveles de profundidad que a su vez tienen otros diagramas, colores, tipos de bordes, etc., o sea, un número de elementos que de guardarlos como tablas aumentaría bastante la complejidad. Sin embargo, cada elemento gráfico implementa un método "*toXml*()" el cual retorna la definición de sí mismo en formato XML, con información como la posición, tamaño, color, etc. Si recorro cada uno de los elementos se

puede tener la definición del diagrama como XML y es exactamente eso lo que se hacemos. Con esta definición en XML que en el fondo es texto ordenado, grabamos en la tabla "*versiones*" en la columna "*fuente*" este XML que representa el diagrama. Cuando el usuario pide editar un diagrama, se recupera esta información y con otro método, "*fromXml*()", se recrea el diagrama.

Este enfoque es poco ortodoxo, pero en mi opinión tiene más ventajas que desventajas en la presente situación. En primer lugar, los diagramas tienen una enorme cantidad elementos y de atributos lo cual obligaría a que además de tener muchas más tablas y relaciones (con el consiguiente aumento en la complejidad que esto implica), si quisiéramos agregar algún detalle nuevo, como que aparezca un *tooltip* cuando pasamos el mouse por el elemento, vamos a tener que alterar el modelo de datos, modificar el servicio que graba la información, actualizar la posible documentación, etc. Sin embargo si este atributo viene dentro del XML que representa al elemento no hay que modificar nada.

Aunque esto pareciera ser miel sobre hojuelas, hay un problema conceptual que no se debe menospreciar. Haciendo esto perdemos las ventajas que nos ofrece una base de datos relacional, al menos en lo que respecta al detalle del diagrama. ¿El usuario Peter cuantos diagramas con más de 30 elementos tiene? No lo vamos a saber usando SQL sin tener un buen dolor de cabeza.

Existe una alternativa a grabar en formato XML la definición del documento como texto. Podemos serializar en el programa el objeto que mantiene el diagrama, y luego esta información binaria la podemos enviar al servidor y grabarla, para posteriormente des-serializar. Ya estamos enviando información binaria al servidor (codificada en base64) cuando grabamos un diagrama: en el cliente se genera una imagen PNG en memoria (binaria) y se envía al servidor. Con esto tenemos la ventaja de grabar exactamente el diagrama que tenemos actualmente, pero no existe ninguna posibilidad de usarlo o integrarlo en otro sistema, como si es el caso de grabar como XML.

Login, registro de usuarios y más.

Ahora que comenzamos con una revisión del sitio desarrollado, nos gustaría presentar el nombre y logo del sitio web que se está construyendo: *Diagster*. Idealmente registraríamos el nombre de dominio diagster.com que está disponible. El logo del sitio comprende letras tipo "*Arial*" de colores sobre una flecha a modo de subrayado.

Y la pantalla asociada de registrarse, que se despliega al presionar el link “Desea registrarse”

Las pantallas de Login y de Registro nos dan la oportunidad de presentar algunos de los conceptos, nuevos para mí al menos, de desarrollar en Flex. Generalmente en una aplicación la interfaz va cambiando dependiendo de la actividad que estemos ejecutando y del avance del usuario en la aplicación, por ejemplo al cambiar de una vista de navegar a una vista de detalle, o en nuestra aplicación, al cambiar desde la pantalla de *Login* a la de *Registro*. Para facilitar dichas transiciones, en Flex existe el concepto de Vistas de Estado (*view states*), que para la misma pantalla nos permite agregar o cambiar elementos dependiendo del estado en que este el usuario. La pantalla de Login y de registro es la misma, pero con estados distintos. Esto permite organizar mejor la aplicación y simplifica lo que de otro modo se podría volver un manejo complicado de eventos para ir cambiando las vistas. Además, se pueden definir transiciones (gráficas) entre las vistas, lo cual no es esencial pero si nos da una cosa extra y hace más placentera la experiencia del usuario, siempre teniendo cuidado de no abusar.

El estado base del sistema de login/registro es la pantalla inicial de login, que es generada con el siguiente código, que crea un panel con 2 textos de entrada, login y password, así como una barra de botones donde está el link para registrarse y un botón aceptar:

```
    <mx:Panel
        title="Login" id="loginPanel"
        horizontalScrollPolicy="off" verticalScrollPolicy="off" >
    <mx:Form id="loginForm">
        <mx:FormItem label="  Usuario:" >
            <mx:TextInput id="username"  />
        </mx:FormItem>
        <mx:FormItem label="  Password:" >
            <mx:TextInput id="password" displayAsPassword="true" />
        </mx:FormItem>
    </mx:Form>
    <mx:ControlBar>
        <mx:LinkButton
            label="Desea registrarse?" id="registerLink"
            click="currentState='Register'"
        />
        <mx:Spacer width="100%" id="spacer1"/>
        <mx:Button label="Login" id="loginButton"
click="login_user.send();"/>
    </mx:ControlBar>
    </mx:Panel>
```

Al presionar el link de registro se cambia el estado a *Register*. Los estados definidos para esta parte de la aplicación son dos:

- default
- Register

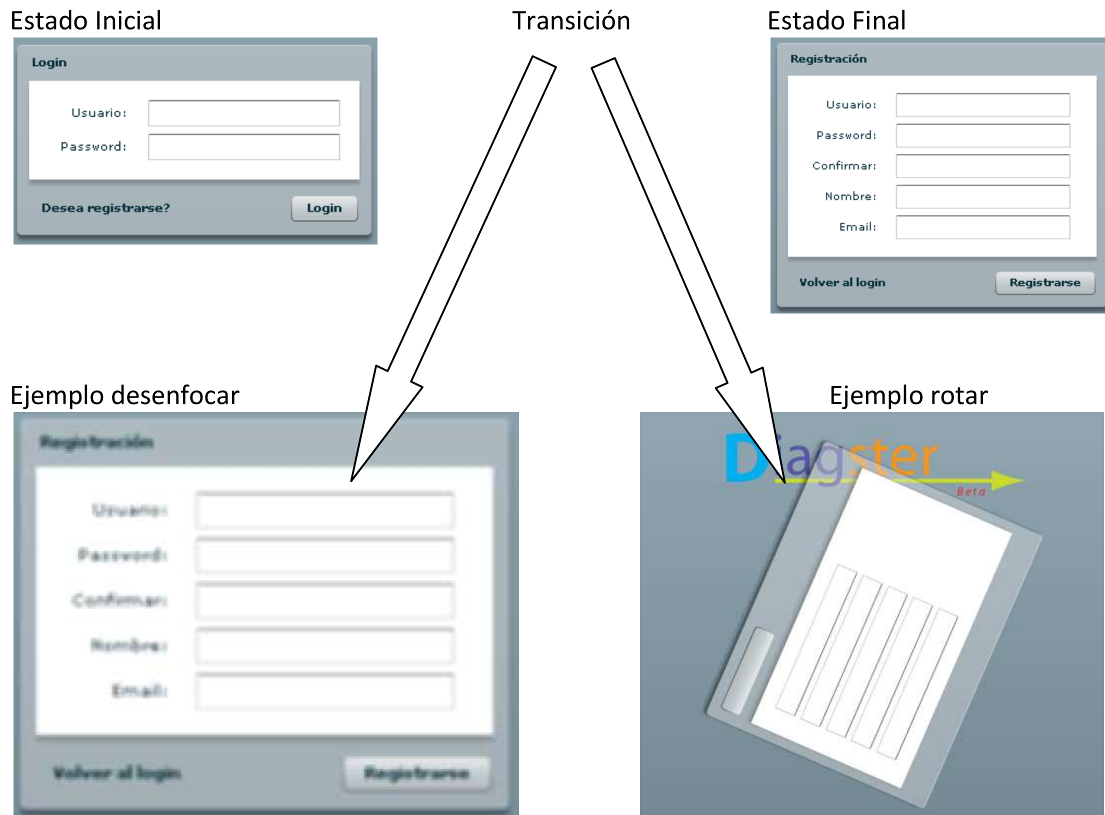
Donde el *default* es el login inicial, y el estado register corresponde a la misma pantalla pero con más campos de ingreso de datos y sin el link de registro, veamos el código que define esto.

```
<mx:states>
    <mx:State name="Register" basedOn="">
        <mx:AddChild
            relativeTo="{loginForm}"
            position="lastChild"
            creationPolicy="all" >
            <mx:FormItem id="confirm" label="Confirmar:">
              <mx:TextInput id="passconfirm" displayAsPassword="true" />
            </mx:FormItem>
        </mx:AddChild>
        <mx:AddChild
            relativeTo="{loginForm}"
            position="lastChild"
            creationPolicy="all" >
            <mx:FormItem label="Nombre:">
                <mx:TextInput id="nombre" />
            </mx:FormItem>
        </mx:AddChild>
        <mx:AddChild
            relativeTo="{loginForm}"
            position="lastChild"
            creationPolicy="all" >
            <mx:FormItem id="emailItem" label="Email:">
                <mx:TextInput id="emailInput" />
            </mx:FormItem>
        </mx:AddChild>
        <mx:SetProperty target="{loginPanel}"
        name="title" value="Registración"/>
        <mx:SetProperty target="{loginButton}"
        name="label" value="Registrarse"/>
        <mx:RemoveChild target="{registerLink}"/>
        <mx:AddChild relativeTo="{spacer1}" position="before">
         <mx:LinkButton label="Volver al login" click="currentState=''"/>
        </mx:AddChild>
        <mx:SetEventHandler target="{loginButton}"
         name="click" handler="register()" />
    </mx:State>
</mx:states>
```

Los dos códigos presentados nos permiten cambiar de la funcionalidad de login a la de registro de una manera ordenada y así podemos reutilizar código. Además, como mencionamos, al definir estados tenemos la posibilidad de usar transiciones. Esto es sumamente simple y da un acabado mejor a la aplicación. Las transiciones se definen de la siguiente forma:

```
        <mx:transitions>
       <mx:Transition fromState="*" toState="*">
          <mx:Blur target="{loginPanel}" />
       </mx:Transition>
    </mx:transitions>
```

Y esta transición será invocada cada vez que se invoque un cambio de estado. Intentaremos ilustrar algunos de los cambios de estado que se pueden ejecutar con sólo cambiar el tipo de transición.



La transición final elegida para el login del sitio fue una más o menos sutil en este contexto de animaciones rotantes. Se eligió la transición "resize" que cambia en forma continua el tamaño de uno de los paneles al tamaño del otro.

Otro aspecto que nos agrado de Flex fue la forma de informar de errores en los datos de entrada. Por ejemplo, queremos validar que el usuario efectivamente ingrese su nombre de usuario y password antes de presionar el botón de login. El código para lograr esto es muy fácil, veamos:

```
private function validateUserPass():Boolean{

    var bResult:Boolean = true;
    if (username.text == "") {
        username.errorString = "Por favor ingrese un nombre de usuario";
        username.setFocus();
        bResult = false;
        }

    if (password.text == "") {
        password.errorString = "Por favor ingrese una password";
        password.setFocus();
        bResult = false;
        }

    return bResult;
}
```

Si se intenta ingresar al sitio sin poner usuario ni password el resultado es esto:

Ilustración 24. Ventana de login

Una sutil indicación de que hay un problema. Los bordes han sido coloreados en rojo, y si se posiciona el mouse sobre alguno de los cuadros en rojo, nos indica cual es el problema:

Ilustración 25. Ejemplo validación Flex

Todo con muy poco esfuerzo. De hecho, tal vez de haberlo hecho uno mismo no habría terminado con una implementación tan robusta. Por ejemplo, ¿qué ocurre si no hay espacio a la derecha del campo en rojo con problemas?

Ilustración 26. Variante validación Flex.

En esta situación se ha encogido el área de despliegue del browser, por lo que no es factible mostrar el mensaje a la derecha como es el comportamiento habitual. En ese caso el *tooltip* de error se ajusta solo a una nueva posición, como se aprecia en la figura.

Seguridad ¿Qué ocurre al ingresar al sitio?

Después de revisar con cierto detalle las pantallas de ingreso y registro, debemos entender que ocurre realmente al ingresar al sitio. El esquema de seguridad que se ha elegido es uno bastante simple, no era nuestro objetivo hacer algo demasiado sofisticado respecto a este aspecto, pero no lo podemos dejar de lado tampoco. Nos ayudaremos de un pequeño diagrama para explicar el esquema.

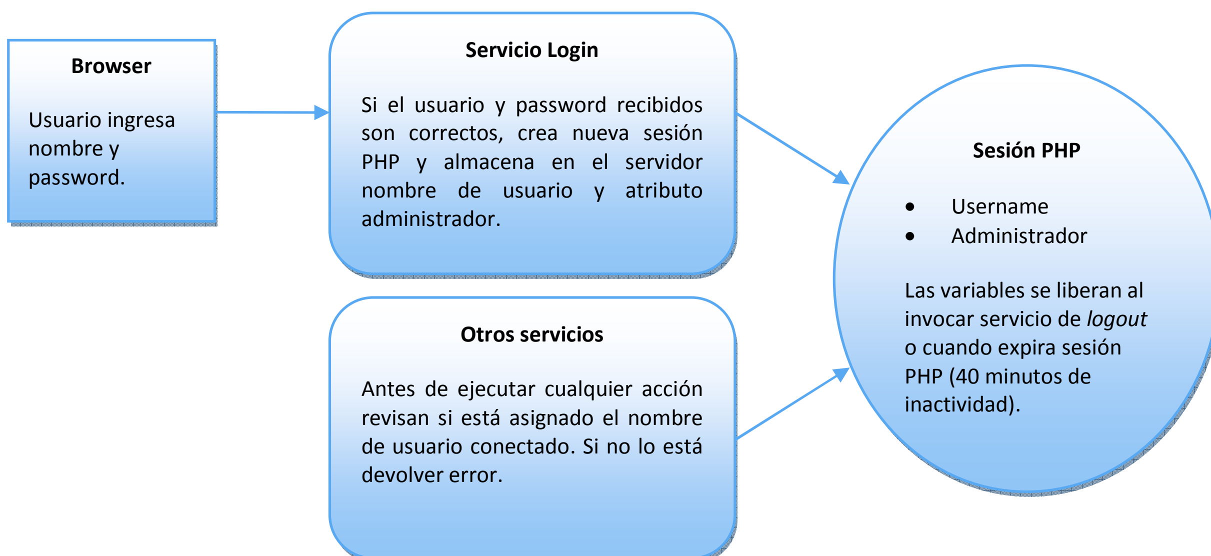


Ilustración 27. Esquema de login y servicios.

Nuestro esquema de seguridad se reduce a que en el evento de que exista un login exitoso, esto es que el usuario digite usuario/password, las envíe al servicio de login y se consulte a la base de datos si existe el usuario y concuerde la password, se creará una nueva sesión PHP y se almacenará en el servidor, en la misma sesión, la información de cuál es el nombre de usuario y si es un usuario administrador. Todo el resto de los servicios que serán invocados con posterioridad deben consultar si el usuario está asignado, y en caso negativo retornar inmediatamente un mensaje de error. La sesión estará activa hasta que se invoque el servicio de *logout* o bien expire por inactividad la sesión PHP en el servidor, lo que sea que ocurra primero.

Cuando no es posible validar al usuario que está intentando conectarse, el servicio retorna un mensaje de error y la sesión en el servidor no es creada. En el cliente, este evento se ve de la siguiente manera:

Ilustración 28. Login incorrecto.

Cuando el servicio de *login* es exitoso, además de crear la sesión en el servidor, se devuelve un mensaje de éxito con lo que la aplicación puede continuar con el siguiente paso, es decir, ingresar al sitio con nuestro usuario validado.

Dentro del sitio

Antes de discutir el sitio "por dentro", quisiéramos presentar a cuatro usuarios que nos ayudarán a ejemplificar las distintas situaciones que necesitemos. Ellos son los usuarios "Admin", que tiene privilegios de administrador en el sitio, y los 3 usuarios normales de nombre "Peter", "Paul" y "Mary". Todos ellos se comportan como usuarios responsables aunque a veces crean algunos diagramas sin mucha semántica, más bien para ejemplificar las posibilidades.

Veamos la pantalla de ingreso y los diagramas que ve Peter al ingresar al sitio.

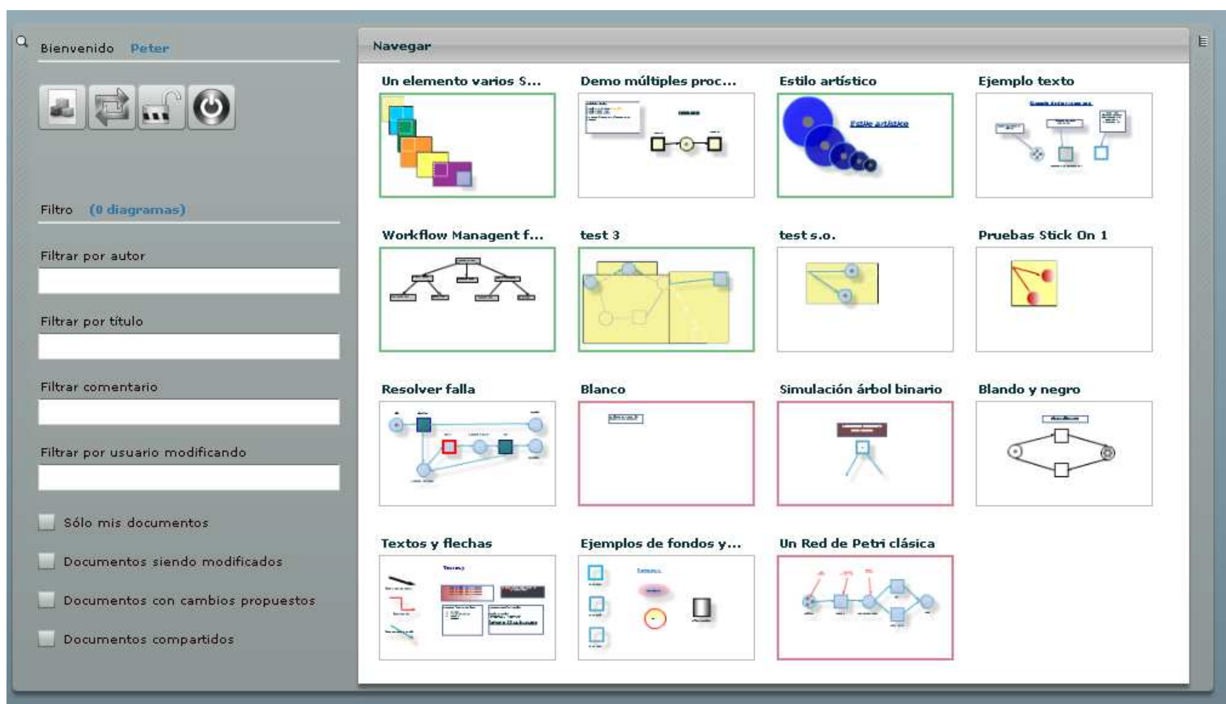


Ilustración 29. Pantalla principal.

En la figura se aprecia lo que será la parte principal del sitio que manejará el usuario (aparte de la sección misma de diagramación). Como se puede ver, tiene una organización parecida a muchas otras aplicaciones, con algunas opciones o herramientas al lado izquierdo, y los elementos sobre los que se trabaja en una parte central-lateral. Tiene un detalle que para mi gusto es fantástico pero que lamentablemente no se me ocurrió a mí (es parte del ejemplo de la tienda virtual), y que consiste en una metáfora de un pizarrón móvil. Al pinchar sobre el borde derecho la parte central se traslada hacia la izquierda y deja ver una nueva ventana de herramientas, en este caso la posibilidad de tener categorías de diagramas. Es una manera ingeniosa de usar el espacio. Tal vez la crítica que se le puede hacer es que es *demasiado* ingeniosa y que el usuario puede que no se entere de esta posibilidad. Es una crítica justa, sin embargo se dan algunos *hints* visuales como los iconos en las partes superiores izquierda y derecha, una textura de agarre que no se puede ver con claridad en la captura de pantalla y un *tooltip* que aparece al pasar con el mouse.

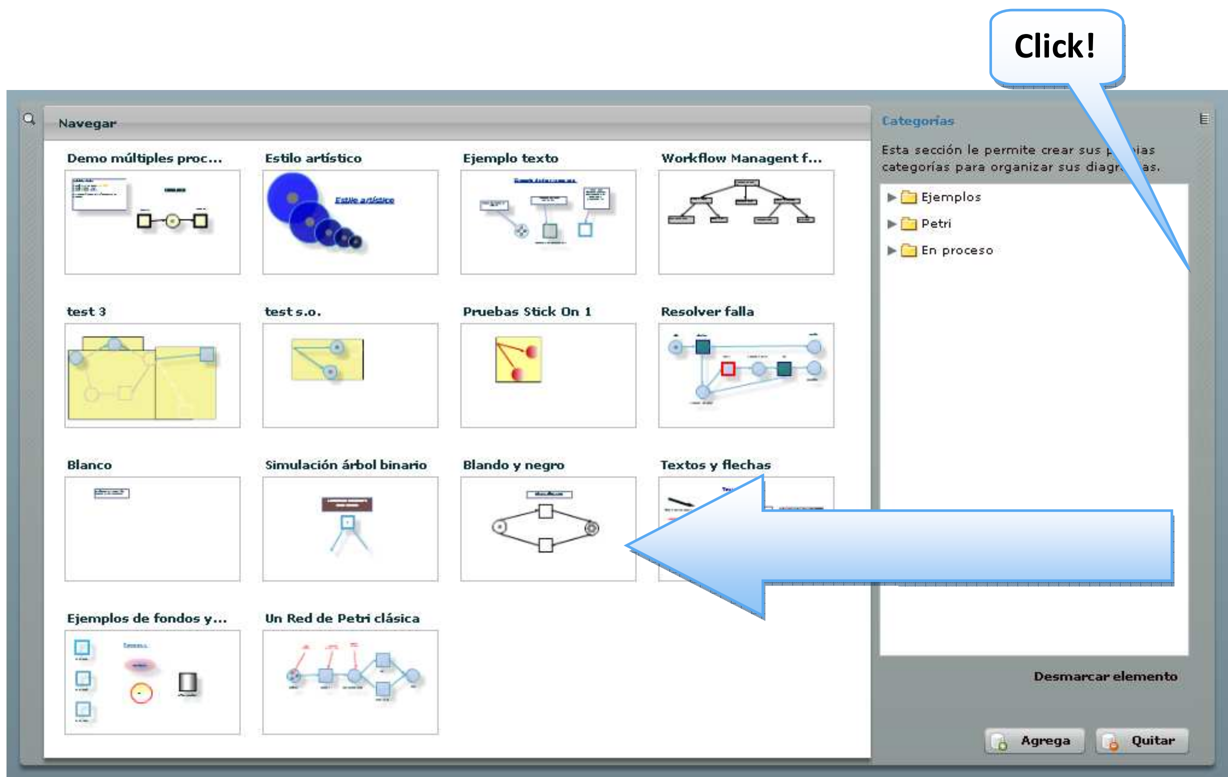


Ilustración 30. Sección categorías.

Sección de controles

En este sector se da una indicación de que usuario está conectado. Además de la cortesía de saludar, tiene utilidad para el caso en que se tenga y utilice más de una cuenta. En seguida vienen cuatro botones que tienen funciones más bien genéricas o que involucran a varios elementos. Los iconos son de colores más o menos fuertes, y en una primera iteración se veían así por defecto:

Tanto color resalta demasiado en mi opinión y no encaja bien dentro del esquema gráfico de la aplicación. ¿Qué hacer en esta situación? Normalmente habría tomado la imagen y usando algún software de edición de imágenes, como Photoshop, se habría generado una versión en blanco y negro de la imagen. Esto tiene un problema directo y es que si necesito incorporar otra imagen tengo que ir a Photoshop de nuevo. Hay una alterativa mucho mejor y ejemplifica la potencia del lenguaje elegido. Se puede usar un filtro que aplicado al ícono de colores lo traspase a una escala de grises, y como en el caso de nuestra aplicación, cuando el mouse este sobre el icono para dar énfasis quitar el filtro, con lo que veremos el elemento actual en colores. Pero ¿cómo debe funcionar dicho filtro? O en otras palabras, ¿Cómo paso una imagen de colores a una escala de grises? Existen al menos 3 formas. Si asumimos que un color es la mezcla de 3 colores primarios -rojo, verde y azul- podemos suponer que si promediamos los 3 colores y reemplazamos la cantidad de rojo, verde y azul por este promedio tendremos un gris equivalente al color original. Este es el primer método, el promedio. Es el más simple pero no es el mejor. ¿Qué tiene de malo? Al ojo humano los colores básicos no son iguales. El azul es mucho más oscuro que el verde por ejemplo. Actualmente se usa un modelo que considera la luminosidad de cada componente de color en lugar de un promedio directo. Es un poco arbitrario cuanto más luminoso es un color que otro. El segundo método para pasar a escala de grises por ejemplo esta calibrado para los antiguos televisores en blanco y negro. El tercer método, usado aquí, esta calibrado para los monitores modernos. En código este método se traduce en definir una matriz y aplicarla como filtro al icono:

```
private var r:Number=0.212671;
private var g:Number=0.715160;
private var b:Number=0.072169;
public var matrixGrayFilter:ColorMatrixFilter = new ColorMatrixFilter
            (
                [r,g,b,0,0,
                 r,g,b,0,0,
                 r,g,b,0,0,
                 0,0,0,1,0]    );
```

Los valores de *r, g* y *b* diferencian los distintos modelos. En el caso del promedio estos valores serian 0.33333 para todos los componentes. Después de tener la matriz basta con aplicar el filtro según el evento, de manera muy simple:

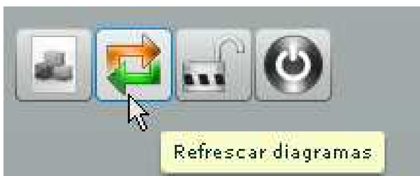
```
        public function applyGrayFilters(e : Event):void {
            e.target.filters = [matrixGrayFilter];
         }

        public function removeGrayFilters(e : Event):void {
            e.target.filters = [];
         }
```

Después de este pequeño paréntesis, continuemos con las funciones de los botones.

Nuevo diagrama

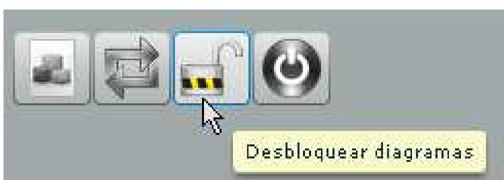
Este botón permite ir al modo de edición con un diagrama en blanco, el cual podernos grabar y del cual seremos los creadores. Tenemos ciertos atributos especiales sobre los documentos que hemos creado nosotros mismos. Podemos compartirlos a otros usuarios, y en caso de que estén siendo modificados (lo cual bloquea el documento), podemos desbloquearlos si lo estimamos necesario.

Refrescar diagramas



Refresca la información desplegada en el navegador de diagramas. Es necesario ya que la información desplegada representa sólo la situación que existía al momento de pedir la información al servidor. En el intertanto, alguien pudo compartirme un documento, o algunos de los documentos compartidos pudo haber entrado en modo de edición, etc. Desafortunadamente, en este tipo de aplicaciones somos nosotros los que debemos preguntar al servidor si ha ocurrido algo, el servidor no puede contactarme sin que yo lo requiera. La única forma de poder tener el estado actualizado es estar pidiendo constantemente la información al servidor cada determinado tiempo, lo cual no es una práctica que sea muy de mi agrado. Por ello la aplicación se refresca ya sea por una solicitud manual o en eventos que sé que existen cambios, como cuando agrego o modifico un diagrama, cuando comparto, cuando desbloqueo, etc.

Desbloquear diagramas



Los diagramas que he creado los puedo compartir con otros usuarios del sitio para que ellos los puedan modificar. Nuestra aplicación por el momento no soporta edición síncrona de diagramas, así es que cuando un usuario desea editar un diagrama este es bloqueado para los demás usuarios. Sin embargo, puede ser que un usuario tome el diagrama y no lo libere más, o

que yo necesite modificar mi diagrama urgentemente, por lo que esta opción me da control sobre el estado de bloqueo de mis diagramas. También puedo usar esta función para liberar diagramas que he estado modificando y sobre los cuales ya no haré más trabajo. En el caso de ser un usuario administrador, esta opción muestra el estado de todos los diagramas y me permite destrabar situaciones que se me reporten.

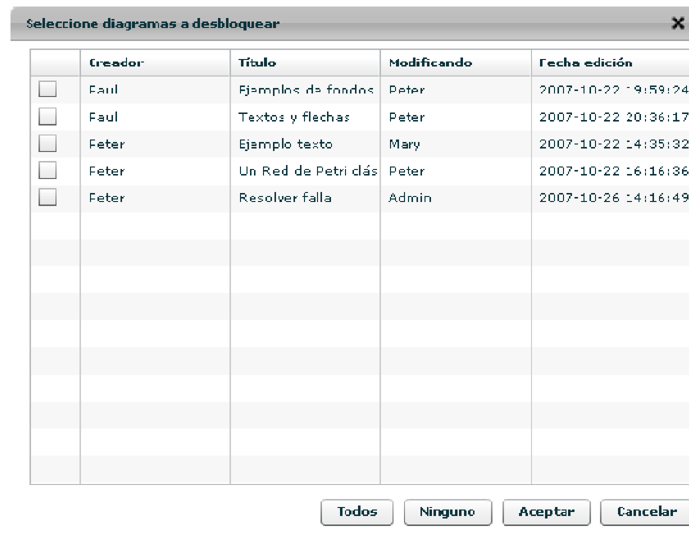
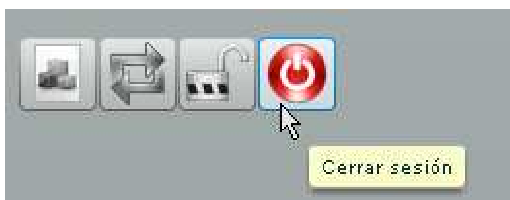


Ilustración 31. Desbloqueo de diagramas.

En este ejemplo se muestra la interfaz de desbloqueo. Muestra el usuario creador del documento, el título, el usuario que está modificando y la fecha de la última modificación. Puede elegir entre desbloquear todos los elementos o un subconjunto de ellos. Nótese que para el ejemplo el usuario que estamos usando es “Peter”. Sin embargo, podemos desbloquear dos diagramas que pertenecen a “Paul”. Esto es porque están siendo modificados por el usuario “Peter” así que es lógico que “Peter” pueda terminar las modificaciones y desbloquear el diagrama, aunque no sea el creador.

Para el caso de un usuario administrador se despliegan todos los diagramas boqueados, independiente del dueño y del modificador.

Cerrar sesión



Esta opción permite salir del sitio. No es obligatorio (y no podemos obligar a) cerrar la sesión al salir, pero es una buena costumbre. Esto libera la sesión que tenemos en el servidor inmediatamente y elimina potenciales problemas de seguridad que se presentarían si no termino mi sesión y dejo el computador abandonado. En este caso podría venir una tercera persona y desde la máquina donde estaba usar los servicios que aún piensan que estoy

conectado. La sesión de todos modos termina luego de cierta cantidad de minutos de inactividad (parámetro del servidor PHP).

Sección de navegación

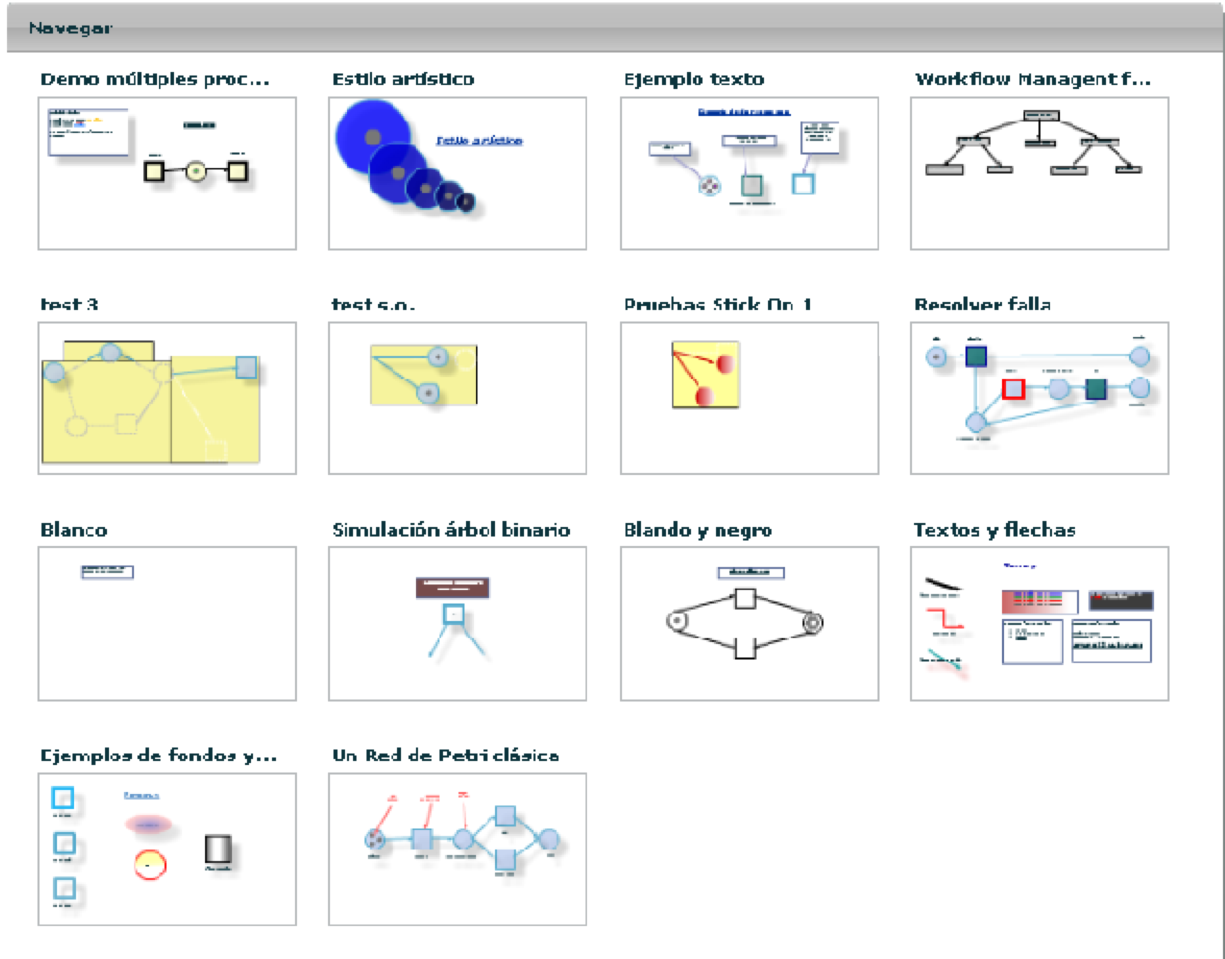
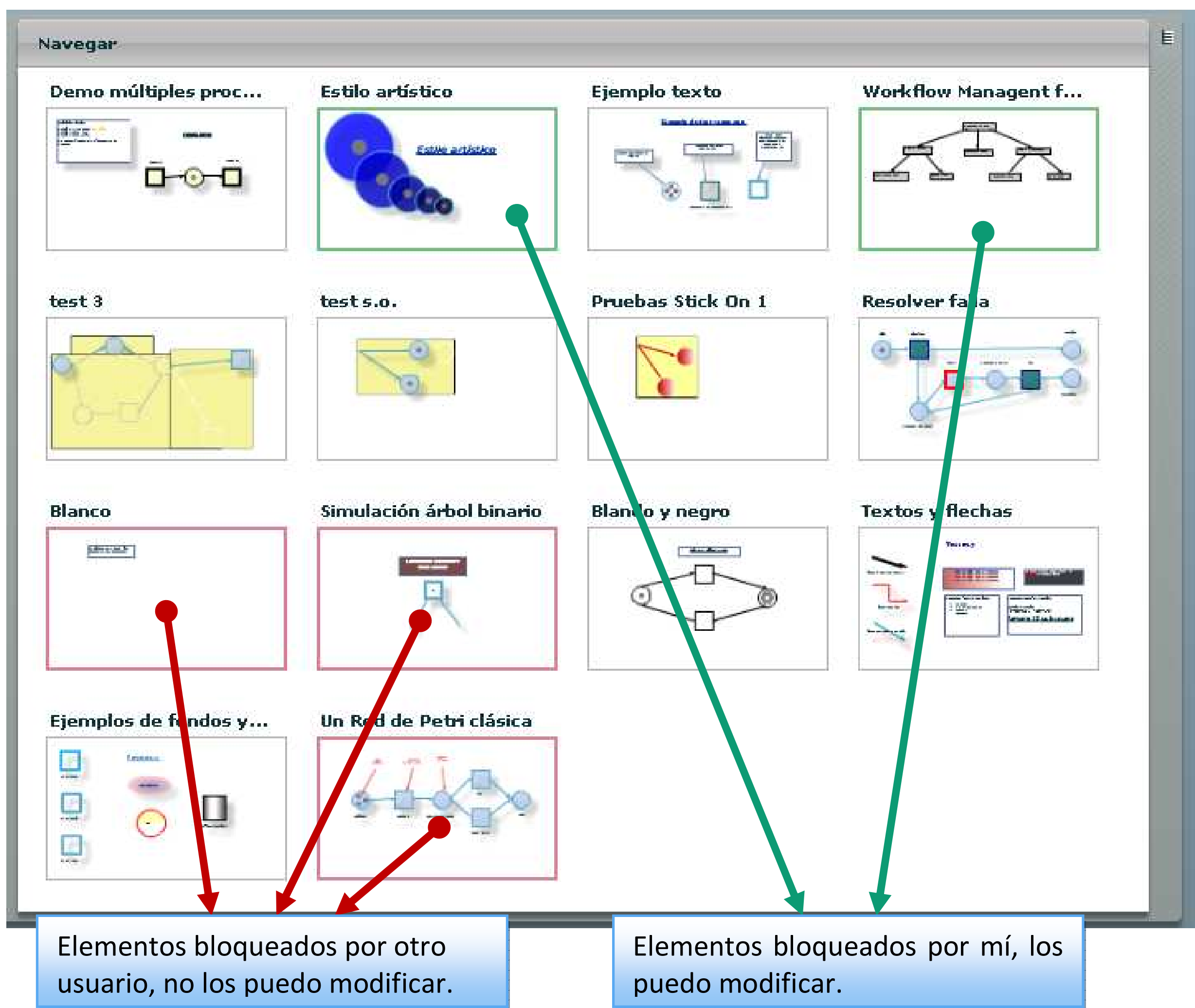


Ilustración 32. Vistas en miniatura.

Esta parte es central dentro del diseño de la interfaz. Despliega los diagramas del usuario y a los diagramas que le han sido compartidos. Como nota al margen, las imágenes de diagramas han sido creadas con la aplicación misma. Dependiendo de la cantidad de diagramas que se necesite desplegar es el tamaño en que se muestran. En pantalla, de una sola vez, caben aproximadamente dieciséis diagramas en tamaño pequeño. Si el número disminuye a una cantidad entre 5 y 9 los iconos de diagramas se muestran un tamaño mayor y con más información llenando 3 columnas. Lo mismo si se tienen entre 1 y 4 iconos. Además, cada unos de estos íconos da un *feedback* visual respecto al estado actual, si está siendo modificado (por mí o por otros). Veamos estos puntos en un poco más de detalle.

Hints visuales de estado



Es conveniente dar información visual respecto al estado de bloqueo de un diagrama sin tener que explícitamente preguntar o enterarnos que está siendo usado sólo cuando lo vamos a modificar. Por ello se implementó un sistema de alertas visuales que consiste en poner un borde de color a los diagramas siendo modificados. Existen dos categorías para este borde que indican una diferencia importante en la práctica. Si no tengo acceso al documento se despliega un color rojo de borde, indicando que otro usuario lo está modificando y no lo puedo editar en estos momentos. Si el color de borde es verde, quiere decir que el documento está siendo usado por mí, lo cual me permite ir a modo de edición con este documento.

Utilización de espacio disponible

Como ya se había mencionado, las vistas previas se ajustan dependiendo del espacio disponible, distribuyéndose en 2, 3 y 4 columnas. Además, si hay más espacio se despliegan en un tamaño mayor permitiendo mostrar información adicional. La cantidad de diagramas desplegados puede variar por varios factores, siendo el principal que se aplique alguno de los varios filtros

sobre los diagramas. También puede variar porque me han compartido o des-compartido diagramas, o el mismo usuario ha creado o eliminado diagramas.

4 columnas

El despliegue en 4 columnas se activa cuando se tienen diez o más diagramas. En esta vista los íconos consisten sólo de una imagen que se ha grabado con la última versión que se ha subido al servidor y el titulo.

Ilustración 33. Vistas en miniatura, cuatro columnas.

3 columnas

El despliegue en 3 columnas es usado cuando se tienen entre 5 y 9 diagramas visibles. En esta vista se tiene una imagen del diagrama un poco mayor y adicionalmente se incorpora la fecha de la última modificación.

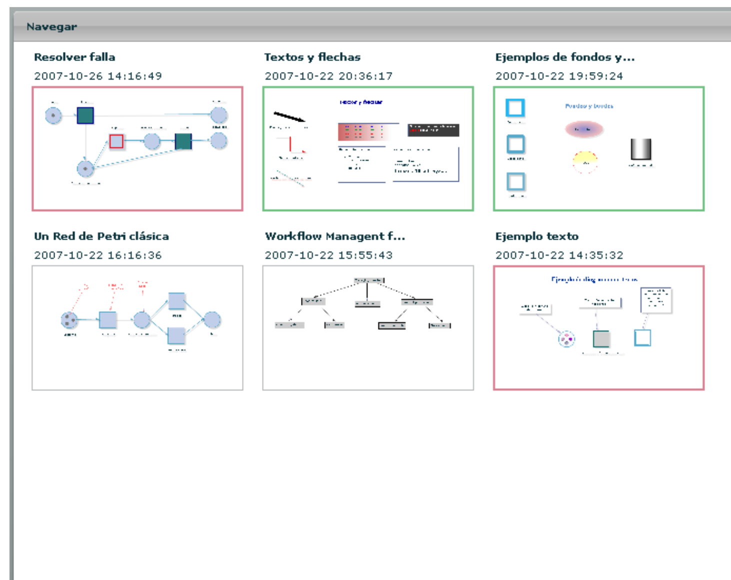


Ilustración 34. Vistas en miniatura, 3 columnas.

2 columnas

La vista en 2 columnas se usa cuando hay entre 1 y 4 diagramas. Los elementos ocupan un cuarto del área disponible, y a la información entregada por la vista en 3 columnas se agrega en la sección inferior parte de la descripción usada al momento de grabar el diagrama.

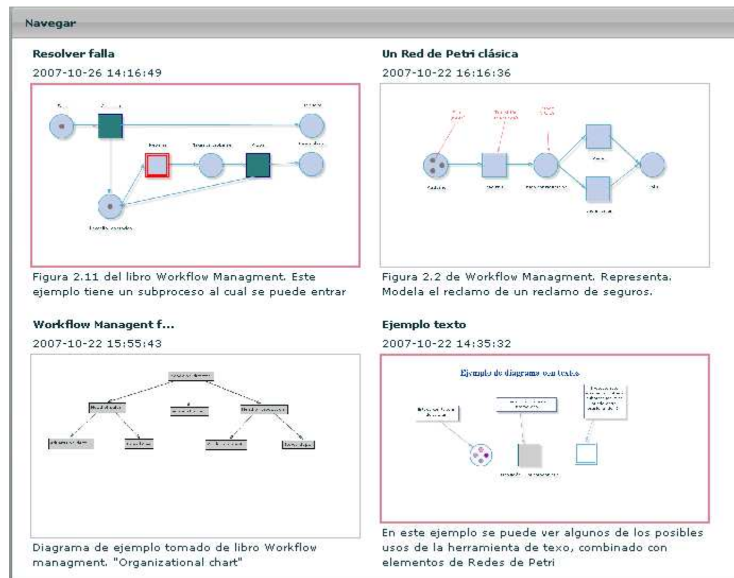


Ilustración 35. Vistas en miniaturas, dos columnas.

Foco y opciones sobre elemento

Cuando el usuario pasa el mouse sobre un elemento del navegador, el estado del elemento cambia y da una indicación visual de que actualmente tiene el foco. Además aparecen varias opciones de operaciones a realizar sobre el elemento en forma de botones. En la siguiente figura se muestra la diferencia sobre un elemento cuando tiene el foco y cuando no.

Las opciones disponibles al tener alguno de los diagramas bajo el mouse o en foco son:

Ver detalle

Esta opción se revisará más adelante ya que nos lleva a una pantalla más compleja y con diversas opciones. Es equivalente a pinchar sobre cualquier lugar de la imagen en foco.

Ver imagen original

Esta opción es muy importante y se tenía planeada desde un comienzo. La aplicación está construida básicamente sobre flash. Ya hemos comentado algunos pro y contra de esta decisión, pero uno de los problemas que presenta el desarrollo nativo en flash es indirecto. Cuando hacemos un diagrama en flash, además de verlo dentro de nuestra aplicación, ¿Qué podemos hacer con él? ¿Podemos usarlo en algún documento Word? ¿Podemos *linkearlo* desde otra página web? ¿Podemos grabarlo como imagen localmente para no depender del sitio web siempre y darle el uso que queramos? En general, lo que hacemos en Flash se queda en Flash. Con esta opción podemos salirnos del mundo de flash y obtener la imagen en formato PNG, reconocido por casi todas las aplicaciones que aceptan imágenes. Podemos grabar la imagen a nuestro disco duro o incluso poner un link en otra página web a esta imagen ya que esta almacenada en el servidor. A continuación presento tres imágenes PNG obtenidas del sitio de esta manera, no han sido modificadas o alteradas en ninguna forma más que el ajuste que hace Word para que calce bien con el ancho de la hoja (espero me disculpen por poner tres imágenes

para ilustrar el punto, pero además sirve para poder ver diagramas que han sido creados en el sitio):

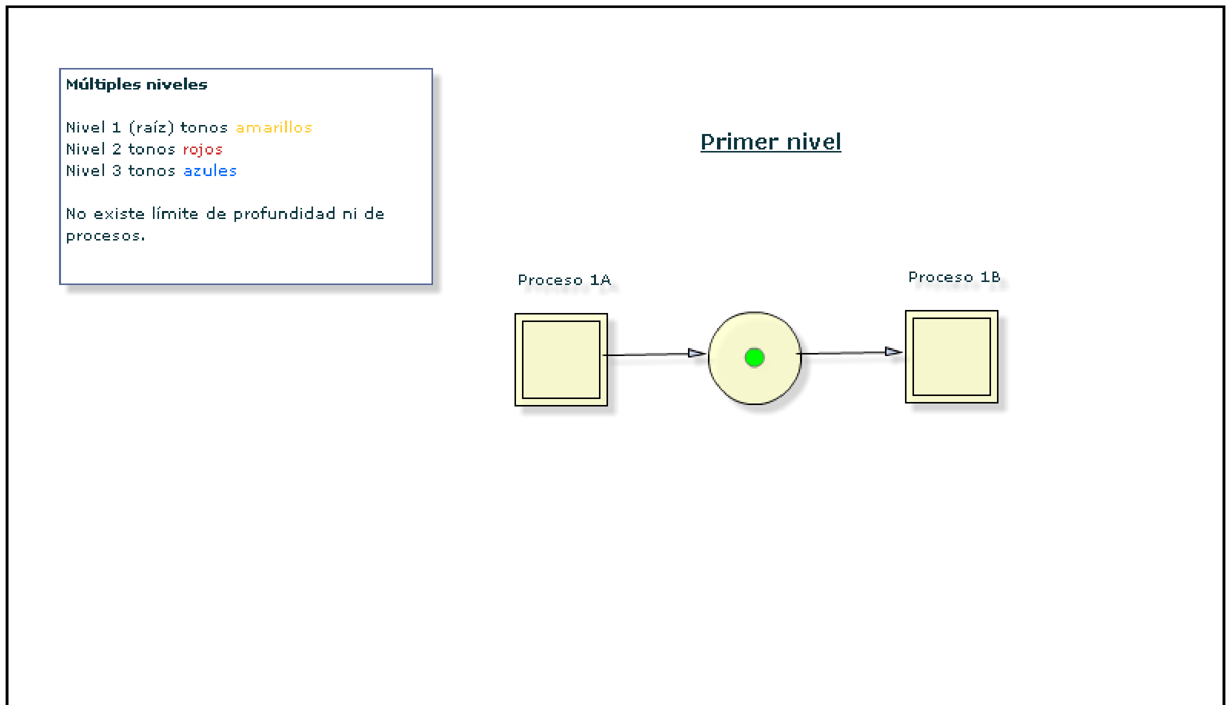


Ilustración 36. Ejemplo imagen PNG generada

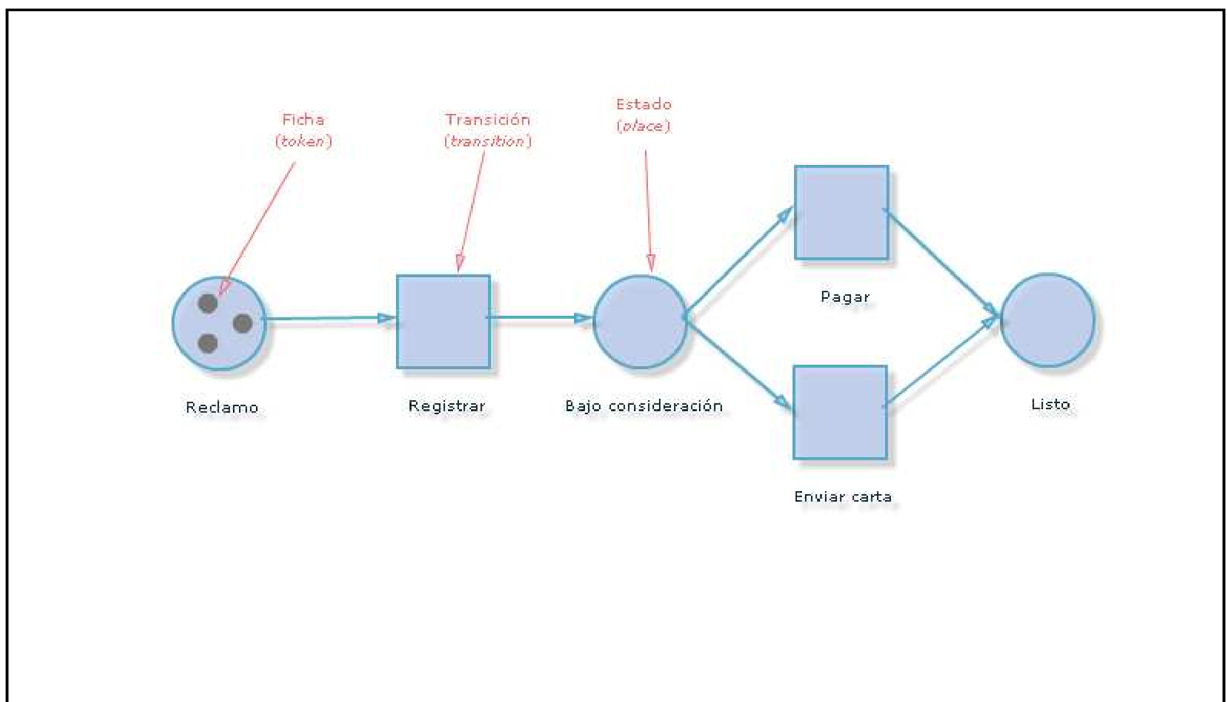


Ilustración 37. Ejemplo imagen PNG generada

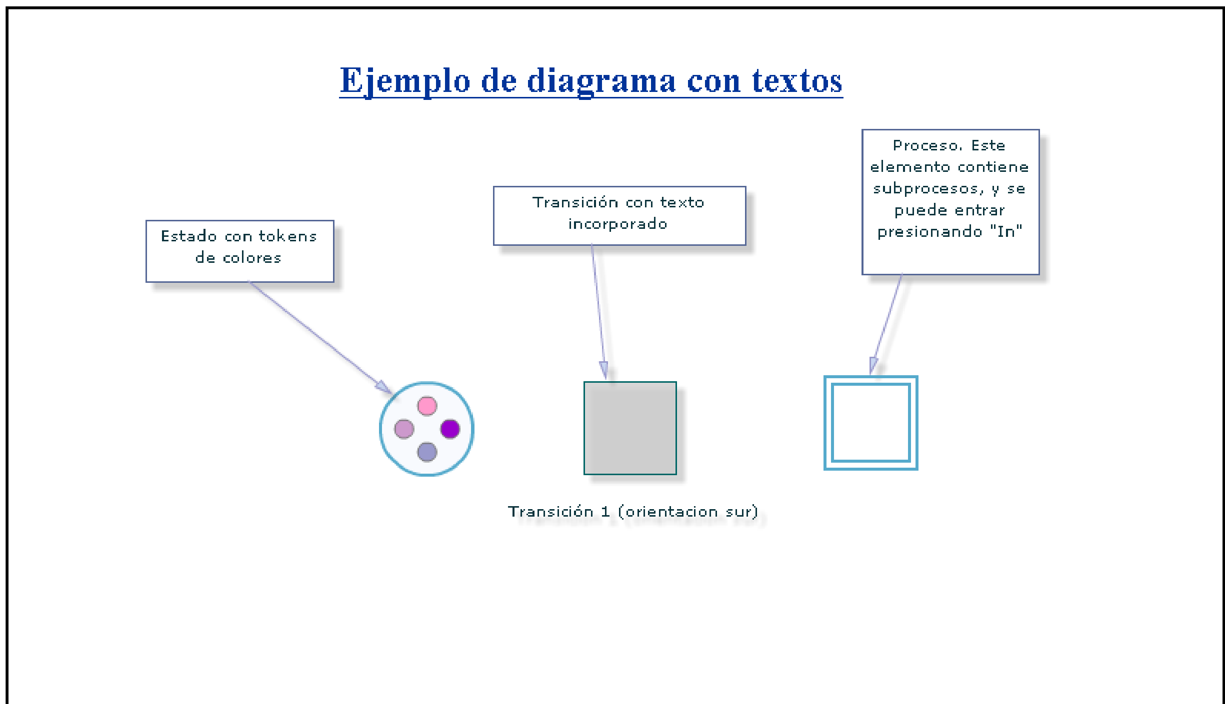


Ilustración 38. Ejemplo imagen PNG generada

Editar diagrama

Al presionar este botón pasamos al modo de edición de diagramas, en donde se construyen y modifican los diagramas. Más adelante se revisa esta funcionalidad en detalle.

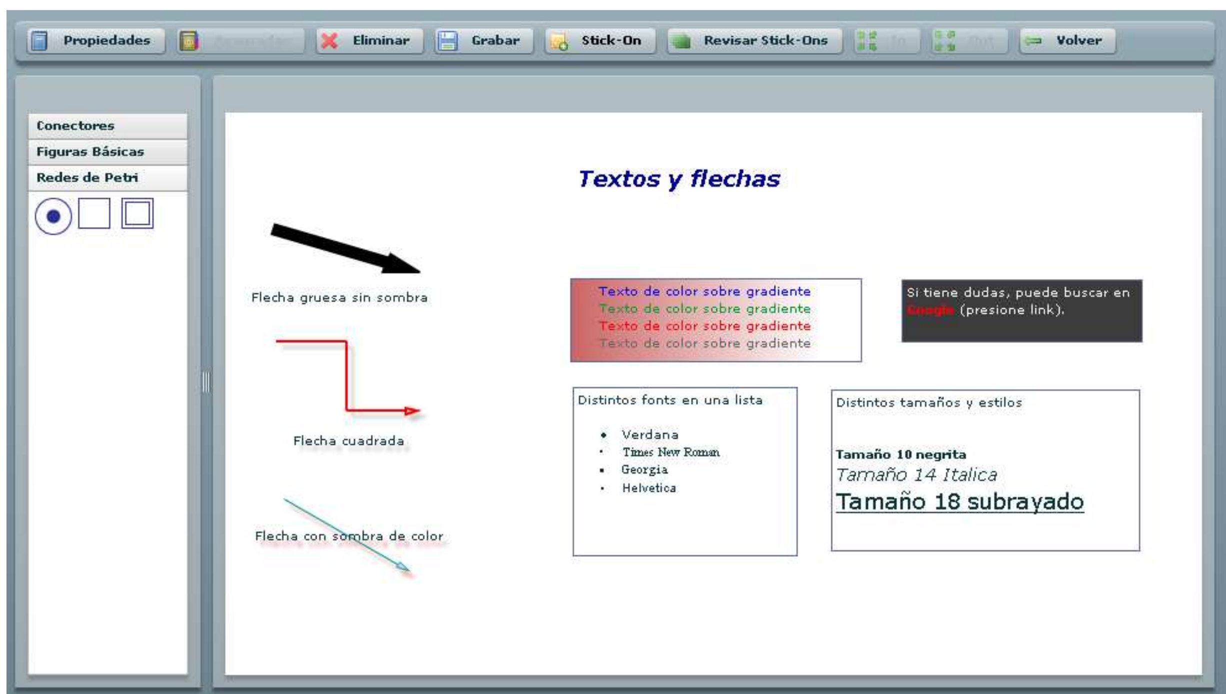


Ilustración 39. Editor de diagramas.

Compartir diagrama

La función de compartir diagrama nos permite dar o quitar permiso a otros usuarios para que modifiquen el diagrama actual, siempre y cuando seamos los creadores del documento. Tiene una interfaz parecida a lo que era "desbloquear diagramas":

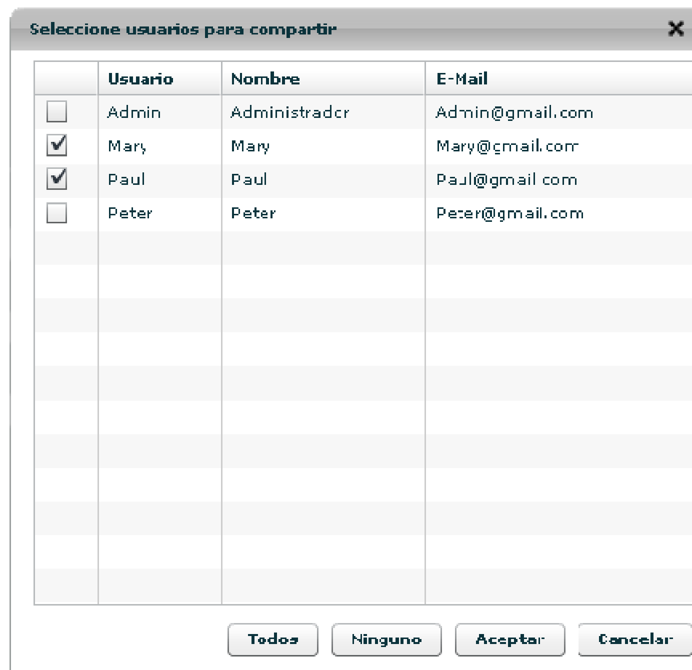


Ilustración 40. Función de compartir diagrama.

Al desplegarse esta ventana se consulta a la base de datos a que usuarios esta compartido el diagrama y se muestran chequeados. Un elemento chequeado quiere decir que este diagrama le aparecerá al usuario dentro de la lista de sus documentos y podrá modificarlo si es que nadie más lo está modificando. Puedo compartir o des-compartir diagramas en cualquier momento.

Desde el punto de vista de la interfaz este modelo puede ser criticable, ya que a diferencia de lo que ocurre en "desbloquear" en donde se muestran sólo los documentos que me pertenecen y están siendo modificados o que yo mismo estoy modificando, en esta pantalla se traen *todos* los usuarios del sitio y puedo marcarlos o desmarcarlos a voluntad. Esto podría ser problemático si comienzan a haber demasiados usuarios en el sitio por motivos obvios, ya que es difícil buscar dentro de una lista demasiado larga. Este problema se ve mitigado en parte gracias a la posibilidad de ordenar esta lista por cualquier columna, incluso por la de *checks* de compartido o no compartido, pero no pareciera ser la solución definitiva. Otro potencial efecto adverso de este enfoque es la cantidad de información que viaja y la capacidad del elemento gráfico para desplegar varios elementos si es que llegamos al orden de los miles de usuarios. Esta solución creo que se puede considerar aceptable para el orden de los cientos de usuarios.

Sección detalle

Comparte el espacio físico y algunas características con la sección de navegación que acabamos de revisar, sin embargo provee funcionalidades distintas. Se accede a esta vista pinchando en el botón "ver detalle" o simplemente haciendo *click* en cualquier lugar libre de las vistas en miniatura.

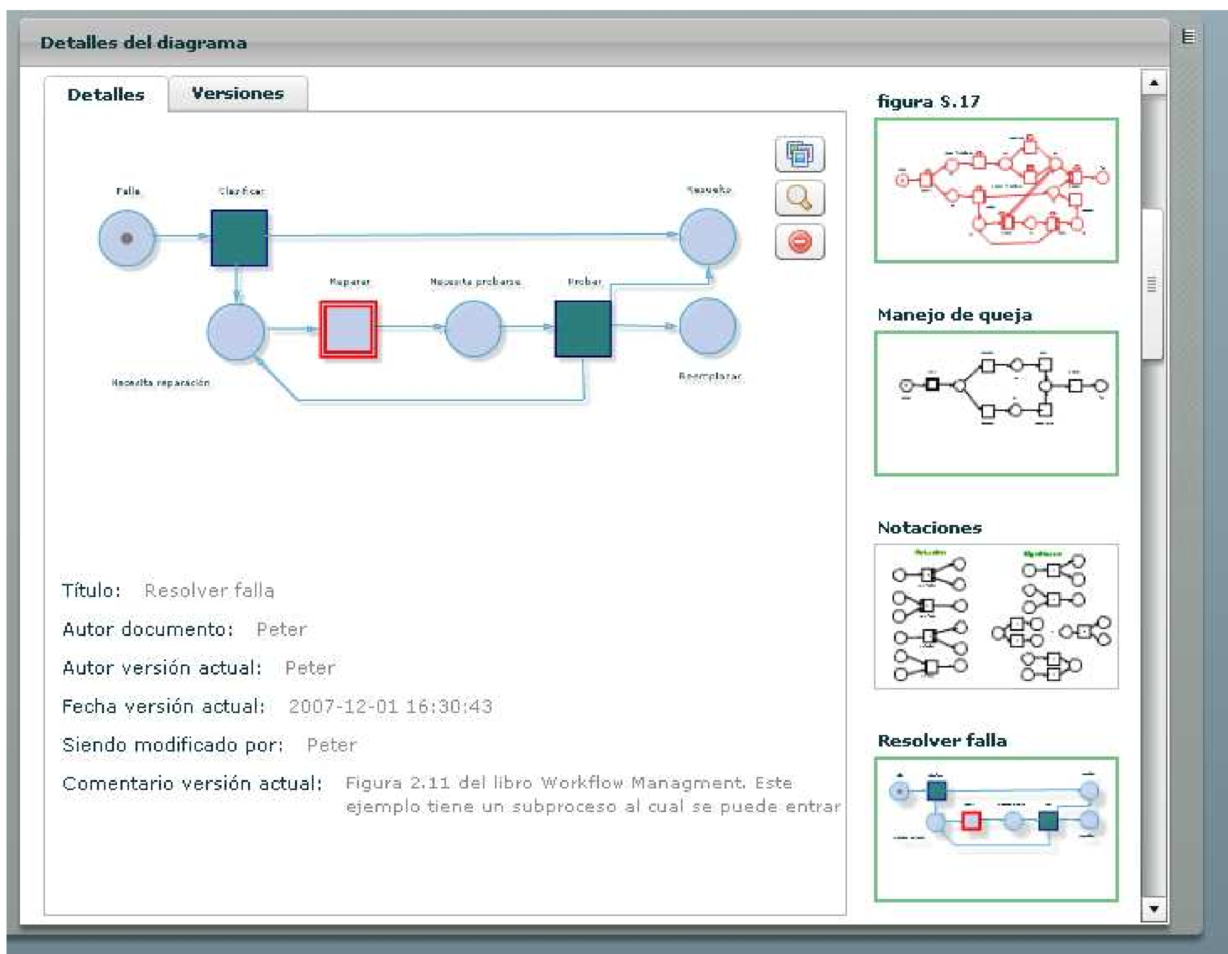


Ilustración 41. Vista detalle de diagrama.

La estructura de esta sección está dividida en dos partes. En el panel derecho tenemos una vista vertical de las miniaturas de los diagramas. Podemos usando la barra de *scroll* seleccionar cualquiera de ellos, y su detalle aparecerá al lado. En el panel de la izquierda se tiene el detalle del diagrama seleccionado, que incluye la vista más grande de la imagen que podemos obtener sin tener que ir a edición o ver la imagen almacenada. Además, se entrega todo el detalle de la versión actual del diagrama, por ejemplo el título, el autor de esta versión, el autor del documento, etc.

Una funcionalidad importante que se hace visible en esta vista es una nueva botonera que ofrece al menos por ahora dos nuevos elementos (por ahora).

Volver a vista de iconos

Permite salir de la vista de detalle y vuelve a la vista de iconos que es de donde previamente veníamos.

Ver imagen original

Misma función que cuando se está dentro del icono, abre en una ventana aparte imagen que se tiene almacenada en el servidor para que el usuario la pueda usar.

Eliminar diagrama

Elimina en forma permanente el diagrama seleccionado, así como todas sus versiones.

Cabe hacer notar que existen dos botoneras parecidas. La que aparece en el icono mismo cuando pasamos el mouse sobre él y que contiene funciones como editar, ver, compartir, etc., y la que aparece cuando entramos a ver el detalle del diagrama. Ambas botoneras actúan sobre el documento actual, por lo que el usuario se podría preguntar ¿por qué existen dos? La respuesta es que la ubicación más cómoda nos parece que es en el icono mismo. Sin embargo, el espacio físico para ubicar botones es limitado y no podemos tener más de cuatro. Sin embargo en el detalle pueden haber muchos más botones, por lo que el criterio es que los botones más usados deben ir en el icono y el resto en el detalle.

Versiones:

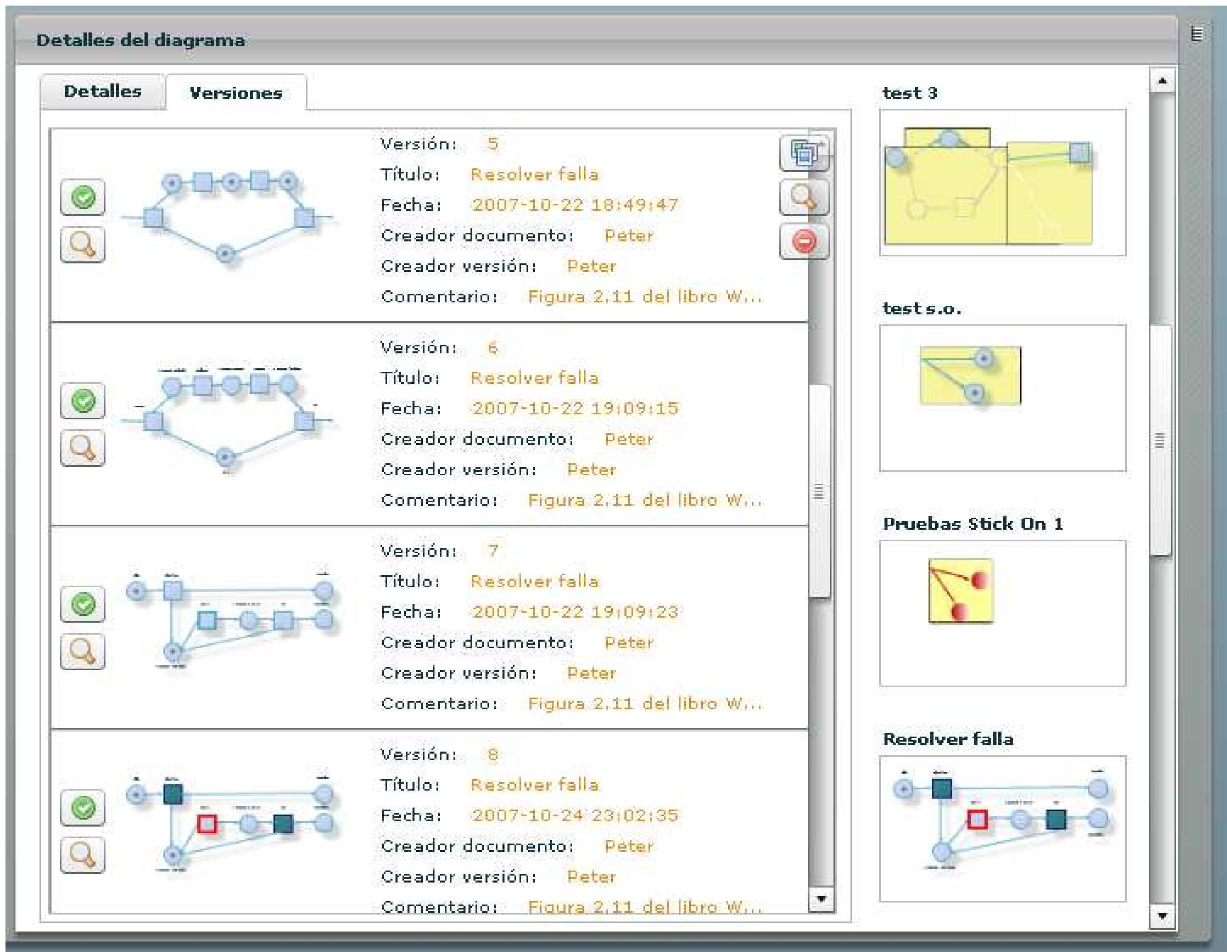


Ilustración 42. Detalle de diagramas, vista de versiones.

Cuando el usuario edita un diagrama en línea, cada vez que se graba se genera una nueva versión del documento y se almacena en el servidor. Esto tiene la ventaja de que el usuario nunca perderá información y siempre tendrá disponible en línea todo su trabajo y las colaboraciones de otros usuarios. El precio de esto es relativamente pequeño y se traduce en un poco más de uso en disco (por las imágenes) y en base de datos (por la información). La imagen almacenada en PNG de un diagrama típicamente pesa 10KB por lo que no creemos que vaya a ser problemático este enfoque.

Los botones a la izquierda de cada elemento de la lista de versiones operan sobre la versión del diagrama sobre la cual están posicionados, y permiten ver la imagen almacenada del diagrama o recuperar una versión determinada y hacerla la actual, con lo que se puede empezar a trabajar con dicha versión.

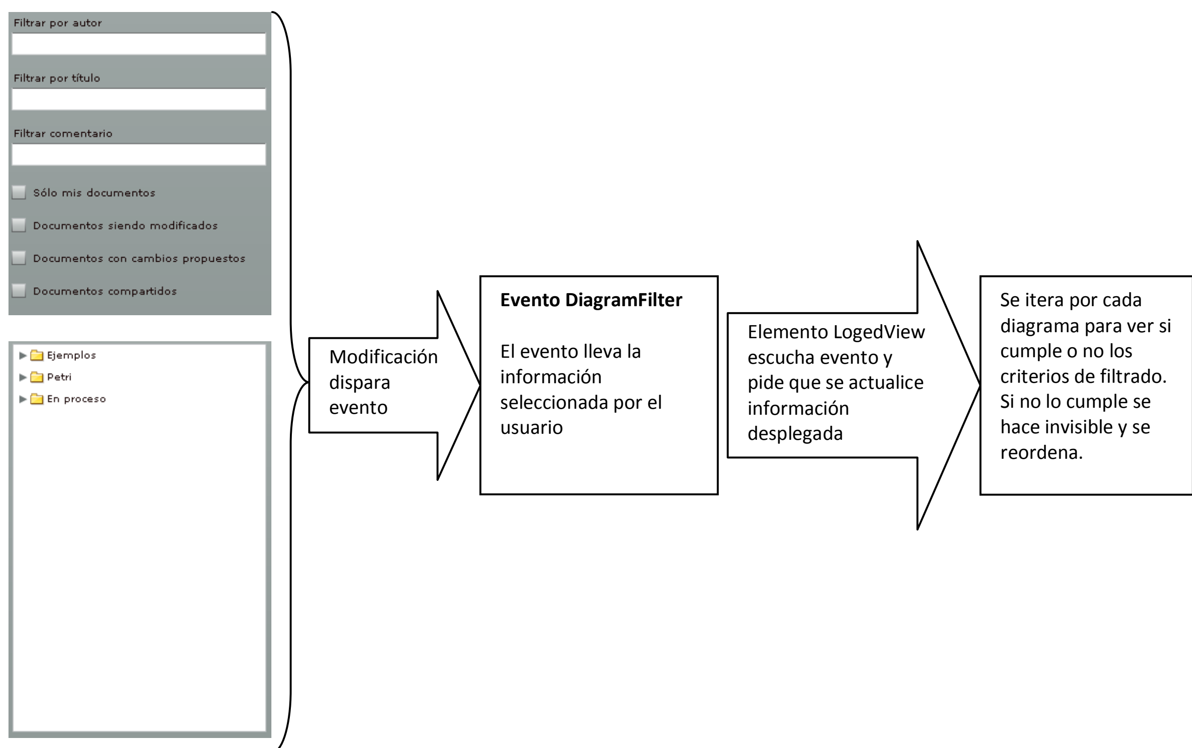
Filtros

Para poder tener cierto orden dentro de los diagramas, el sitio permite un esquema de filtrado más o menos sofisticado. Se puede especificar que diagramas quiero tener visibles en el panel de las vistas en miniaturas de una diversidad de formas, y aunque se puede ver un poco complicado, la implementación es en realidad más simple de lo que parecer. Por eso, antes de mostrar en detalle los posibles filtros analizaremos su implementación.

Cada uno de las vistas en miniatura de los diagramas tiene asociado un objeto que guarda la información que caracteriza al diagrama (clase Diagram) y que ha sido recibido desde un servicio web. Existe otro objeto (clase DiagramFilter) que puede almacenar la información que el usuario tiene seleccionada como filtro y que tiene un método que dado un Diagrama indica si cumple o no los criterios seleccionados en base a comparaciones muy sencillas. Cuando alguno de los criterios de filtro de la interfaz se modifica, se dispara un evento que lleva como información a la clase DiagramFilter actualizada y en el panel de vistas en miniaturas se hace un análisis de cuales elementos se deben ocultar.

```
public class Diagram
{
    public var owner:String;
    public var diagramId:int=0;
    public var diagramVersion:int;
    public var titulo:String="";
    public var description:String="";
    public var fecha:String;
    public var image:String;
    public var modifiedby:String;
    public var userediting:String;
    public var checkout:String;
    public var stickon:String;
    public var sharednumber:Number;

    public function Diagram(){}

    public function fill(obj:Object):void
    {
        for (var i:String in obj)
        {
            // algunas consideraciones especiales:
            if (i=='image')
               this[i] = Constantes.CONST_WEB_IMAGE_LOCATION + obj[i];
            else
               this[i] = obj[i];
        }
    }
}
```

```
public class DiagramFilter
{
    public var count:int;
    public var autorfilter:String='';
    public var titlefilter:String='';
    public var comentfilter:String='';
    public var useronly:Boolean=false;
    public var blocked:Boolean=false;
    public var stickedon:Boolean=false;
    public var shared:Boolean=false;
    public var categoriesActive : Boolean = false;
    public var categoriesUsers : Array = [];
    public var categoriesIds : Array = [];
    public function DiagramFilter()
    { super();}

    public function accept(diagram:Diagram):Boolean
    {
        if (useronly &&
            diagram.owner.toLowerCase()!=
            Application.application.username.toLowerCase())
            return false;

        if (autorfilter!='')
            if ((diagram.owner==null)||
                (autorfilter!='' &&
                diagram.owner.toLowerCase().indexOf(autorfilter.toLowerCase())!=0))
                return false;

        if (titlefilter!='')
            if ((diagram.titulo==null)||
                (titlefilter!='' &&
                diagram.titulo.toLowerCase().indexOf(titlefilter.toLowerCase())==-1))
                return false;

        if (comentfilter!='')
            if ((diagram.description==null)||
                (comentfilter!='' &&
                diagram.description.toLowerCase().indexOf(comentfilter.toLowerCase())==-1))
                return false;

        if (blocked && ((diagram.userediting=='')||(diagram.userediting==null)))
            return false;

        if (stickedon && (diagram.stickon!='S'))
            return false;

        if (shared && (diagram.sharednumber<=0))
            return false;

        var i:int = 0;
        var bExist:Boolean = false;
        if (categoriesActive) {
            for (i=0;i<categoriesUsers.length;i++) {
                if ((categoriesUsers[i]==diagram.owner)&&(categoriesIds[i]==diagram.diagramId))
                    bExist= true;
            }
            if (!bExist) return false;
        }
        return true;
    }
}
```



Funciones de filtrado

Las posibilidades que tiene el usuario para filtrar los documentos visibles se dividen en dos secciones. Primero están ciertas condiciones sobre los diagramas mismos, como que el nombre tenga ciertos caracteres, o que la descripción contenga determinado texto. Por otro lado, se permite al usuario crear categorías para poder tener agrupados los diagramas. Las distintas opciones de filtrado no son excluyentes, pudiendo combinarse para lograr encontrar él o los diagramas buscados con mayor facilidad. Veamos en detalle cada una de las opciones de filtrado.

Filtros por características

Esta clase de filtros comprende filtros relacionados con atributos directos del diagrama.

Ilustración 43. Opciones de filtro.

Filtrar por autor. A medida que se digita en el área de texto se va actualizando el listado de diagramas disponibles. En el caso de la búsqueda por autor, se entregan los diagramas cuyo nombre del creador comience con el texto escrito. No es sensible a mayúsculas.

Filtrar por Título. A medida que se digita en el área de texto se va actualizando el listado de diagramas disponibles. En el caso de la búsqueda por título, se entregan los diagramas cuyo título contenga el texto digitado. No es sensible a mayúsculas.

Filtrar comentario. A medida que se digita en el área de texto se va actualizando el listado de diagramas disponibles. Al igual que el filtrado por título, se entregan los diagramas que tengan un comentario que contenga el texto digitado. No es sensible a mayúsculas.

Filtrar por usuario modificando. Permite poder obtener los diagramas que están siendo modificados por un usuario dado. Si ingresamos nuestro usuario podremos tener una lista de los diagramas que estamos modificando y que tenemos bloqueados.

Sólo mis documentos. Muestra solamente los diagramas que ha creado el mismo usuario, no lo que le pudieron haber compartido otras personas.

Documentos siendo modificados. Muestra los documentos que están siendo modificados y están bloqueados, ya sea por mí o por otros usuarios.

Documentos con cambios propuestos. Muestra los documentos que contienen Stick-Ons.

Documentos compartidos. Muestra los documentos que pueden ser modificados por más de un usuario.

Por ejemplo, para poder encontrar todos mis diagramas que tengo compartidos, basta marcar las opciones de *sólo mis documentos* y *siendo compartidos*. Alternativamente, puedo marcar *documentos compartidos* y en *filtrar por autor* poner mí nombre. Este método me permitiría saber que diagramas me ha compartido un usuario dado con sólo escribir su nombre en el campo de filtrar por autor.

Filtros por categoría

Los filtros por categorías nos permiten catalogar los diagramas usando criterios definidos por los mismos usuarios. Para lograr esto, se permite a las personas crear su propia estructura de carpetas con los nombres que deseen, para posteriormente asociar los diagramas sobre los cuales tiene acceso a alguna de las categorías que ha creado.

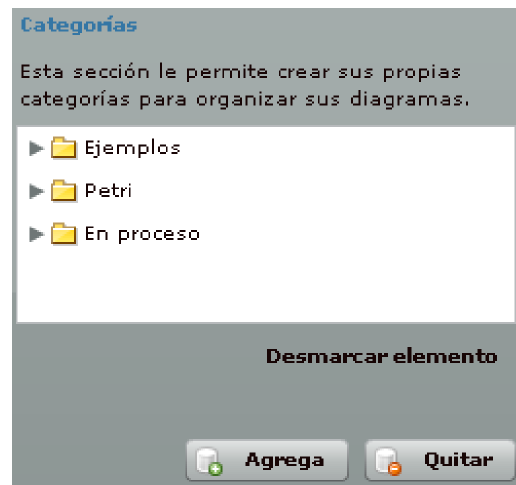


Ilustración 44. Categorías creadas por usuario.

Para agregar una categoría basta con presionar el botón agregar. En este punto debemos indicar si deseamos crear el nuevo elemento como una categoría raíz o como el hijo del elemento actualmente seleccionado, tal como se muestra en la siguiente pantalla.

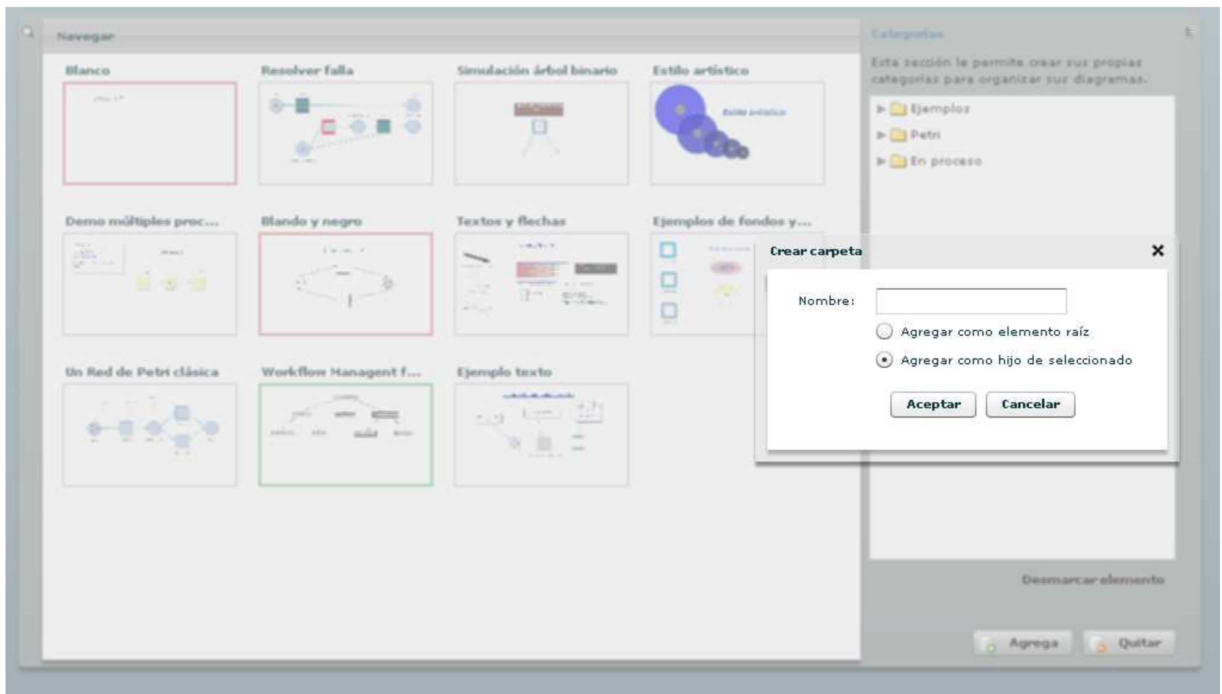


Ilustración 45. Ejemplo de creación de categoría.

Para agregar un diagrama al árbol de categorías se debe arrastrar la vista en miniatura hacia el directorio sobre el cual deseamos tenerlo

Ilustración 46. Agregar diagrama a categoría.

A continuación se muestra un árbol con categorías ya creados y con documentos asociados. Si se pincha en una categoría se muestran todos los hijos directos que tenga. Si pincho en uno de los hijos sólo se muestra ese elemento. En el ejemplo el usuario ha pinchado en “Gráficos”, categoría que consta de 4 elementos, y se han hecho visibles en el panel de vistas en miniaturas.

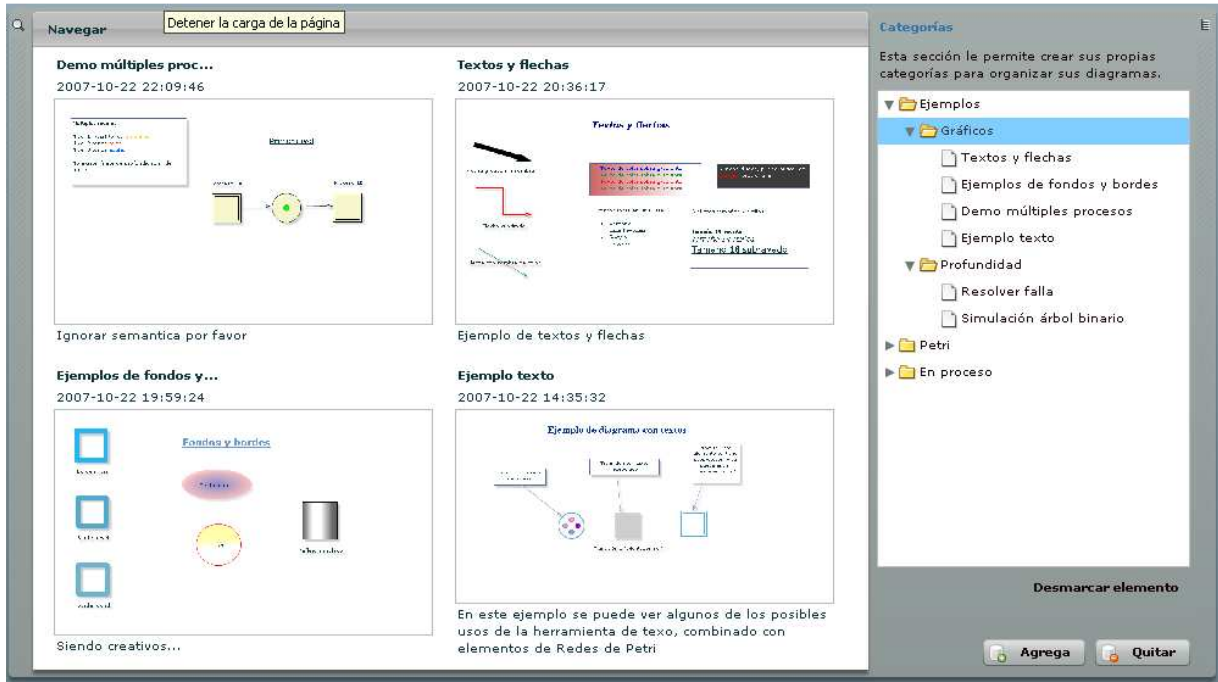


Ilustración 47. Usando árbol de categorías.

El filtro por categorías es compatible con los filtros por características en el sentido de que si tengo seleccionado un criterio de características, se respeta cuando selecciono una categoría, y se aplican ambas condiciones.

Justo bajo el árbol de categorías se puede ver un texto que dice “Desmarcar elemento”. Esto es en realidad un botón y nace de no tener una forma sencilla e intuitiva de desmarcar un elemento del árbol una vez seleccionado. Por eso cuando se desea no filtrar por categorías es con este botón que se desmarca cualquier elemento que haya estado marcado y se refresca la información desplegada.

5. Stick-Ons

Este capítulo “intermedio” entre la sección del sitio web y el editor de diagramas, presenta un diseño de los Stick-Ons que aplicaremos a los diagramas y pretende justificar la implementación creada en el editor de diagramas.

Justificación de los Stick-Ons

Usando el esquema de versiones implementado actualmente podemos manejar la situación en que Paul genera una nueva versión de un documento y lo graba. Si el cambio es del agrado de Peter el cambio permanece, y si no lo es, basta recuperar la versión previa del documento y aquí no ha pasado nada. Sin embargo, que ocurre si Mary hace una modificación sobre el documento que había modificado Paul? Recordemos que la modificación de Paul no era del agrado de Peter, sin embargo Peter no tuvo la oportunidad de recuperar la versión previa antes de que Mary realizara sus cambios. Llegamos a una situación en que Peter desea mantener el cambio de Mary, pero no el de Paul. Desafortunadamente no hay nada que Peter pueda hacer para tener sólo el cambio de Mary, ya que este cambio fue hecho sobre el documento que había modificado Paul. El pobre Paul tendrá que recuperar la versión original y aplicar sobre ella los cambios de Mary, o simplemente eliminar manualmente los cambios de Paul. Está de más decir que esta es una operación tediosa y peor aún, se puede introducir errores al realizarla. Tal vez a Peter le agradaría que Paul y Mary hubieran usado Stick-Ons para hacer sus cambios.

Diseño de Stick-Ons para diagramas

Originalmente los Stick-Ons fueron introducidos en el contexto de edición de texto [1]. Nosotros deseamos aplicarlos a la edición de diagramas, por lo que un análisis de las posibles diferencias es recomendable. Existen al menos dos puntos que no son completamente equivalentes y que pueden tener un impacto directo en el diseño que se hará de los Stick-Ons. Estos puntos son que la modificación de una parte del diagrama tiene un impacto muy limitado sobre el resto del diagrama y que en general pueden convivir en el mismo espacio físico distintos elementos.

Diferencias entre texto y diagramas

La primera diferencia que consideraremos consiste en el impacto que tiene la modificación de una parte del diagrama comparado con la misma acción cuando se realiza sobre texto. Las operaciones que se realizan en un principio son las mismas: agregar, cambiar, eliminar. Sin embargo, hay también un tipo de operación común en los diagramas y que no tiene un pariente directo en el mundo del texto, y es el mover un elemento.

Veamos que ocurre al eliminar un elemento de un texto. Dentro de un párrafo la acción que se debe realizar es clara, hay que reordenar el párrafo para llenar el espacio que ha desaparecido.

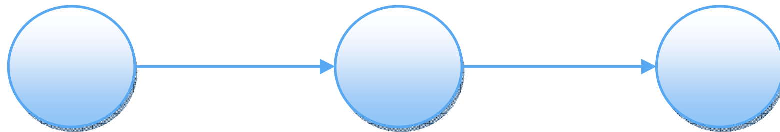
Bien está todo eso -replicó don Quijote-, pero quédense los zapatos y las sangrías por los azotes que sin culpa le habéis dado; que si él rompió el cuero de los zapatos que vos pagastes, vos le habéis rompido el de su cuerpo; y si le sacó el barbero sangre estando enfermo, vos en sanidad se la habéis sacado; así que, por esta parte, no os debe nada.

2. El espacio dejado por la eliminación desaparece y el texto se reajusta para llenar el vacío.

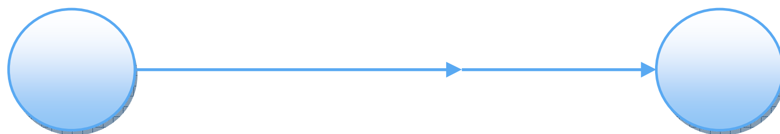
1. El texto marcado va a ser borrado

Bien está todo eso, pero quédense los zapatos y las sangrías por los azotes que sin culpa le habéis dado; que si él rompió el cuero de los zapatos que vos pagastes, vos le habéis rompido el de su cuerpo; y si le sacó el barbero sangre estando enfermo, vos en sanidad se la habéis sacado; así que, por esta parte, no os debe nada.

La situación es muy distinta cuando eliminamos un elemento de un diagrama, por ejemplo al eliminar la parte central en la siguiente figura:



No hay ningún motivo para suponer que debe quedar así:



O así:

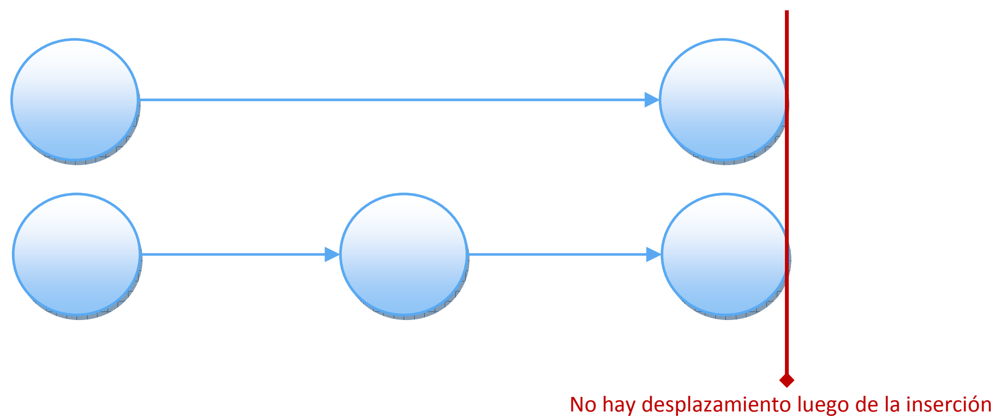
Que sería el caso análogo al de texto. En esta situación estaríamos tomando decisiones por el usuario que no necesariamente son correctas, lo cual se transformaría rápidamente en una actitud molesta. Por ejemplo en este caso el usuario pudo eliminar el círculo central para poner un cuadrado, por lo cual no sería correcto estirar y unir la flecha como en la figura. Por lo tanto no intentaremos asumir intenciones, así que la eliminación en el caso del diagrama para nosotros no tendrá un efecto en el contexto exterior del diagrama –como si ocurre con la eliminación de texto en un párrafo–, por lo que quedará simplemente así:

De la misma manera, modificar o agregar tiene un impacto muy limitado en el contexto del diagrama, limitándose tal vez a los elementos que están anclados al elemento modificado. Esto no es así en el caso del texto, ya que si agrego una sentencia todo lo que está a la derecha debe ajustarse:

Agregar este texto tiene el efecto de desplazar a la derecha lo que antes estaba aquí.

> Bien está todo eso, pero quédense los zapatos y las sangrías por los azotes que sin culpa le habéis dado; que si él rompió el cuero de los zapatos que vos pagastes, vos le habéis rompido [Nota, esta conjugación ya no se usa] el de su cuerpo; y si le sacó el barbero sangre estando enfermo, vos en sanidad se la habéis sacado; así que, por esta parte, no os debe nada.

En los diagramas, agregar un elemento no tiene efecto en sus vecinos (salvo en los conectores pero esto puede verse como una segunda operación manual), por ejemplo en la situación descrita en la figura al agregar un elemento no existe un desplazamiento hacia la derecha del elemento más extremo.

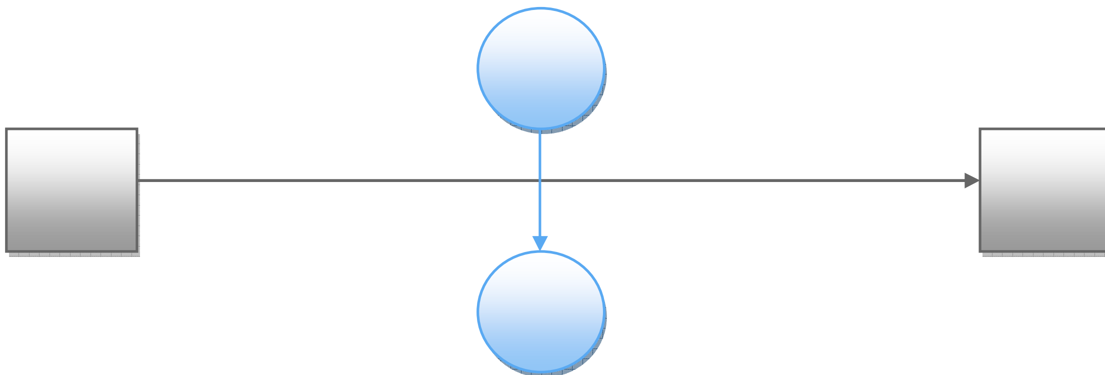


En definitiva, no nos interesa demasiado el contexto de las modificaciones en el caso de los diagramas, sólo los elementos que explícitamente han sido cambiados, modificados o eliminados.

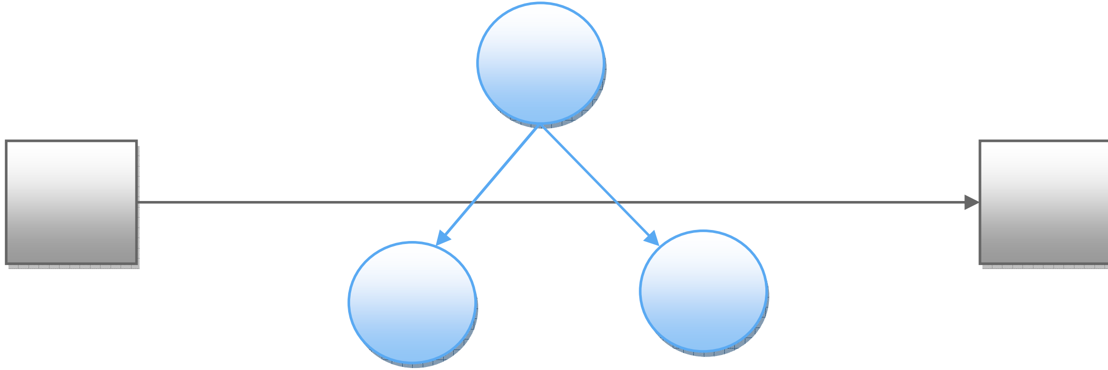
Otra diferencia que se puede observar entre diagramas y textos es que en el diagrama los objetos pueden compartir el espacio, no así en el caso del texto a menos que se trate de párrafos especiales que seguramente necesitarán un tratamiento diferente, pero normalmente no se tiene esta situación, por ejemplo:

> Bien está todo eso, pero quédense los zapatos y las sangrías por los azotes que sin culpa le habéis dado; que si él rompió el cuero de los zapatos que vos pagastes, vos le habéis rompido el de su cuerpo; y si le sacó el barbero sangre estando enfermo, vos en sanidad se la habéis sacado; así que, por esta parte no os debe nada.
>
> OTRO TEXTO NO RELACIONAD

El caso recién mostrado es patológico y no se encontrará salvo en ejemplos especialmente fabricados o casos muy específicos, como marcas de agua en el papel o cosas así. Sin embargo, el caso normal es tan predominante que podemos decir como regla general que el espacio usado por una letra o una palabra le pertenece a ella y en ese espacio no puede existir otra letra o palabra. Eso no es cierto cuando pensamos en diagramas. Por ejemplo:



En este caso, a menos que se impongan condiciones respecto al corte de conectores, tenemos que ambas flechas comparten parte del mismo espacio, al menos en el punto donde se interceptan. Esto no es el caso solo de los conectores, sino que también de las figuras. Eventualmente, podríamos poner un círculo más pequeño dentro de otro círculo, para simular fichas en una Red de Petri por ejemplo, pero no podemos poner Letras dentro de otras letras. Incluso podemos modificar la parte central sin que la parte gris se vea afectada para nada.



Para el cambio ejemplificado, los elementos cuadrados unidos por una flecha no tienen ninguna relevancia.

Finalmente, el arrastrar un elemento del diagrama para moverlo de lugar es una operación que no tiene equivalente directo al momento de editar texto, pero es una operación muy común al trabajar con diagramas. Esta diferencia y las anteriores van a ser importantes en la siguiente sección, donde se propone una pequeña modificación a la forma de poner un Stick-On.

¿Qué se selecciona cuando se selecciona?

Al igual que Gonzalo Rojas en su poema *¿Qué se ama cuando se ama?,* nos hacemos una pregunta del mismo estilo. La forma de poner un Stick-On en el caso de texto es seleccionar el texto a reemplazar y presionar un botón “nuevo”, lo que hace que aparezca automáticamente un pequeño cuadrado de color sobre el cual escribir un nuevo texto, y si es necesario el Stick-On puede crecer para ajustarse al texto ingresado. Sin embargo, este esquema que funciona perfectamente para el caso de texto, presenta algunos inconvenientes al tratar de llevarlo a los diagramas.

Una cosa fundamental al seleccionar texto con el mouse es que la selección se extiende de forma de cubrir automáticamente las letras. Impensable es que la selección tome la mitad de una letra por ejemplo, ya que no sabríamos que ocurre si borro lo marcado por esa selección. Al mismo tipo de inconsistencias podemos llegar si permitimos que al momento de seleccionar en un diagrama para especificar un Stick-On se seleccionen áreas que contengan trozos de elementos, por lo que el Stick-On debería ampliarse para contener en su totalidad los elementos que tenían alguna parte en el área seleccionada. El problema es que esta extensión podría involucrar a elementos que en realidad no me interesan (ya que los elementos pueden

compartir el espacio) agrandando mucho el Stick-On y haciendo que pierda su función de dar una indicación gráfica de *donde* está el cambio, o bien puede ser imposible extender la selección para tener elementos enteros sin tener que llegar a cubrir el diagrama completo.

Otra ventaja que ofrece el seleccionar el lugar donde va a ir el Stick-On en el caso del texto es que si la superficie crece, como tenemos bien definidos los bordes del Stick-On, el texto que sigue dentro del párrafo se podrá reajustar de acuerdo a los cambios realizados. Esta ventaja, según lo discutido en el sentido de que los cambios en el diagrama no afectan en general a sus vecinos, no aplica cuando pasamos al mundo de los diagramas.

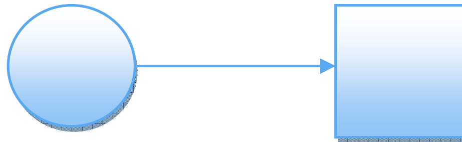
Como mencionábamos en la sección anterior, el mover un elemento no tiene equivalente directo al momento de editar diagrama (cortar y pegar podría ser lo más parecido). Para considerar esta operación en un Stick-On uno se imagina que el usuario marcar el espacio alrededor del elemento a mover para poner el Stick-On y al igual que cuando el Stick-On se ajusta a un texto más grande, en este caso se agranda si se mueve al elemento fuera de los limites. Sin embargo, si el Stick-On se va a ajustar automáticamente, cabe preguntarse si era realmente necesario marcar la posición del Stick-On original.

Por los motivos que acabamos de comentar hemos decidido cambiar un poco la forma de "pegar" un Stick-On. Aunque las razones expuestas no son lapidarias ni mucho menos, si hacen que podamos pensar en una alternativa a seleccionar la ubicación física del Stick-On con el mouse en el caso de los diagramas. La alternativa que proponemos es que el sistema pegue automáticamente los Stick-Ons cuando el usuario ha realizado un cambio, siempre y cuando haya indicado que se trata de una modificación opcional mediante un botón en la interfaz. Por ejemplo, en el caso recién mencionado de mover un elemento, la forma de uso sería la siguiente. Tengo un diagrama y deseo mover un elemento algunos pixeles a la derecha, pero sólo como una propuesta a evaluar. Para ello presiono el botón "nuevo Stick-On" y el sistema entra en un estado especial de grabar los cambios. Voy y arrastro el elemento a la derecha. Esta operación genera un Stick-On que cubre la posición del elemento original y se extiende hasta cubrir la posición final del elemento. Le indicamos al sistema que hemos terminado la modificación y el resultado final es un Stick-On que tiene la figura en su posición final, y al levantar el Stick-On se muestra la figura en su posición original y la sombra del Stick-On que acabo de levantar. Revisaremos los casos en más detalle en la siguiente sección.

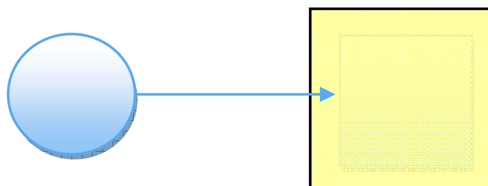
Casos de uso básicos

Eliminar

Partamos por el más sencillo a mi gusto, que es eliminar un elemento. El usuario presiona botón Stick-On y procede a eliminar un elemento. Partiendo de la siguiente situación:

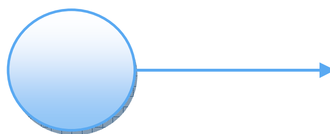


El usuario ha indicado que va a hacer un cambio tentativo ya que presionó el botón Stick-On, marca el cuadrado y lo elimina. Si es todo lo que deseaba modificar, hay que avisarle al sistema que ha finalizado con sus cambios volviendo a presionar el botón Stick-On. Con esta acción debe aparecer un Stick-On "en blanco" cubriendo el elemento eliminado. Al removerlo se podrá ver el cuadrado que quedo tapado por el Stick-On. La situación final se esquematiza en la siguiente figura:

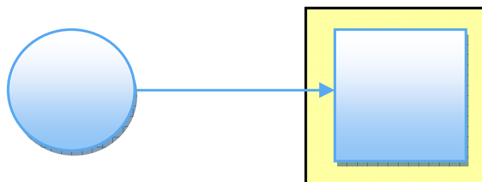


Agregar

Agregar elementos, situación original:



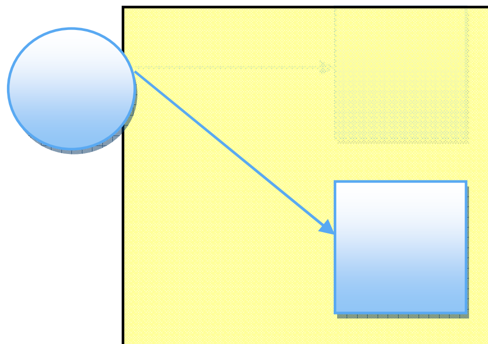
Agregamos un cuadrado y finalizamos los cambios. Debe aparecer el cuadrado sobre un Stick-On que al ser removido hace que desaparezca el cuadrado y muestra un espacio en blanco.



Mover

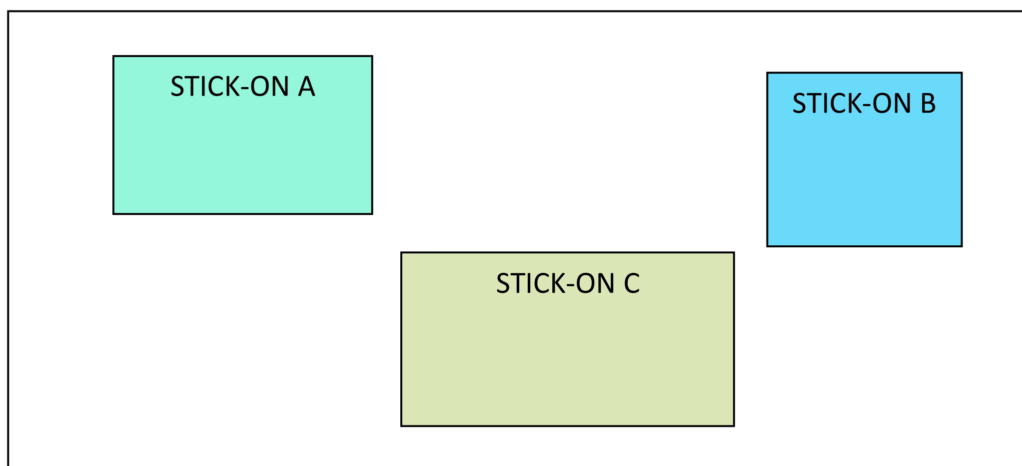
El siguiente caso refleja la situación en que el usuario mueve hacia abajo el cuadrado de la derecha. El conector está anclado en el cuadrado por lo que también se moverá.

El Stick-On que aparece cuando el usuario termina sus modificaciones debe cubrir las posiciones originales de la flecha (que ha movido producto del desplazamiento del cuadrado) y el cuadrado, así como las posiciones finales de ambos, como se muestra en el esquema.



Múltiples Stick-Ons

Los casos básicos con un solo Stick-On se manejan sin dificultad y dan una buena indicación de que algo se ha cambiado, permitiendo recuperar la versión alternativa con facilidad. Cuando se usan más Stick-Ons algunas situaciones más complejas pueden ocurrir y se deben analizar y definir el comportamiento que tendrán. Veamos primero el caso más simple, en que se tienen varios Stick-Ons pero no existe solapamiento ni adyacencia entre ellos.



El usuario puede sacar cualquiera de los Stick-Ons A, B o C, reduciéndose el problema en cada oportunidad al manejo de un solo Stick-On ya revisado y ver el diagrama resultante en cada situación. Puede que se acepten los cambios presentados por los Stick-Ons A y B, pero no los propuestos por el Stick-On C. Para ello tendría que sacar el Stick-On C de tal modo de tener en pantalla la versión que le interesa y luego indicarle al sistema que tome una "fotocopia" de este estado, generando una nueva versión en limpio sobre la cual eventualmente se podrá seguir trabajando. Un sencillo cálculo usando el principio multiplicativo nos dice que en este caso se pueden generar 2 x 2 x 2 = 8 versiones distintas de este documento, dependiendo de si se acepta o no cada uno de los Stick-Ons.

Solapamiento

Cuando se pegan Stick-Ons al diagrama se pueden producir algunos solapamientos entre Stick-Ons o entre Stick-Ons y elementos que no pertenecen directamente a los Stick-Ons. En esta sección analizaremos estos casos y definiremos las acciones a tomar. Partamos diagramando la situación con la siguiente figura.

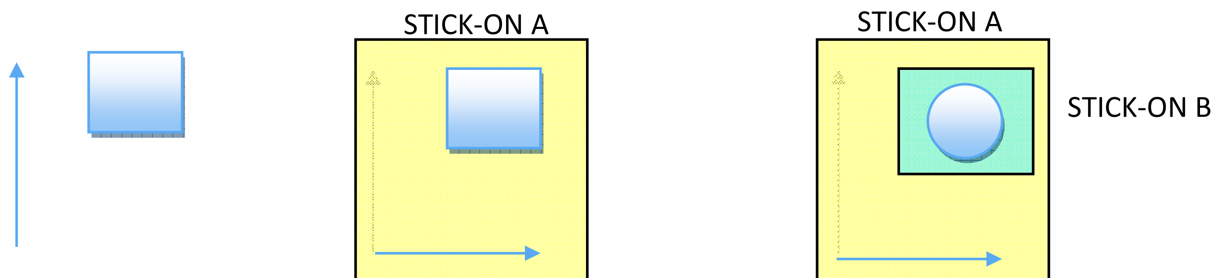


Ilustración 48. Ejemplo solapamiento

La situación representada en la figura es la siguiente. Inicialmente tenemos una flecha y un cuadrado lado a lado. Alguien desea modificar la orientación de la flecha pero lo quiere hacer como una propuesta de cambio. Para ello entonces pone un Stick-On A en donde ha movido la flecha vertical a una posición horizontal. Llegamos a la situación central de la figura. El cuadrado en realidad no tiene que ver con el Stick-On A, ni depende de si se acepta o no el Stick-On A, pero a quedado dentro de la superficie del Stick-On A, con lo que no queda claro si esta en el Stick-On o no. Adicionalmente, -parte derecha de la figura- supongamos que se pone otro Stick-On, B, el cual representa un cambio del cuadrado por un círculo. Este nuevo Stick-On B también coincide con el espacio usado por el Stick-On A, pero al igual que el cuadrado que le dio vida, no tiene relación directa con él. En este caso, no queda claro que ocurrirá al quitar el Stick-On A. ¿Sale el Stick-On B también? ¿Permanece?

Parte de los problemas aquí comentados y algunos que se describen un poco más adelante en la sección de “Sombras” surgen de tener Stick-On rectangulares. Un camino a explorar puede ser usar Stick-Ons con formas irregulares. Sin embargo, no deseamos apartarnos de las formas rectangulares. Después de todo, un rectángulo es un rectángulo, y tiene muchas ventajas también, por ejemplo desde el punto de vista de implementación y visualización sobre figuras más complejas. Por lo tanto definiremos las acciones que tendrá nuestro sistema para los casos en que solapan dos Stick-Ons o cuando un Stick-On queda sobre un elemento que no está dentro de los cambios propuestos.

Inicialmente para el caso en que un Stick-On quedaba sobre terceros elementos por el sólo hecho de estar cerca al Stick-On pensábamos tener Stick-Ons semitransparentes, que permitieran ver lo que quedaba abajo y sobre él mostrar los elementos que eran propuestos por el Stick-On. Sin embargo, al tener varios Stick-Ons que solapaban la situación se complicaba y no se lograba decidir qué elementos pertenecían a cual Stick-On. Al desechar la idea de Stick-Ons con transparencia, llegamos a otro tipo de solución para este caso y que es el implementado en esta memoria. Cuando un Stick-On tapa un elemento que no tiene que ver con los cambios, este proyecta una “Silueta” sobre el Stick-On. Esto para cualquier elemento que este siendo tapado, ya sea que pertenezca a otro Stick-On o no. Esta silueta nos permite tener una visión del gráfico que está quedando aún cuando estén pegados los Stick-Ons opacos, y por otro lado podemos tener una visión clara del cambio que propone el Stick-On. Por ejemplo el caso medio de la figura anterior quedaría de la siguiente manera:

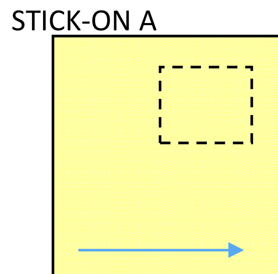


Ilustración 49. Ejemplo de silueta

El cuadrado que ha sido tapado proyecta una silueta y el Stick-On muestra sus cambios en forma clara. Además, las líneas de trocitos son exclusivas para las siluetas, el usuario no puede poner elementos de este estilo en los diagramas, ya que no son parte de las propiedades estándares de estilo.

El caso en que quedan sobrepuestos dos Stick-Ons lo trataremos de la siguiente manera: Los Stick-Ons van quedando uno sobre otro en función del momento en que se pudieron. Los más nuevos quedan arriba y se pueden aceptar o rechazar independientemente siempre. Si uno cubre totalmente a otro, basta con quitar el más grande y tomar una decisión sobre el que estaba cubierto, o ir al listado general de Stick-Ons que tendremos y desde ahí aceptar o rechazar. Esto aunque un segundo Stick-On esté utilizando elementos pertenecientes a un primer Stick-On, como mostramos en la siguiente figura.

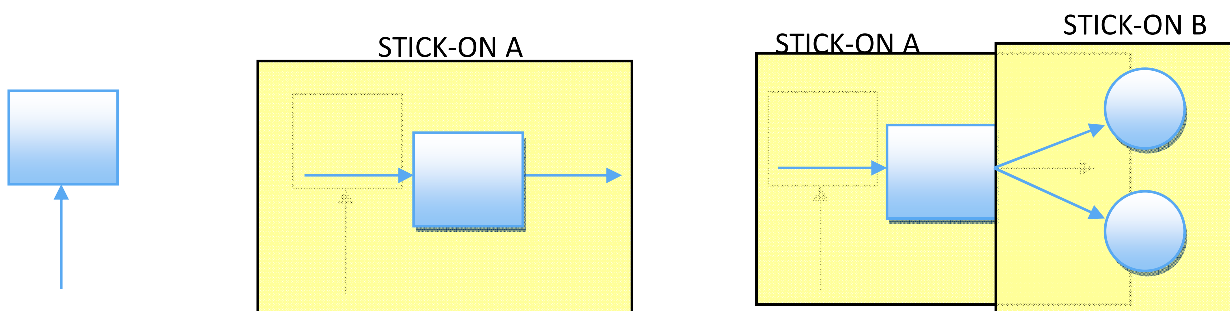


Ilustración 50. Ejemplo de Stick-On sobre elementos de otro Stick-On

En este ejemplo mostramos un caso en que el solapamiento ya no es tan “circunstancial” como lo que habíamos analizado antes. En este caso el Stick-On B usa elementos del Stick-On A (la flecha hacia la derecha). El procedimiento que se usó para poner el Stick-On B fue mover la punta de la flecha un poco hacia arriba, y se agregaron los otros elementos. Si quito el Stick-On A, supuestamente la flecha que dio en parte origen al Stick-On B ya no existirá. Tal vez se podría pensar en sacarla del Stick-On B, sin embargo no es lo que haremos ya que no nos parece que solamente limitaría las posibilidades de lo que puede hacer el usuario sin ser algo

necesariamente correcto, ya que por otro lado se podría argumentar que el Stick-On B hizo una copia de la flecha y que se quite el original no implica que deba perder la copia.

Aprovechando el diagrama anterior, podemos usar la parte final del ejemplo para mostrar un posible Stick-On que cubra los elementos desplegados:

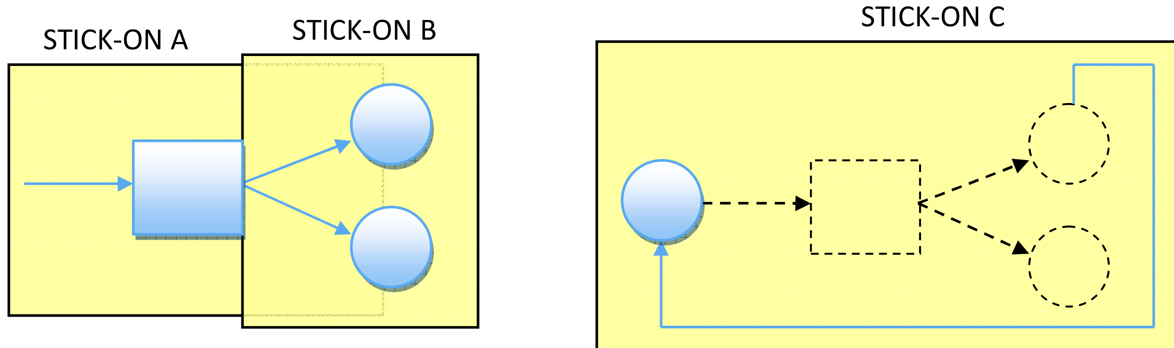


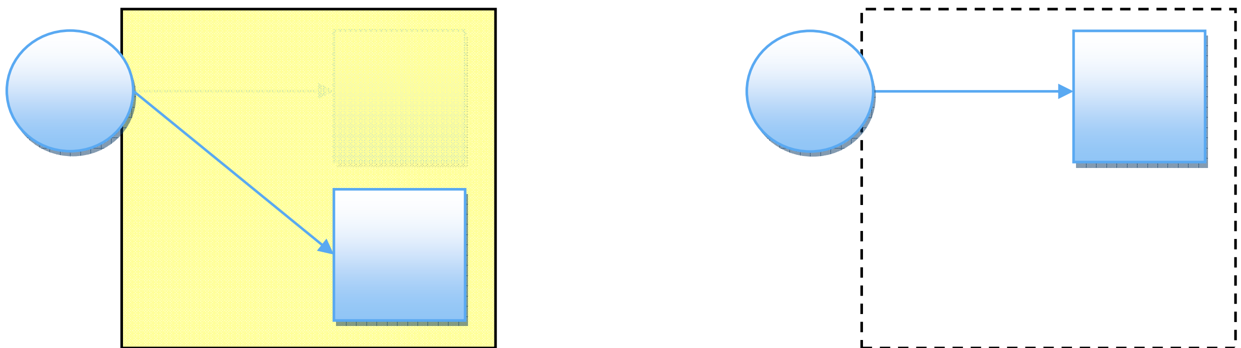
Ilustración 51. Ejemplo silueta de varios Stick-Ons

En el ejemplo se han agregado usando un Stick-On una flecha que la vuelta por el perímetro del diagrama previo, y llega a un nuevo elemento circular. Eso tiene el efecto de tapar los elementos antiguos ya que el nuevo Stick-On rectangular tiene que cubrir a la nueva flecha y el nuevo elemento. Sin embargo, los elementos externos que son cubiertos quedan claramente descritos por su silueta y lo que hay debajo de este Stick-On puede ser una serie de Stick-Ons y elementos, pero la silueta es una sola.

Finalmente, una palabra sobre el color de los Stick-Ons. En [1] se discuten algunas alternativas respecto a posibles esquemas de color, pudiendo representar el orden en que han sido puestos o que represente al creador del Stick-On. Debido a que nuestro sistema es precisamente para diagramar y en general podemos tener cuadrados de colores muy parecidos a los Stick-Ons, para intentar disminuir la confusión hemos decidido que por defecto todos los Stick-Ons tengan el mismo color. El usuario puede a su gusto cambiar posteriormente el color de los Stick-Ons, pero por defecto los Stick-Ons son creados en un tono amarillo crema.

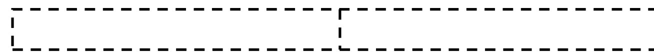
Sombras

Dentro del proceso en que el usuario revisa las distintas versiones a que dan lugar los Stick-Ons mediante acciones de "pegar y despegar" surge un concepto que no hemos profundizado hasta ahora, y es la *sombra* que deja el Stick-On al ser removido. Esta sombra existe para darle al usuario una indicación de que aquí había un Stick-On y que sólo que fue removido para inspeccionar lo que había bajo él. Puede ser visto como restos de pegamento que dejó el Stick-On al ser quitado. Dentro de esta sección de diseño denotaremos a las sombras como un rectángulo incoloro cuyos bordes son líneas cortadas. Por ejemplo en la siguiente figura se muestra un sector de un diagrama con un Stick-On y a la derecha vemos la sombra que deja al momento de removerlo para ver los elementos anteriores:



La implementación de los Stick-Ons debe tener cuidado de que cuando el usuario este pegando y despegando Stick-Ons para revisar las distintas versiones generadas no se pierdan datos ni de que se le presente información que pueda confundirlo. En [1] se revisan algunos casos que ya hemos mencionado en la introducción de este documento. Veremos cómo se aplican cuando tratamos con Diagramas. Esta parte es independiente de *cómo* se ha puesto el Stick-On, ya sea manualmente o en forma automática como proponemos. Conceptualmente además, es independiente de la información que muestra o no muestra el Stick-On por lo que las figuras serán solo esquemáticas sin considerar los elementos de los diagramas.

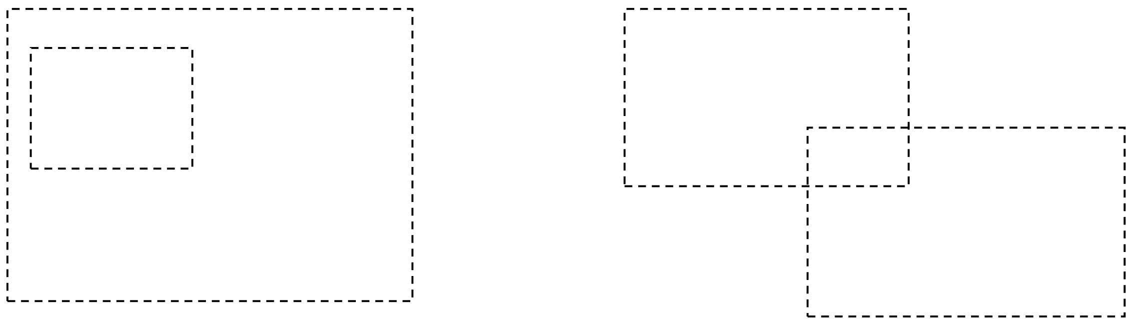
La principal fuente de problemas al momento de la revisión de los Stick-Ons en el caso de texto es que si existe un solapamiento entre dos o más Stick-Ons, si se despegan y las sombras de estos Stick-Ons son representadas por líneas punteadas, llegamos a un caso ambiguo como el que despliega en la figura:



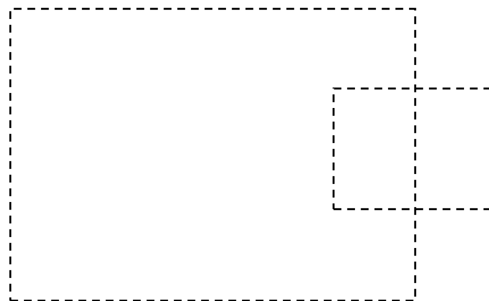
Lo más intuitivo según [1] es que esto son dos sombras de Stick-Ons contiguos que han sido despegados, con lo cual estamos de acuerdo. La otra interpretación que se podría tomar para este caso es que se trata de un Stick-On más largo que ha sido despegado y otro más corto que cubre la mitad y que también se ha despegado. Para evitar la ambigüedad la implementación de los Stick-Ons en texto tiene cuidado de que siempre que se presente el caso recién descrito se

trata de dos Stick-Ons contiguos, y en un caso más general, la implementación debe tener cuidado de que esta figura no lleve a ambigüedades.

Veamos las posibilidades de las sombras que dejan los Stick-Ons en diagramas, o para el caso, en dos dimensiones. En una primera instancia, podríamos pensar que la situación ha mejorado, ya que no cualquier caso de solapamiento de Stick-Ons que han sido removidos lleva a ambigüedad en dos dimensiones, como si ocurre en una dimensión (caso de texto). Por ejemplo si un Stick-On está totalmente contenido en otro no hay ambigüedad, así como tampoco necesariamente la hay cuando solapan parcialmente:

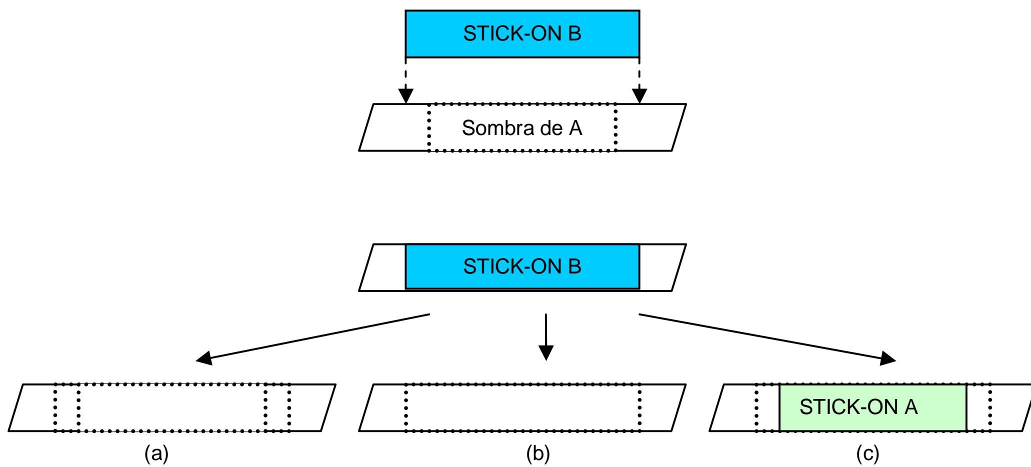


En estos casos no hay problemas en interpretar las sombras. En la figura de la derecha si hay una posibilidad extra a considerar y es que en el pequeño cuadrado que se forma en la intercepción haya una tercera sombra, pero no es necesario ser tan rebuscado para que aparezcan los problemas. Consideremos la siguiente figura que puede ser muy común:



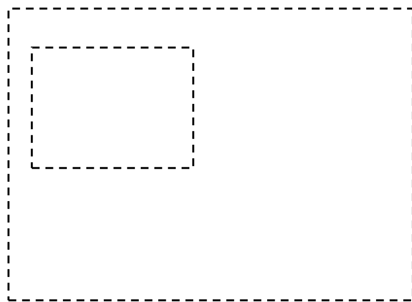
Llegamos al mismo problema que se aborda en el caso de los Stick-Ons en texto pero en un caso más general. No podemos saber si son dos sombras de dos Stick-Ons o, tal vez menos probable pero posible, tres sombras que coinciden justo en el borde derecho del Stick-On mayor.

Revisemos entonces los casos discutidos en [1]. El primer caso consiste en poner un Stick-On que tape completamente la sombra de otro Stick-On previamente removido. Si se me disculpa, voy a volver a poner la ilustración de esta situación:

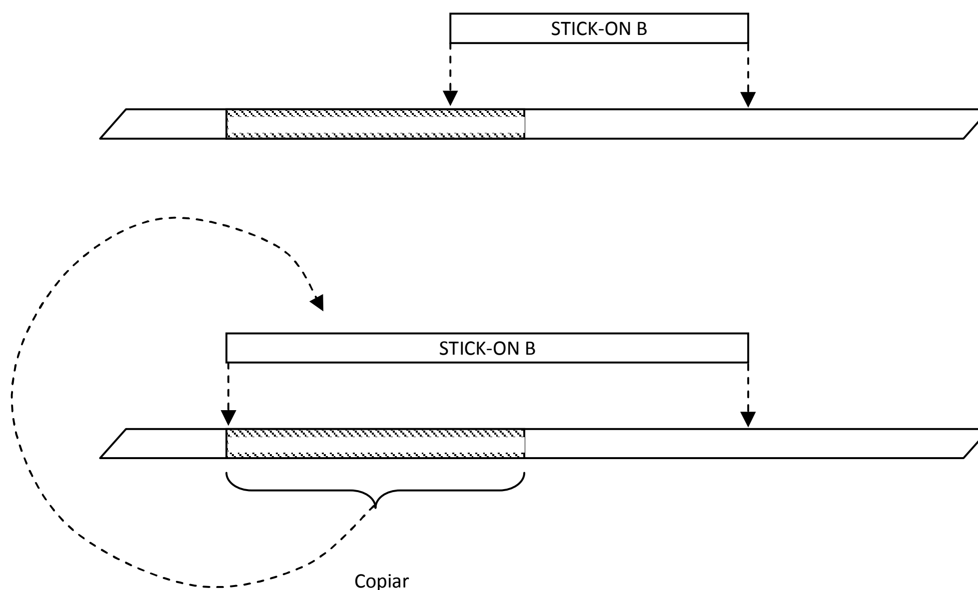


Lo que se muestra en el diagrama es la situación en que se ha removido un Stick-On y posteriormente se pega otra que lo cubre completamente. Si se retira el segundo Stick-On, B, podemos llegar al caso (a) que sería mal interpretado ya que eso representa 3 Stick-Ons adyacentes. Si sólo se muestra la sombra del recién despegado Stick-On B perderíamos información ya que no podríamos volver al Stick-On A. La solución propuesta consiste en que el sistema, justo antes de pegar el Stick-On B pegue de vuelta el Stick-On A. Al remover el Stick-On B llegaríamos a lo ilustrado en (c), veríamos el Stick-On A y todo marcharía bien.

Esta solución es sencilla y elegante, pero en general no aplica en dos dimensiones, salvo casos muy especiales. Si se pone un Stick-On que cubre completamente una sombra la situación normalmente será un caso donde no hay ambigüedad, por ejemplo una de las situaciones ya descritas:



El caso del solapamiento parcial es el que tendremos la mayoría de las veces en dos dimensiones. Veamos la situación en texto:



La situación mostrada refleja el caso en que se pega un Stick-On sobre una sombra, pero este no alcanza a cubrirla totalmente. Si no hacemos nada, el usuario podría pegar el Stick-On B y luego sacarlo, llegando a la figura que deseamos evitar de varias sombras paralelas. La solución en este caso se muestra en la parte inferior y consiste en agrandar el Stick-On B de tal modo de que cubra completamente la sombra, y así el problema se reduce al primer caso analizado. Si el Stick-On sobresale a la izquierda la situación es simétrica.

Desafortunadamente, el agrandar el Stick-On para que cubra la sombra no funciona tan bien en dos dimensiones. Conceptualmente la solución sigue siendo correcta, pero en la práctica nos podemos encontrar con situaciones desagradables que quisiéramos evitar. Veamos primero una posibilidad que puede ser problemática:

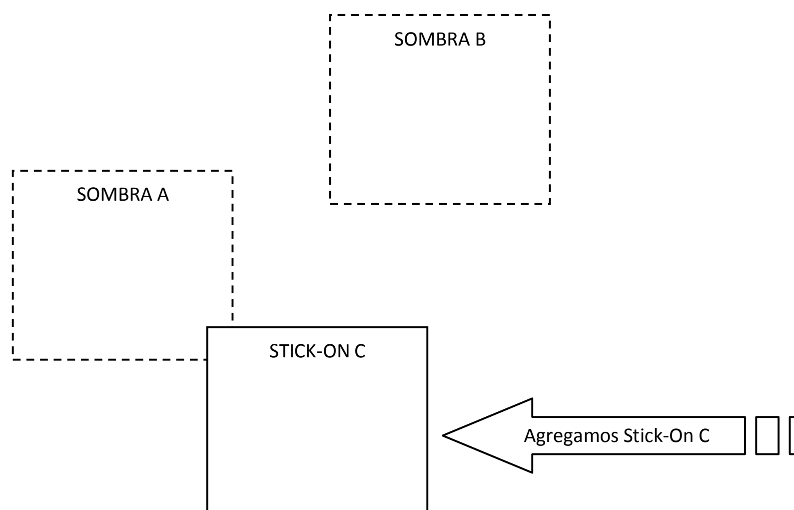


Ilustración 52. Pegado de Stick-On sobre sombra

La figura representa un problema para el esquema de agrandar el Stick-On de manera de cubrir la sombra. En principio el conflicto viene de usar Stick-Ons rectangulares, tal vez se podría explorar alguna alternativa, pero si nos ajustamos a rectángulos puede surgir el siguiente efecto al expandir el Stick-On C para que cubra la sombra A:

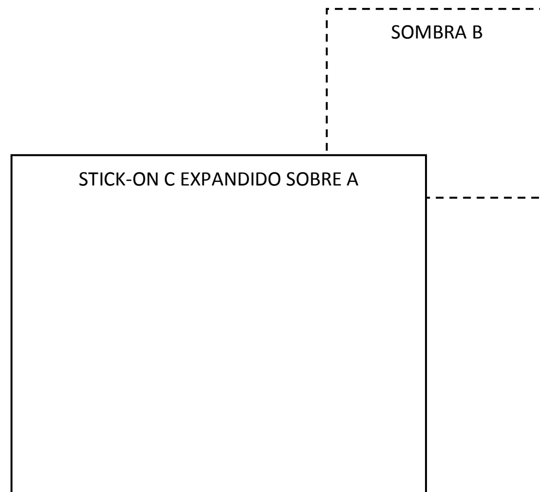


Ilustración 53. Stick-On C ha crecido para cubrir sombra A

El Stick-On C ha crecido bastante para cubrir la sombra A, pero no resolvimos el problema ya que al crecer se ha repetido la situación, ahora con la sombra B. Volviendo a crecer tendríamos un Stick-On mucho mayor que el original:

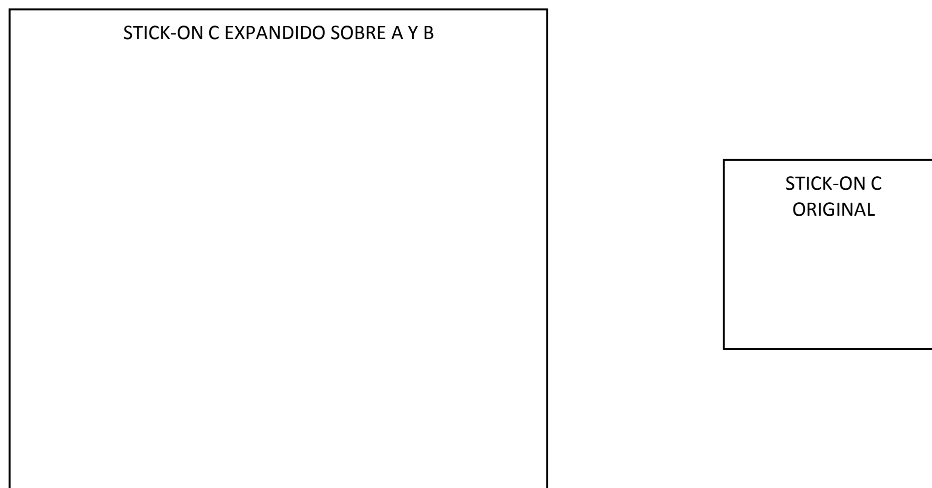


Ilustración 54. Stick-On después de crecer comparado con el original

Esta situacion de tener que crecer ya que tomamos una nueva sombra puede seguir repitiéndose, nada lo impide. El problema es que se pierde una de las gracias del Stick-On que es indicar en forma sencilla visualmente *donde* está el cambio propuesto. Un Stick-On gigante se aproxima a simplemente a una nueva versión del documento completo y no era eso lo que se buscaba cuando se puso el Stick-On C.

Esta pérdida es más o menos importante, sobre todo considerando que vino de un pequeñísimo solapamiento entre el Stick-On C y la sombra del Stick-On A. ¿Qué alternativa podríamos considerar en vez de extender el Stick-On? Lo que nos gustaría proponer es hacer más fuerte la metáfora de sombra y efectivamente reflejar las sombras como serian las sombras en el mundo real. Si dos sombras se solapan dan lugar a un tono más oscuro. Para ello descansaremos en las capacidades de Flash para tener sombras semitransparentes. Esperamos que para el ojo sea fácil distinguir la configuración de Stick-Ons que dio origen a las sombras que se ven en pantalla ya que estamos copiando como sería la situación en el mundo real, incluso podemos crear sombras “suaves” en que el contorno es un degrade, lo que debe ayudar aún más a identificar las formas.

6. Editor de Diagramas

Revisaremos el diseño y la funcionalidad de la parte del sitio web en donde se trabaja directamente con los diagramas. La mayoría de las imágenes han sido tomadas del sitio web.

Elementos diagramación

En una primera etapa se ha desarrollado la aplicación con una funcionalidad básica de diagramar. Se ha intentado generar un diseño flexible y escalable para poder potenciarlo posteriormente, y no encontrarnos con demasiadas dificultades al intentar agregar Stick-Ons en las siguientes etapas.

Diagramas de clases

El modelo base del sistema de diagramación en su parte gráfica se puede resumir en el siguiente modelo:

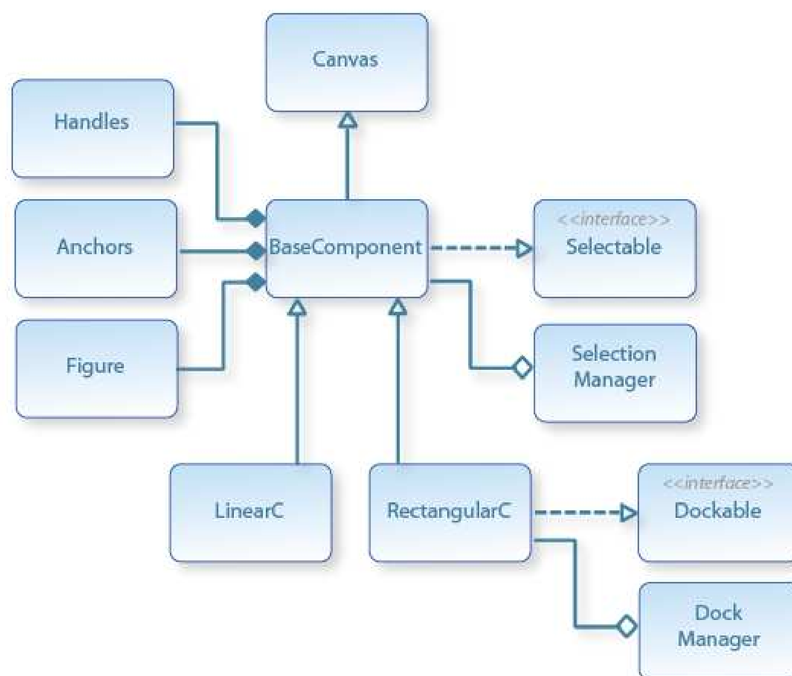


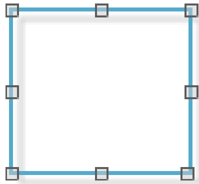
Ilustración 55. Diagrama básico de clases

BaseComponent

La componente principal es *BaseComponent*, que hereda del elemento flash Canvas. Una de las virtudes del Canvas es que nos permite posicionar libremente dentro de él otros elementos. En

este caso, deseamos poder manejar 3 tipos de elementos distintos dentro de cada elemento del diagrama: *Handles* (manillas), *Anchors*(puntos de anclaje) y la *Figura*.

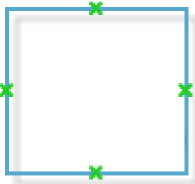
Handles



Son las manillas usadas para modificar el tamaño de un objeto del diagrama. Una de las clases más sencillas implementadas. La funcionalidad actual que tiene consiste en poder dibujarse en cierta ubicación con la forma especificada. Puede ser un cuadrado como en la figura o se puede extender su funcionalidad para que sea un círculo. Permiten que el usuario interactúe fácilmente con el elemento gráfico con solo una operación de arrastrarlos.

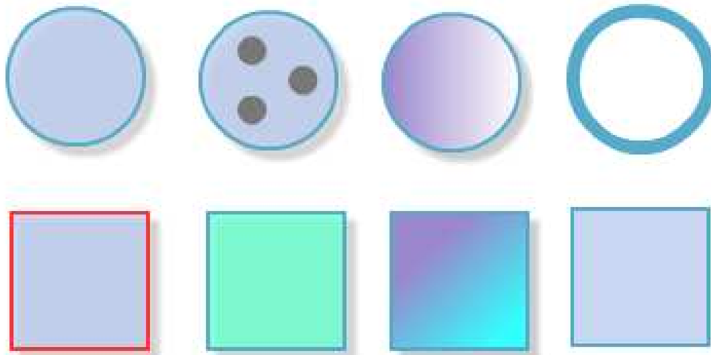
Están implementados como una lista de *Handles* dentro de la componente base, pero solo se crean efectivamente dentro de sus descendientes LinearC y RectangularC.

Anchors



Puntos de anclaje. Se activan y vuelven visibles cuando el *DockManager* detecta que se está realizando una operación de anclaje. Esto significa que se está arrastrando uno de los extremos de un conector y ha entrado dentro del rango de alguno de los puntos de anclaje o Anchors.

Figure



La figura permite dibujar el elemento gráfico. No existen restricciones de forma, el desarrollador debe heredar de alguna de las clases *LinearC* o *RectangularC* y sobrescribir el método *drawFigure* para tener una forma especial, por ejemplo un cuadrado, un círculo, estrella o figuras más complejas. Contiene además la lógica para poder aplicar estilos propios de Flash, como color del borde, grosor del borde, si tiene sombra, color de la sombra, ángulo de la sombra, gradiente, etc.

LinearC

Hereda de una de las componentes base (*BaseComponent*). Implementa la funcionalidad de elementos lineales, por ejemplo una flecha. En la implementación actual, esta clase es especializada en dos formas distintas, una flecha simple con origen y destino y una flecha con ángulos rectos. Una característica especial de los objetos de este tipo es que la componente *DockManager* reconoce cuando se está arrastrando uno de los extremos y si encuentra que esta a una distancia menor a una constante dada de un punto de anclaje (*anchors*) ejecuta una acción de anclaje, ligando dos objetos distintos de modo que si se mueve uno, esta acción es informada a todos los posibles elementos lineales que están anclados en el elemento para que se ajusten en caso de ser necesario. Revisemos algunos ejemplos gráficos de elementos que implementan este elemento.

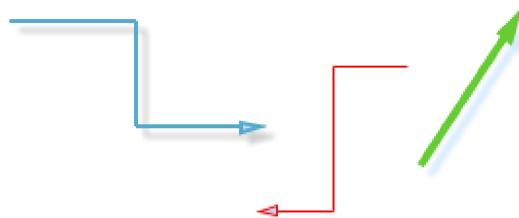


Ilustración 56. Ejemplos de estilos gráficos

Estos son algunos de los conectores que se pueden hacer actualmente con el sistema. Son definidos por un origen y destino y la forma es dada al sobrescribir el método *DrawFigure*. Los

colores y el estilo son manejados por las propiedades comunes a todos los elementos gráficos implementados. Por eso, si se desea una flecha de mayor grosor o de otro color, o sin sombra, es manejado por el sistema. Si se desea hacer una flecha curva, se deberá heredar la clase LinearC y sobrescribir el método DrawFigure e implementar el algoritmo que dibuje un spline o la curva que se desee.

RectangularC

Componente rectangular, al igual que LinearC hereda de BaseComponent, su función es manejar los elementos rectangulares. Aunque en el fondo la clase LinearC también es un elemento rectangular en el fondo, el objetivo de RectangularC es proveer la funcionalidad básica para elementos descritos por cuatro puntos, como elipses, rectángulos, transiciones y estados en redes de Petri, etc.

Interfaces Selectable y Dockable

Al igual que JAVA, Actionscript no permite herencia múltiple. Sin embargo, provee Interfaces con lo que se puede tener algo similar. En este caso, los elementos del diagrama, ya deben implementar las interfaces Selectable y Dockable. Estas interfaces definen los siguientes métodos:

```
public interface Dockable
{
    function showDockPoints() : void;
    function hideDockPoints() : void;
}
```

```
public interface Selectable
{
    function select() : void;
    function deselect() : void;

}
```

Los elementos que implementan estas interfaces pueden ser usados por los administradores de Selección y de Dock. Veamos un poco más de que se tratan.

SelectionManager

Cuando selecciono un objeto haciendo *click* con el mouse, el comportamiento esperado es que muestre las manillas o *anchors* para que pueda modificar su tamaño y forma. Sin embargo, existe otro evento que debe ocurrir, y es que cualquier otro elemento que haya estado seleccionado debe desmarcarse. ¿Dónde debe ir ésta función? No parece lógica agregarla en la clase del objeto ya que cada objeto tendrá que estar enterado de que otro objeto esta

seleccionado para avisarle que hubo una nueva selección y debe desmarcarse. El enfoque que elegimos fue usar una clase *SelectionManager*, sobre la cual se registran los elementos que implementan la interfaz *Selectable* y que se encarga de indicarle a cada objeto si debe estar seleccionado o deseleccionado. Adicionalmente, cuando deseamos modificar propiedades de un objeto usando el sistema, se le consulta al *SelectionManager* que elemento, de haberlo, tiene seleccionado el usuario.

DockManager

Clase algo más complicada que *SelectionManager*, maneja los eventos de *Docking*. Esto es, si estoy arrastrando un extremo de un conector y me acerco a un elemento que acepte *docking* (RectangularC) este debe desplegar sus puntos de Docking, y en caso de acercarme a alguno de los puntos de dock o anclaje a menos de una distancia dada, debe anclar el extremo del elemento arrastrado al punto de anclaje seleccionado.

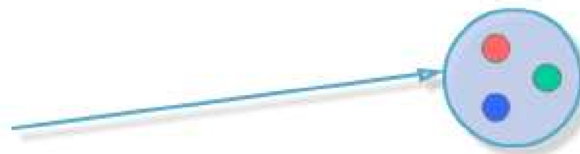
Ejemplo

Veamos un ejemplo simple que ilustre los elementos mencionados. Tenemos una situación inicial en que están dibujados un conector y un elemento de una red de Petri, un Estado con 3 tokens de color:

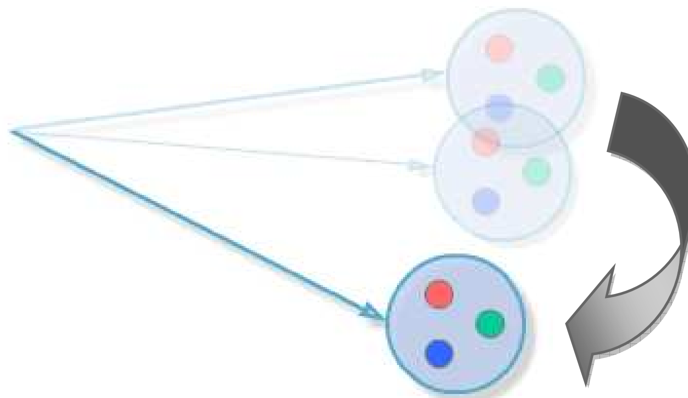
Ambos elementos al ser creados fueron agregados a una lista interna que tiene la clase SelectionManager. Por eso, al pinchar sobre la flecha se activan las manillas para que podamos moverla. En caso de que otro elemento este seleccionado, esa selección desaparece:

Al tomar una de las manillas y arrastrarla hacia el circulo (Estado) cambia la forma y orientación de la flecha. Si nos acercamos lo suficiente al elemento, se activan los puntos de Dock debido a que el DockManager encuentra que un extremo de un conector se está arrastrando cerca de un elemento que acepta anclaje:

Si nos acercamos más aún a alguno de los puntos de anclaje (cruces verdes), se deberá producir un evento de docking real:



Ahora estos elementos están ligados. Esto quiere decir que si muevo el círculo, todos los elementos que tenga anclados se reajustarán correspondientemente:



Diseños alternativos del modelo

Inicialmente el diseño no era como el anteriormente descrito. La primera iteración del sistema intentaba partir de una clase básica que tuviera la funcionalidad de las manillas desde un principio, y después al ir extendió la clase (mediante herencia) ir agregando funcionalidad y creando una jerarquía que permitiera separar conectores de figuras conectadas, siempre manteniendo la funcionalidad bien encapsulada, ojalá en las clases antecesoras. Sin embargo, al poco andar de esta implementación nos encontramos sobrescribiendo la mayoría de los métodos solo por el hecho de haber puesto la funcionalidad de arrastrar las manillas demasiado tempranamente. Los elementos rectangulares, con cuatro manillas funcionaban bien, pero al intentar extender el modelo a elementos lineales con dos manillas no se pudo preservar casi nada del código original teniendo que sobrescribirlo casi todo, por lo que se cambio al enfoque al actual, en que si bien las manillas ya existen en la clase básica (*BaseComponent*), la funcionalidad de ellas y su cantidad es definida solo a la altura de elementos más especializados (aunque no tanto aún) como son *LinearC* y *RectangularC*.

Diseño de un diagrama

Antes de revisar las funciones del editor de diagramas quisiéramos tomarnos un minuto para detallar lo que es un diagrama en el contexto de la aplicación y de qué manera fue implementado. La noción común que se tiene de un diagrama es un conjunto de elementos gráficos, como círculos, cuadrados, flechas, etc., distribuidos en un espacio dado. Para nosotros esto sigue siendo cierto, pero tenemos un par de funcionalidades que afectan, más que a la definición semántica de diagrama, a su representación interna en el sistema. Tenemos un elemento especial que modela un "proceso" (en una Red de Petri). Gráficamente es un cuadrado con borde doble, pero conceptualmente representa a un sub-proceso de mayor complejidad, pero cuyo detalle está escondido bajo este cuadrado de borde doble. Dentro del sistema se puede efectivamente entrar a este detalle (zoom in) y se ve un nuevo diagrama, el cual a su vez puede tener otros "procesos" a los cuales también se puede entrar y tener otra vista de diagramas, o bien se puede hacer "*zoom out*" para salir del proceso actual e ir al padre. Esta funcionalidad le da características de árbol a nuestros diagramas, en donde cada elemento que sea del tipo proceso nos lleva a un nuevo diagrama hijo del diagrama actual, más precisamente hijo de un elemento del diagrama actual.

Otra funcionalidad que afecta nuestra definición de diagrama (no la definición semántica de la palabra diagrama, sino su representación interna y en definitiva la estructura de datos utilizada) es el uso de Stick-Ons para tener un control local de versiones. Los Stick-Ons además de tener su propia representación gráfica dentro de un diagrama, tienen asociados dos listas de elementos. Una lista con los elementos que tapa el Stick-On y otra lista con los elementos que el Stick-On propone, dando origen a dos versiones alternativas. Por ejemplo, si se mueve un elemento de posición en el diagrama y se está pegando un Stick-On, se almacena el elemento original en la lista de elementos tapados y se guarda una copia como elemento propuesto. Así, cuando el Stick-On se despega se despliega el elemento original, y cuando el Stick-On es vuelto a pegar se muestra el elemento alternativo, la copia que ha sido movida.

Lo anterior se resume en el siguiente diagrama, en el cual tenemos un objeto central, que llamamos nodo. En un caso simple, de un diagrama sin profundidad y sin Stick-Ons el nodo solamente tiene un listado de elementos que representa lo que el usuario ve en pantalla. El nodo raíz no tiene nodo padre y tampoco está asociado a un "proceso" en particular. Si se agrega un nuevo proceso y para este nuevo proceso se definen elementos gráficos esto se refleja agregando un nuevo nodo hijo al nodo raíz, y este nodo hijo tiene su propio arreglo de elementos y eventualmente puede tener una serie de hijos también.

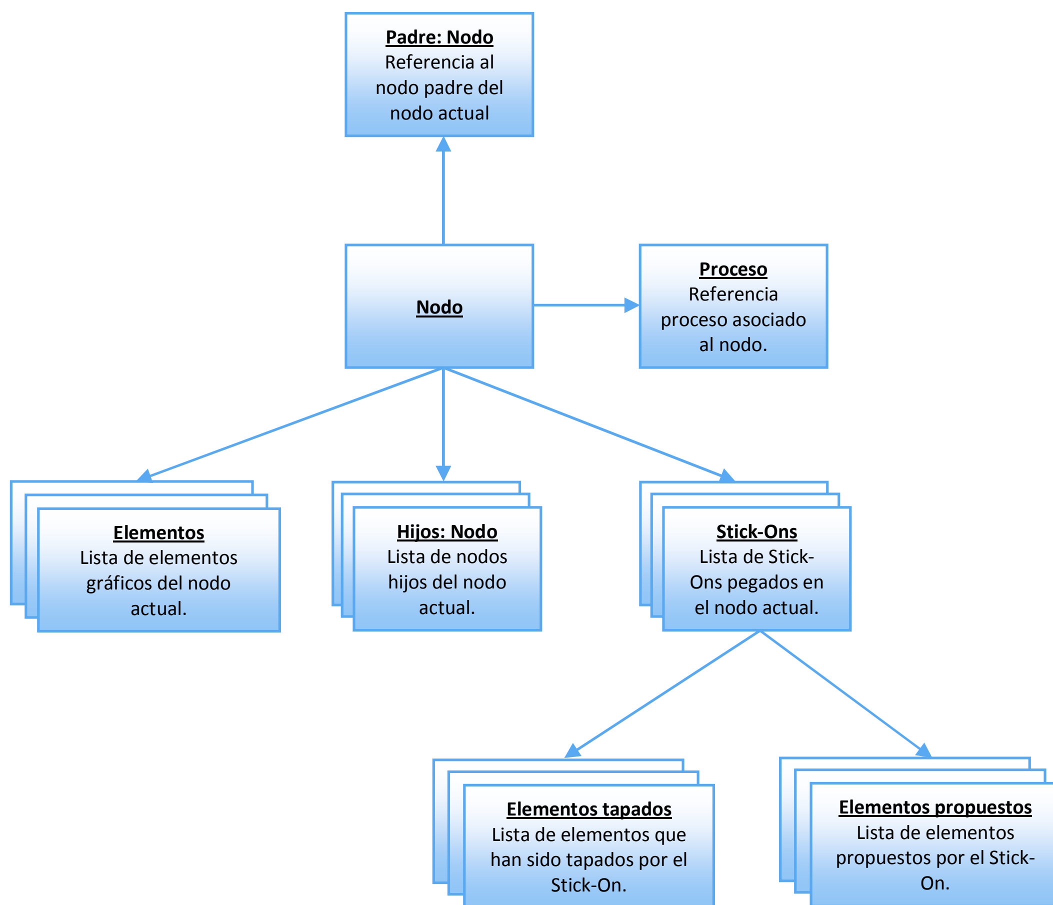
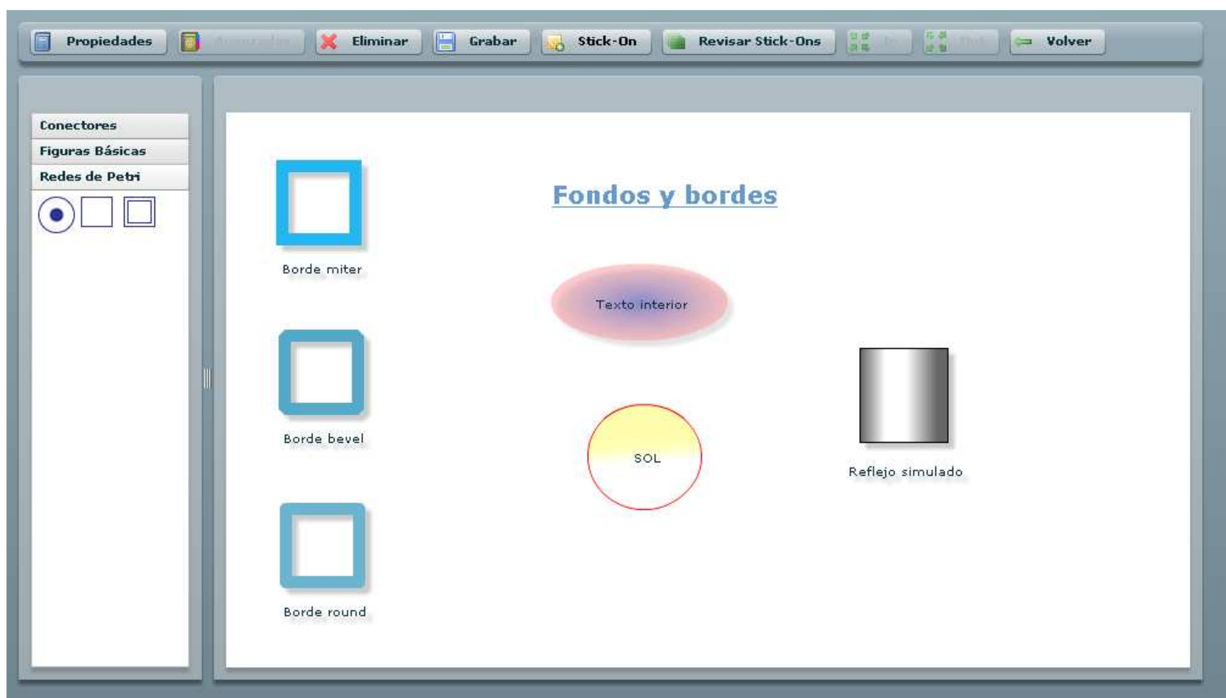


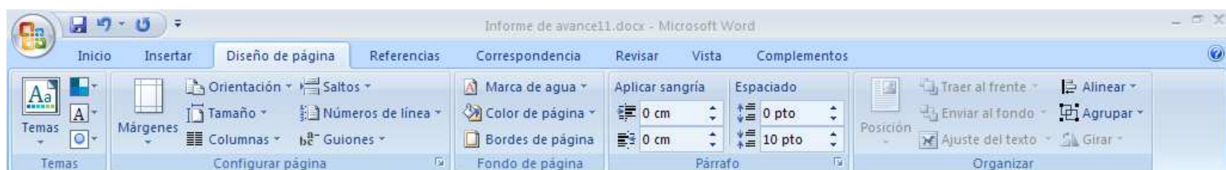
Ilustración 57. Estructura de un diagrama

Editor

Si se desea crear o modificar un diagrama, el sistema pasa de la vista de diagramas en miniaturas ya documentada con anterioridad a un nuevo estado que no hemos revisado aún y que nos permite el manejo de los elementos del diagrama, tales como las propiedades gráficas, el poder agregar elementos, eliminar, grabar, etc. Este estado es lo que denominaremos como el editor de diagramas, o simplemente editor.



El editor se encuentra dividido en tres secciones, como se aprecia en la figura. En la parte superior están ubicadas las herramientas. En estos momentos no se tienen demasiados botones, pero si se desea seguir agregando funcionalidades va a ser necesario replantearse el esquema. Lo tradicional sería agregar un menú justo arriba de los botones y dejar los botones sólo como íconos parea funciones comunes, por ejemplo grabar, borrar, etc. Sin embargo después de usar la nueva interfaz de Office 2007 creo que yo recomendaría intentar implementarla aquí, ya que tenemos los elementos necesarios y el espacio requerido.



La imagen muestra la nueva interfaz, en este caso de Word 2007. Esta divido en paletas, no muy altas, y en cada paleta tenemos botoneras y controles de interfaz en general más complejos que los que se tienen en un menú tradicional.

Los elementos que quedan están bajo el menú. En la parte izquierda, y como es tradicional en las herramientas de diagramación se tiene un control de "acordeón", el cual permite tener una amplia variedad de elementos gráficos agrupados por categorías, usando el mismo espacio físico. Nosotros actualmente tenemos tres categorías, para conectores, elementos comunes y elementos de Redes de Petri. Se pueden aumentar sin problema las categorías y los elementos de cada categoría. Finalmente, a la derecha, tenemos la "tela" donde dibujaremos el diagrama. Actualmente tiene un tamaño único definido. Una posible mejora consistiría en poder cambiar el tamaño o escala, en base a distintos tamaños de papel, como A4, Carta, Legal, etc. La forma de poner un elemento que está en el acordeón (parte izquierda) en el área de trabajo (parte

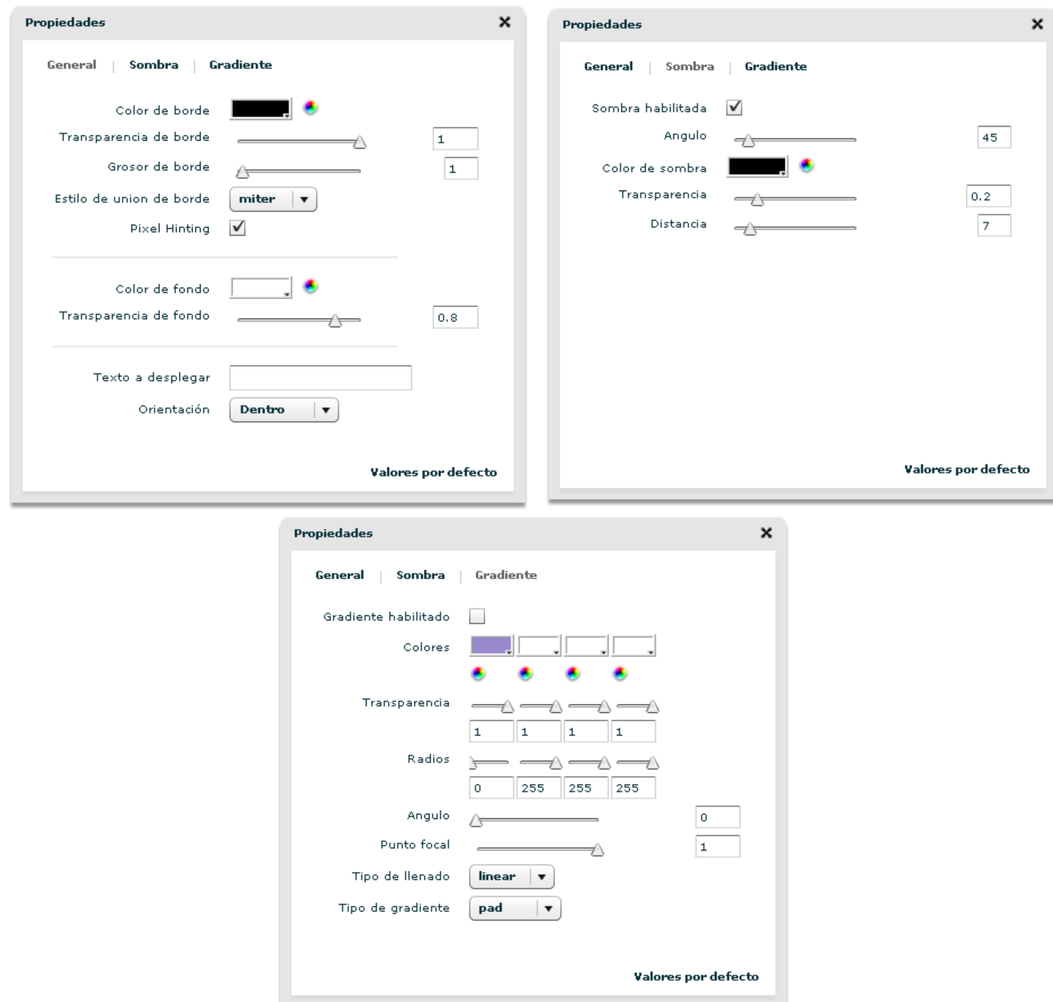
derecha) es arrástralo y soltarlo donde se desea que quede. El sistema actualmente no permite que se suelte el elemento en ningún otro lado que no sea el área de trabajo a la derecha (la tela).

Barra de herramientas

Se revisarán las funciones que están ubicadas en la barra de herramientas.

Propiedades

Este botón permite especificar el aspecto visual de las componentes gráficas. Atributos como color de borde o de fondo, nivel de trasparencia, si proyecta sombra, si el fondo se dibuja con un gradiente y entre que colores son definidos aquí. Estas propiedades son aplicadas a nivel de uno de los objetos base de las figuras, del cual heredan el resto de objetos, incluso los elementos de texto, por lo que se pueden configurar estas propiedades a todos los objetos del sistema. Podríamos decir que estas son las propiedades básicas.

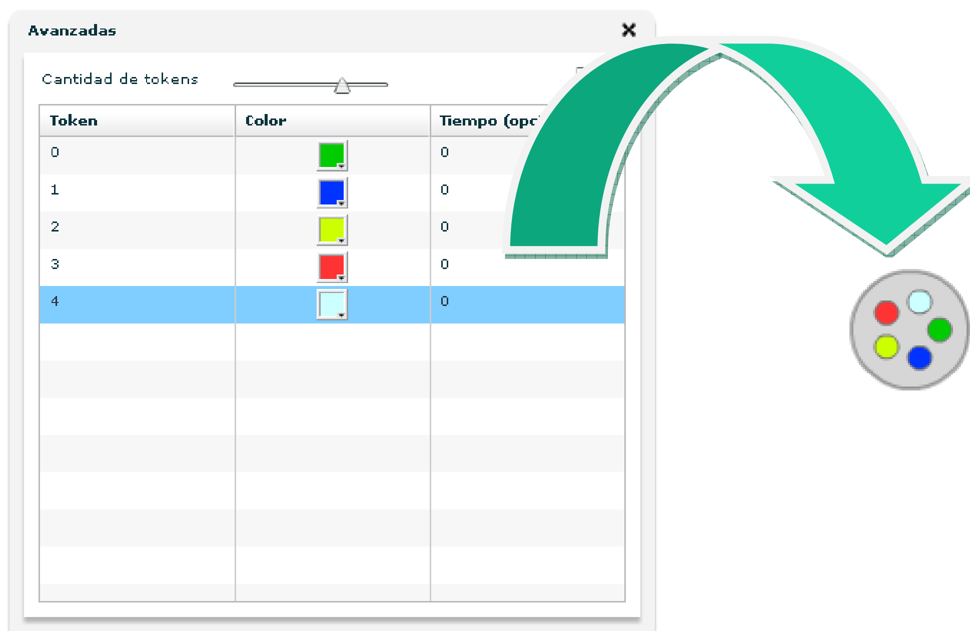


Propiedades Avanzadas

Este botón permite tener acceso a propiedades que no son comunes a todos los elementos y generalmente esta desactivado. Sólo cuando un elemento que efectivamente tiene propiedades avanzadas es seleccionado este botón se activa. Actualmente solamente los elementos de *Texto*, *Estado* y *Transición*, estos dos últimos en la sección de Redes de Petri, tienen propiedades avanzadas.

Estado

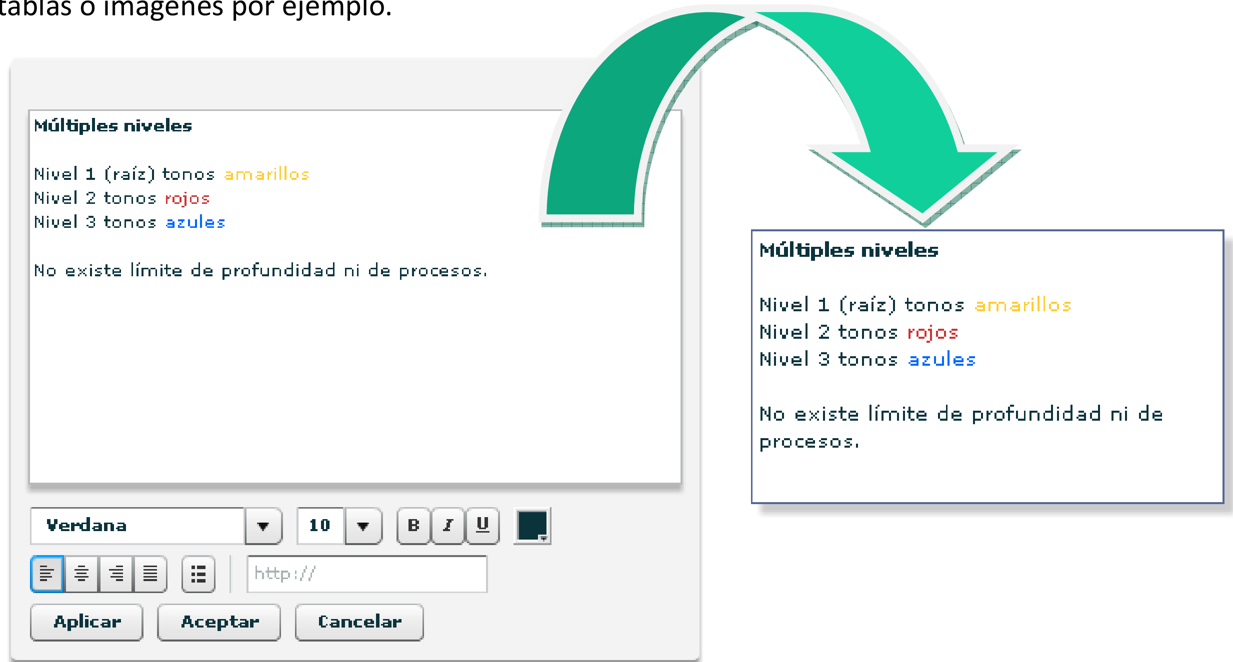
Como ya mencionamos, el elemento Estado tiene propiedades avanzadas, pudiéndose crear para un estado seleccionado una cantidad de *tokens* o fichas que va entre 0 y 7. Mayor cantidad de tokens hace que se comiencen a solapar por lo que se impuso este límite. En general, una cantidad mayor de tokens debe representarse con números, lo cual es posible usando las propiedades básicas y poniendo un texto centrado. En este contexto el límite de siete *tokens* es para la extensión del modelo de Redes de Petri para fichas de color solamente.



Texto

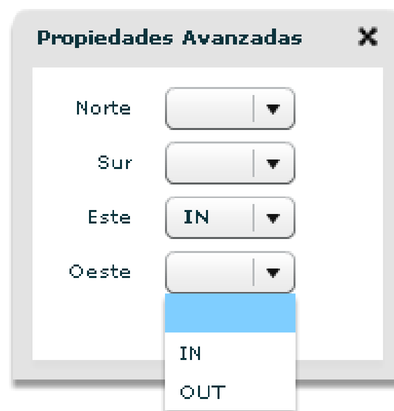
El segundo elemento que admite propiedades avanzadas es el elemento de texto. En este cuadro se puede escribir texto con diferentes colores y tamaños, pudiéndose pre visualizar los resultados (botón *Aplicar*) antes de aceptar los cambios. Esta información es guardada y tratada como HTML, por lo que aunque en este momento las opciones están limitadas a lo que permite

hacer la interfaz, si se podría potenciar este aspecto con textos más complejos, que contengan tablas o imágenes por ejemplo.



Transición

El elemento "Transición" en la sección de redes de Petri también admite propiedades avanzadas para poder contar con algunas notaciones encontradas en [12]. Dichas notaciones nos permiten diagramar elementos de los tipos *AND-Split, AND-Join, OR-Split,* etc. que no son sino representaciones de algunas construcciones encontradas frecuentemente en Redes de Petri. Para poder darle esta propiedad a un elemento Transición, en propiedades avanzadas tenemos el siguiente cuadro:



Para cada una de las cuatro orientaciones posibles de un rectángulo hay dos posibilidades, IN y OUT que significa que se diagrama un triangulo apuntando hacia fuera o hacia adentro. Con esto podemos diagramar los elementos de notación comentados en cualquier orientación, por ejemplo (elementos creados en el sitio):

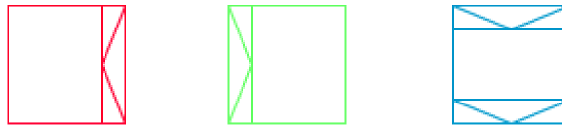


Ilustración 58. A la izquierda Este-IN, al centro Oeste-IN, a la derecha Norte-IN, Sur-OUT

Eliminar

Elimina el elemento seleccionado. En caso de estar pegando un Stick-On la eliminación de un elemento no resulta directamente en un borrado del objeto, si no que se agrega a la lista de elementos tapados por el Stick-On, de modo que cuando se despegue el Stick-On el elemento recién borrado se vuelve visible.

Grabar

Graba cambios en pantalla. Si el documento es nuevo se debe ingresar un título, el comentario es opcional. Si es que el documento no es nuevo, se proponen el título y el comentario anterior. Cada vez que se graba se genera una versión del documento, la cual es factible de ser recuperada desde la pantalla de versiones.

La pantalla que se le presenta al usuario es la siguiente:

Grabar

Título: Titulo del diagrama

Descripción: Comentario para la versión

Grabar y liberar diagrama

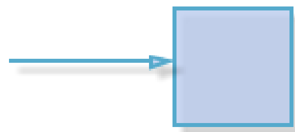
Aceptar Cancelar

La opción de grabar y liberar el diagrama pretende facilitar la operación que esperamos sea más o menos común de: tomar un diagrama para editarlo, hacer nuestros cambios y cuando esté

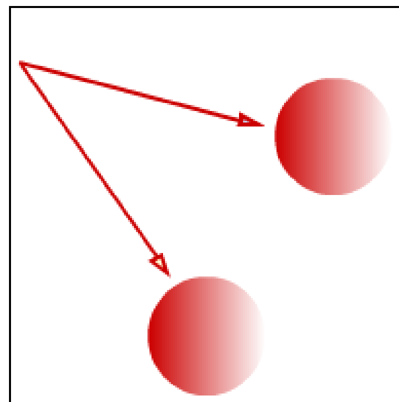
listo grabar y al mismo tiempo indicarle al sistema que no lo seguiremos modificando, lo cual lo libera para que otros usuarios lo usen.

Stick-On

Al presionar este botón se le indica al sistema que los cambios que se hagan de aquí y hasta que se vuelva a presionar el botón son cambios propuestos. El sistema los agregará y desplegará en un Stick-On el cual posteriormente se podrá pegar o despegar para ver las versiones alternativas. Por ejemplo si tenemos este elemento en pantalla y deseamos proponer una nueva versión local, debemos presionar el botón Stick-On:

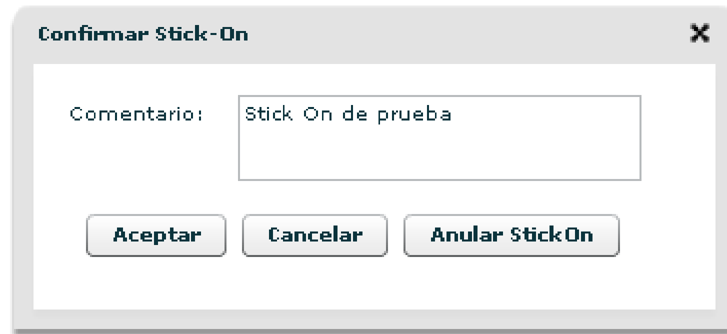


Por ejemplo, si los cambios que deseamos hacer son cambios gráficos por un lado, usar colores rojos y un gradiente por ejemplo y además cambiar el cuadrado por un círculo y agregar otro círculo, lo que se debe hacer es presionar el botón Stick-On, y ejecutar los cambios deseados, con lo que tendremos la siguiente situación:

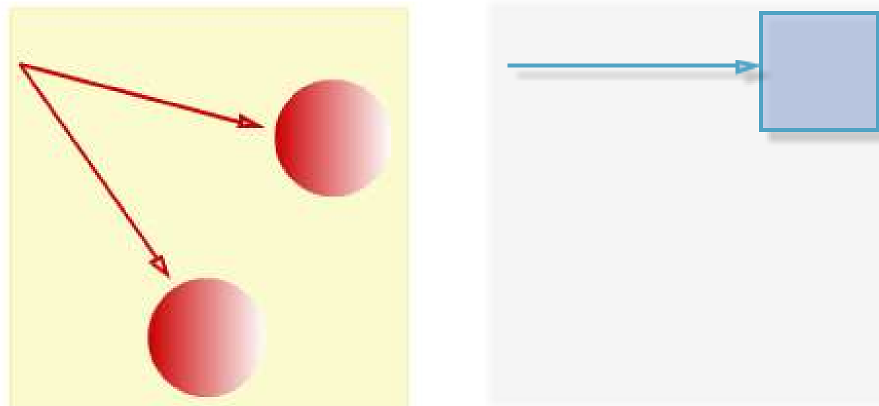


En la figura anterior ya se han hecho los cambios deseados. Como se le ha indicado al sistema que estamos poniendo un Stick-On, automáticamente se dibuja un cuadrado negro que nos va indicando donde quedará el Stick-On cuando terminemos las operaciones. Este cuadrado se ajusta dinámicamente cuando movemos, agregamos o eliminamos, y cubre la zona que ha tenido cambios, es decir cubre a los elementos originales y a los elementos nuevos. Si ya hemos

finalizado con los cambios y volvemos a presionar el botón Stick-On, se nos presente la siguiente pantalla:



La cual nos da tres opciones, *Aceptar* confirma el Stick-On y lo pega definitivamente, *Cancelar* solamente cancela y oculta este diálogo, permitiendo que se siga trabajando en el Stick-On, y finalmente Anular deshace los cambios que hemos hecho desde que empezamos apegar el Stick-On, para el caso de que nos arrepintamos de proponer los cambios que habíamos empezado hacer. Si aceptamos el Stick-On se pegará al documento y tendremos estas situaciones, dependiendo de si el Stick-On está pegado o despegado:



A la izquierda tenemos el Stick-On pegado y nos muestra los cambios propuestos. Si se "despega" queda la situación de la derecha, que nos muestra el estado anterior de esta parte del diagrama y además nos da una pequeña indicación visual (una sombra) de que hemos despegado el Stick-On.

Revisar Stick-Ons

Si se presiona el botón de revisar Stick-Ons, el sistema entra en un modo de revisión en el cual se pueden pegar o despegar los Stick-Ons usando el mouse. Además se abre una ventana en donde se pueden ver todos los Stick-Ons que se han pegado en los diagramas. Desde aquí también se puede pegar y despegar los Stick-Ons e ir directamente al proceso que contiene al Stick-On.

Ilustración 59. Herramienta lista de Stick-Ons

Este listado de Stick-Ons es más necesario de lo que podría aparecer a primera vista. Para entender porque es tan importante debemos recordar que los diagramas pueden tener una estructura de árbol y tener muchos hijos que contienen sub diagramas, y a su vez estos sub diagramas pueden tener más sub diagramas, por lo que debemos tener un lugar centralizado en donde poder ver los Stick-Ons que están pegados al documento. Por ejemplo alguien pudo poner un Stick-On a tres niveles de profundidad, por lo que no lo veremos cuando recién entramos a revisar los cambios. Es posible que ni siquiera nos enteremos de que alguien ha hecho una propuesta usando Stick-Ons al documento. Por eso este control es capaz de desplegar todos los Stick-Ons pegados, y nos permite ir al proceso que los contiene. Si este cuadro es cerrado, el sistema sale del modo de revisión y vuelve al modo de edición, donde se puede seguir trabajando normalmente con las figuras.

Las opciones que nos presenta este cuadro son “Limpiar” y “Fotocopiar”. *Limpiar* remueve los Stick-Ons del nivel actualmente visible, dejando el estado que está siendo representado actualmente por los Stick-Ons (ya sea que estén pegados o despegados) como definitivo. *Fotocopiar* lo que hace es sacar una copia del documento completo, respetando la información desplegada por los Stick-Ons, ya sea en este nivel o en otros, y genera un nuevo documento alternativo. Eventualmente se podría usar esta opción para “copiar” un diagrama, aunque no se estén usando Stick-Ons.

Zoom In, Zoom Out

Uno de los elementos gráficos que se ha implementado, el *proceso*, permite esconder un diagrama más complejo bajo un elemento aparentemente simple. Esto corresponde a la extensión jerárquica de las Redes de Petri, pero puede ser visto desde un prisma más general como la posibilidad de "entrar" a un elemento de un diagrama e ir descubriendo nuevos detalles, que si se despliegan todo de golpe se hace muy complejo de entender. Cuando un elemento del tipo proceso tiene el foco, se activa el botón "In" y se permite ingresar dentro de esta parte del diagrama. Inicialmente estará vacio, pero se puede construir un nuevo diagrama y nuevos procesos. Una vez que hemos entrado a un proceso, se activa el botón "Out" el cual nos permite salir al padre de este diagrama. Veámoslo con un ejemplo.

En las primeras secciones de este documento se comento la extensión jerárquica en las Redes de Petri, y se diagramó un ejemplo tomado de [12]. Implementáremos aquel ejemplo con el sistema que se ha construido. A continuación esta la figura que habíamos usado, creada usando las funciones de diagramación de Word y copiando el ejemplo de [12]:

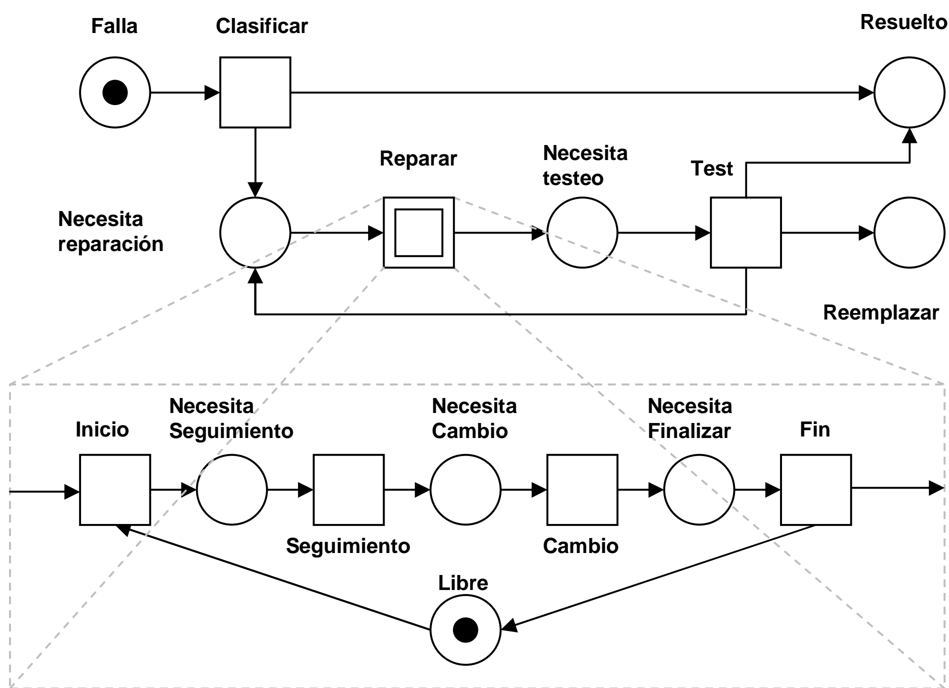
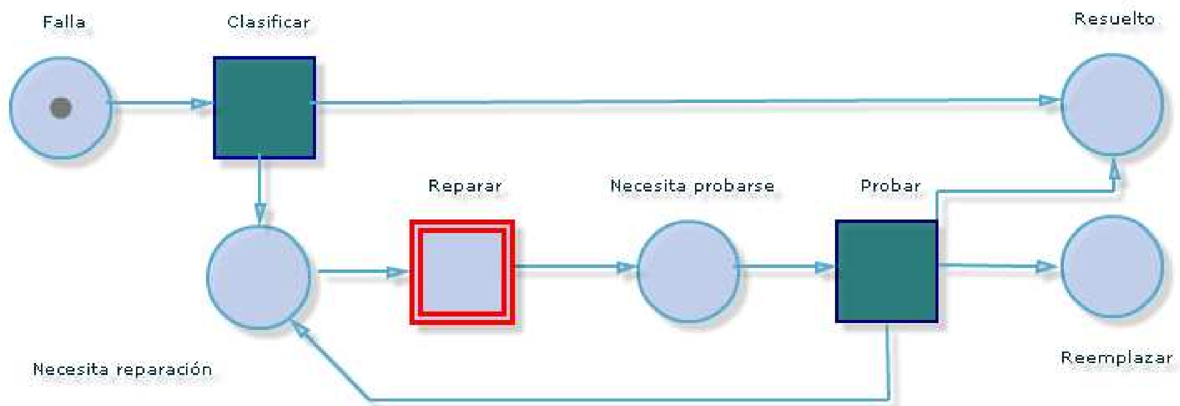


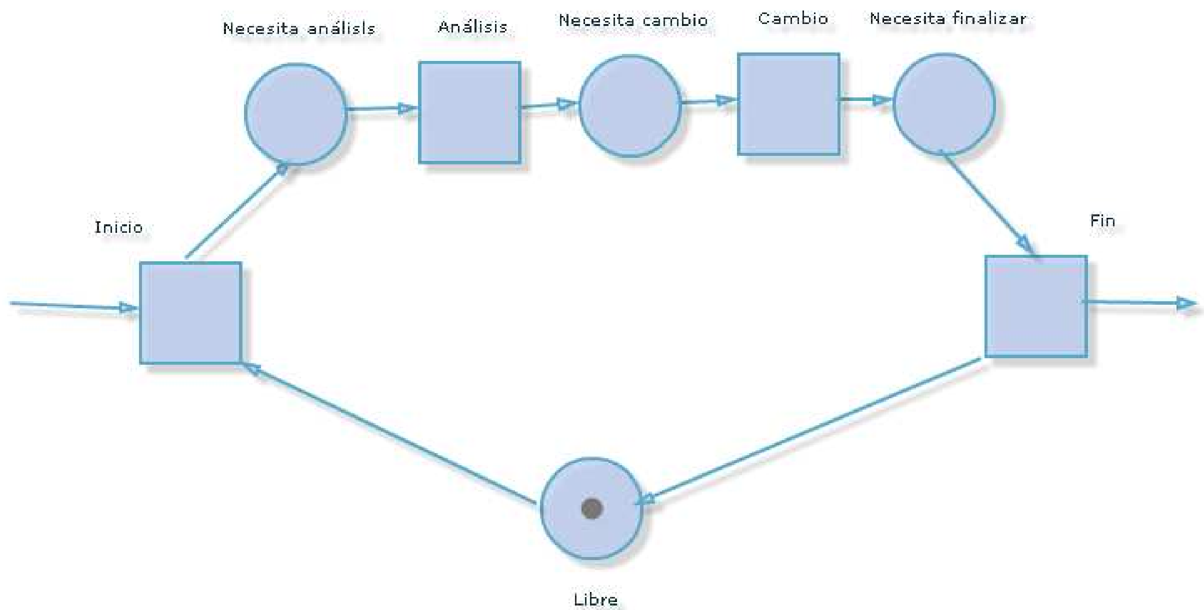
Ilustración 60. Extensión jerárquica en Redes de Petri.

Este diagrama representaba un modelo para manejar fallas técnicas en un departamento de producción. El proceso "Reparar" inicialmente es representado como una solo actividad, sin

embargo esta actividad no es indivisible y se puede descomponer como se muestra en el diagrama. Veamos el equivalente en el sistema construido:



Hemos dado mayor énfasis al elemento Proceso, poniéndole sus bordes en rojo. Si marcamos el Proceso “Reparar” y presionamos “In”, entramos al detalle del proceso, con lo que veríamos algo así:



Volver

Vuelve al modo de navegación de diagramas desde el modo de edición.

7. Discusión y Conclusiones

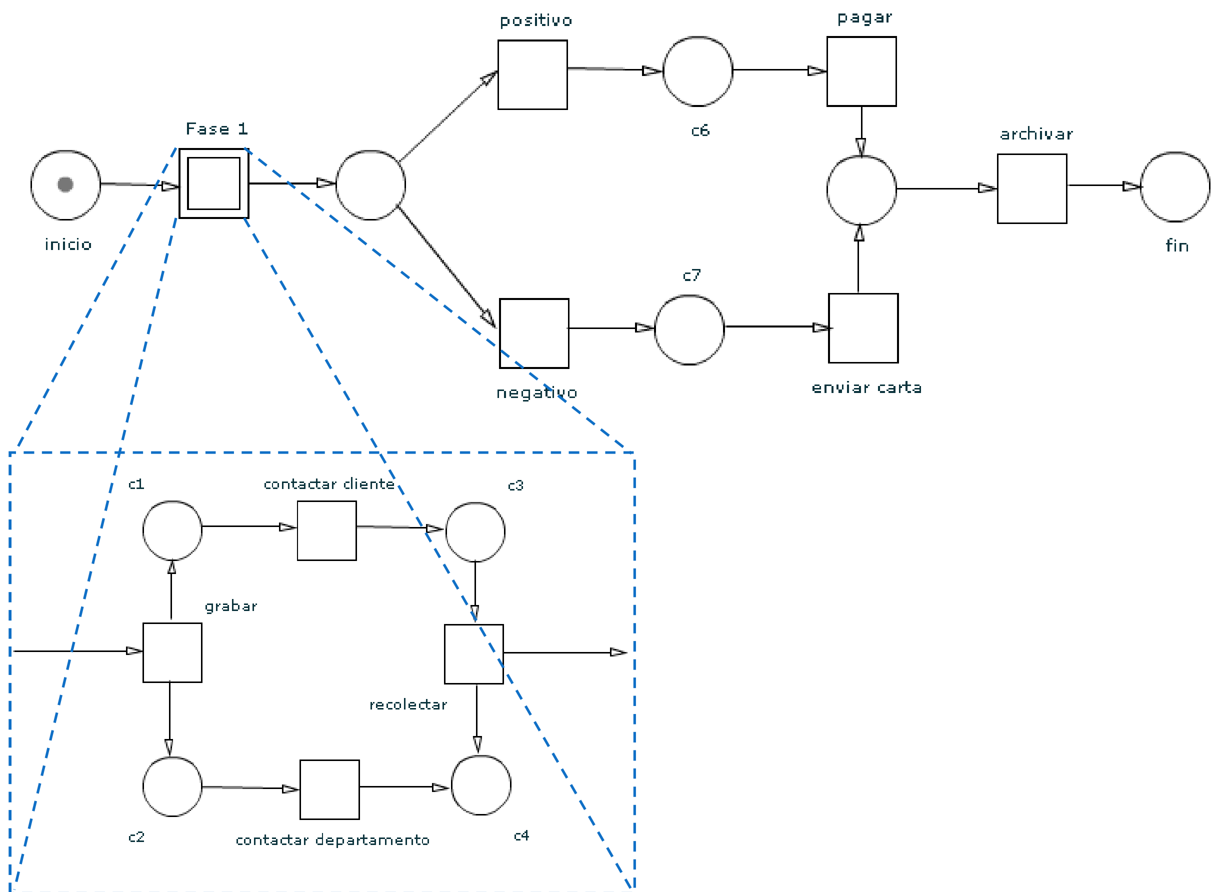
Resultados

Revisaremos algunos resultados obtenidos durante el presente trabajo de memoria. El sitio web, tal como se ha documentado en capítulos anteriores, ya ha sido construido y tiene las funcionalidades básicas necesarias para poder implementar algunos diagramas que mostraremos como resultado del trabajo. Además, revisaremos algunos casos de uso de Stick-Ons para ver cómo ha resultado nuestro diseño en la práctica.

Partamos presentando algunas redes de Petri tomadas de "*Workflow Managment*" [12]. Recordemos que uno de los objetivos era poder diagramar redes de Petri según la caracterización que hace el autor de [12]. Se han seleccionado algunos diagramas que varían de sencillos a complejos intentando abarcar la mayor cantidad de elementos. **Todos los diagramas que se mostrarán a continuación fueron construidos en el sitio web**.

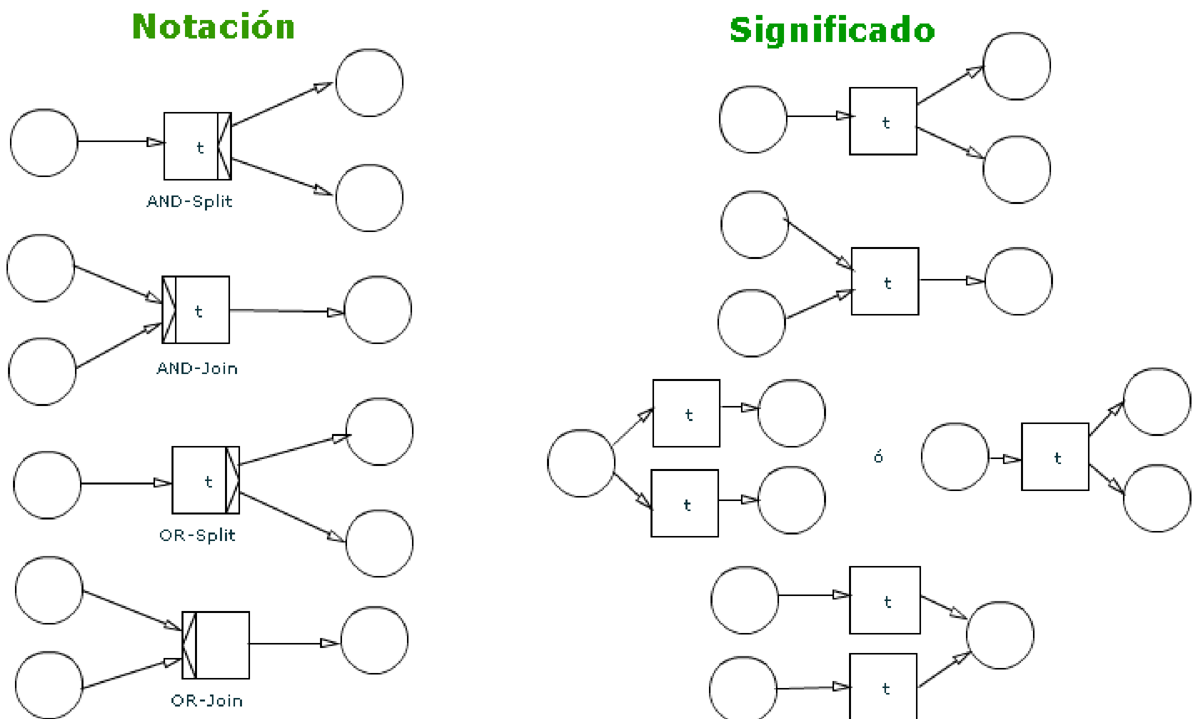
Manejo de quejas

La siguiente red de Petri modela el proceso de manejo de quejas. Corresponde a la figura 2.14 de [12] y es usada además para presentar un subproceso llamado “Fase 1”. La presentación en papel difiere de lo que tendríamos en el sitio web ya que acá tenemos el proceso principal y el subproceso detallados al mismo tiempo. Dentro del sitio web es necesario “entrar” en forma interactiva al subproceso “Fase 1” para poder ver la parte inferior de la figura y se pierde el proceso actual. Por supuesto se puede “salir” y se llega al estado anterior. En la figura presentada se han agregado las líneas punteadas azules usando el procesador de texto para que quede más claro el origen y el subproceso.



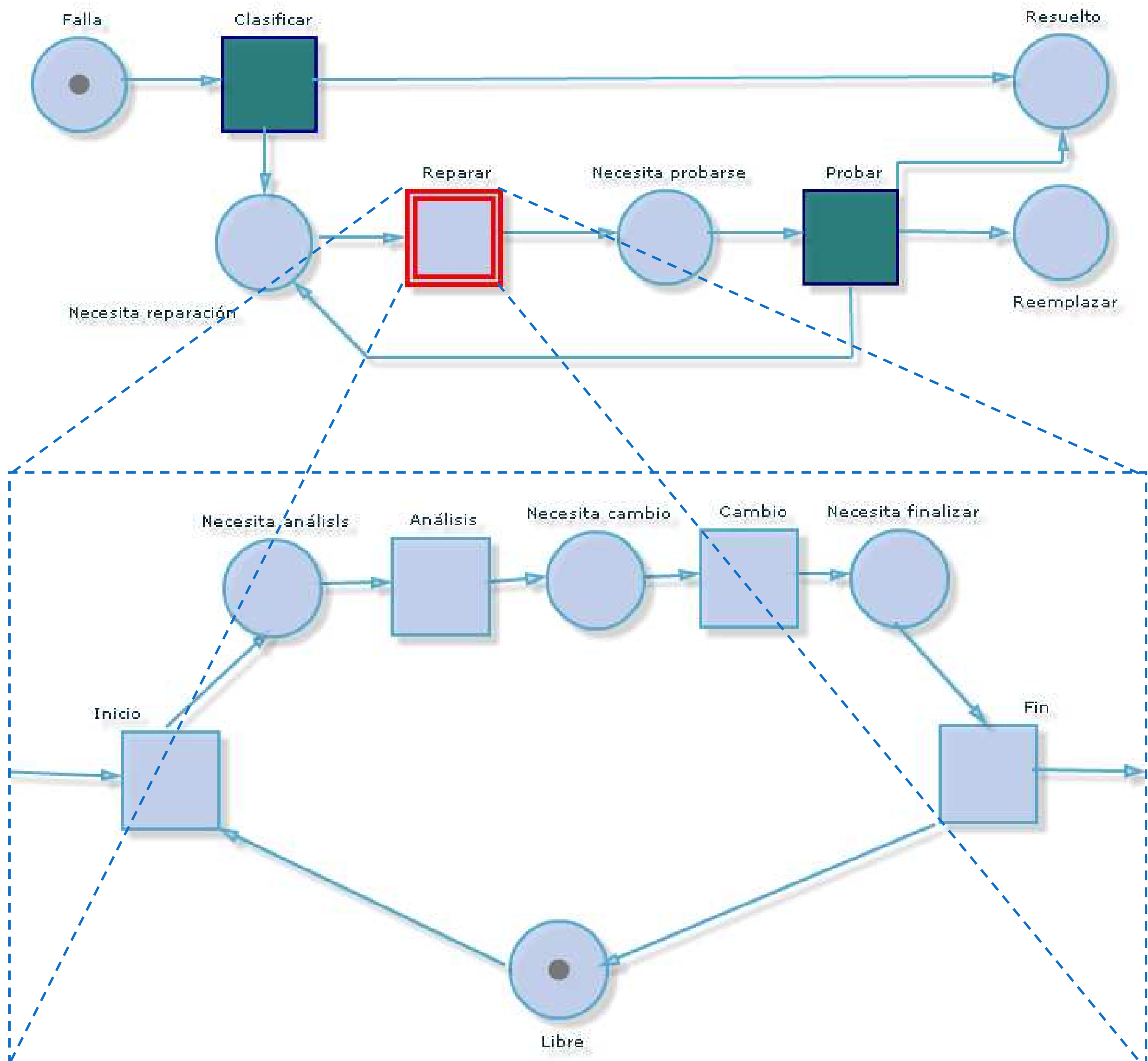
Notación para construcciones comunes

Los elementos gráficos *"transiciones"* implementados en el sitio web pueden representar algunas construcciones comunes como "AND-Split". En realidad se puede representar cualquier combinación, para ello tenemos cuatro variables en "Propiedades avanzadas" que nos permiten decir si para la transición seleccionada tiene un símbolo especial al norte, sur, este u oeste, y si es que va hacia adentro o hacia fuera (OUT o IN en las propiedades). Por ejemplo en el diagrama que se muestra a continuación, las propiedades de los elementos a la izquierda serían, de arriba hacia abajo, ESTE-IN (AND-Split), OESTE-IN (AND-Join), ESTE-OUT (OR-Split), OESTE-OUT (OR-Join). Este diagrama fue tomado de la página 59 de [12] y es el diagrama usado para explicar estas construcciones y fue replicado en el sitio web:



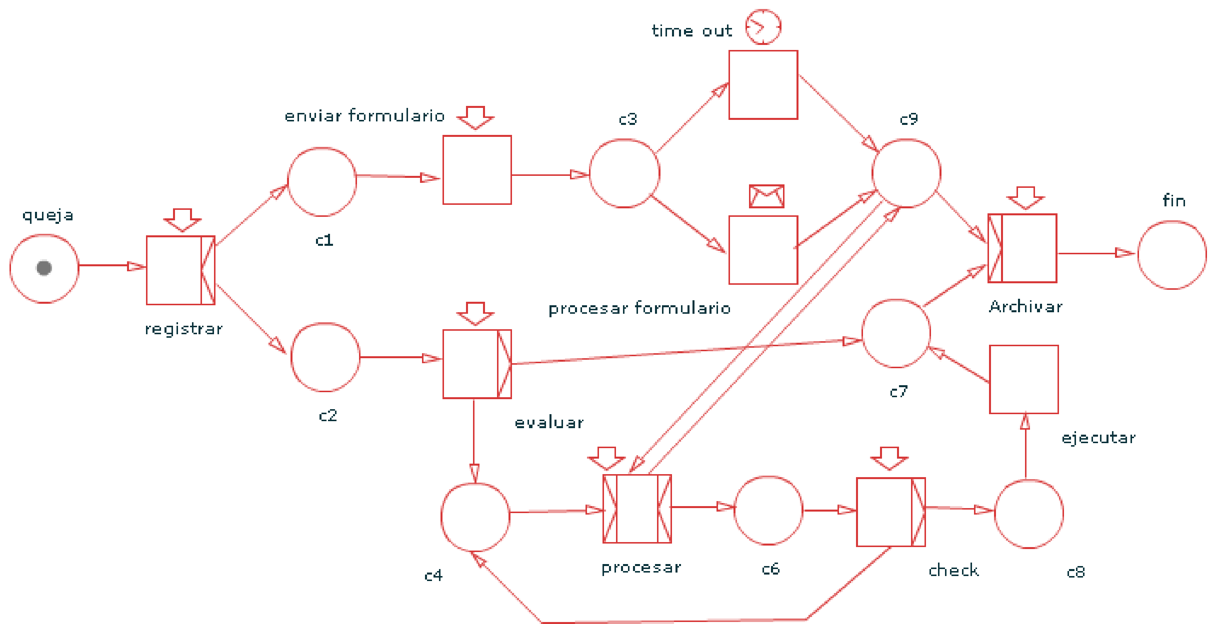
Resolver Falla

Este diagrama ya ha sido mostrado con anterioridad durante el documento debido a que fue de los primeros realizados y se usó para documentar la función de entrar y salir a un proceso. Como esta tomado del libro "*Workflow Managment*" [12] decidí incluirlo de nuevo como resultado. Al igual que previamente se hizo con "Manejo de quejas" agrego manualmente líneas punteadas azules para representar de donde viene el subproceso. Este ejemplo está en la página 47. La diferencia en colores y estilos es arbitraria para ejemplificar posibles cambios gráficos, aunque realmente el arte no es lo mío.



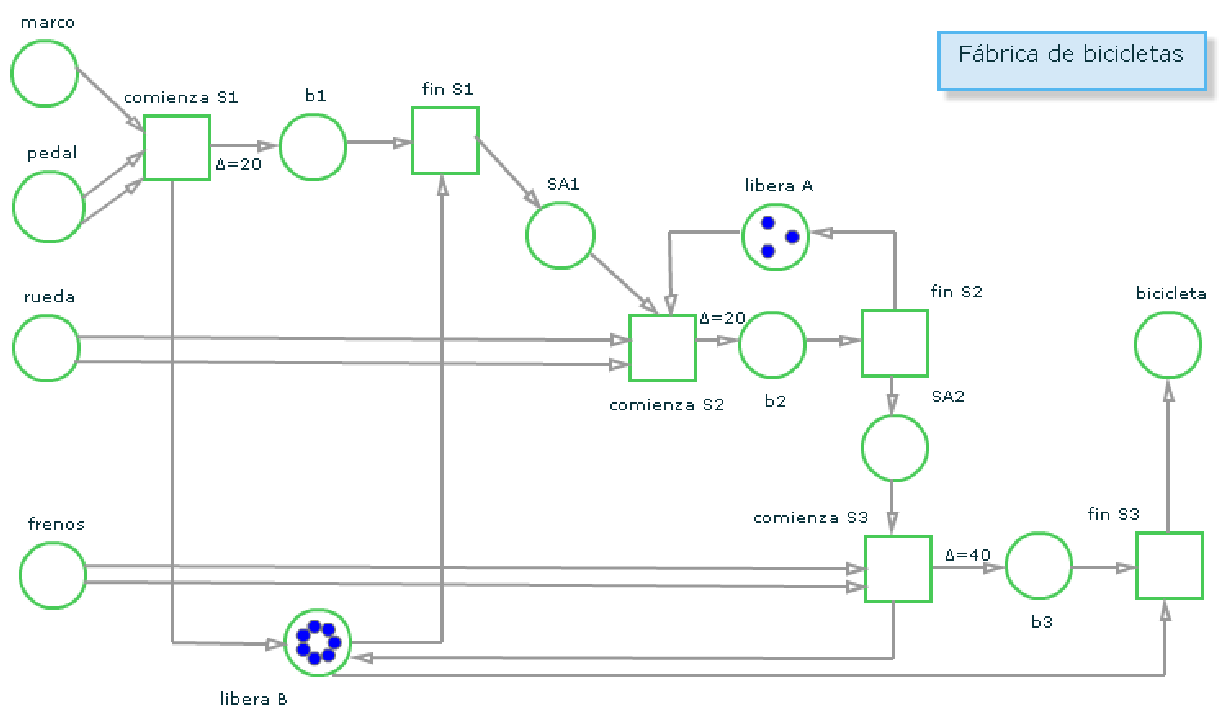
Ejercicio 2.8. Manejo de quejas

Ya graficamos una red de Petri para el manejo de quejas. Ahora graficaremos la solución al problema 2.8 del mismo capítulo que está en la sección de soluciones página 318. Es interesante este modelo ya que además de ser más complicado tiene algunos elementos nuevos y que son soportados por el sistema, los *triggers* introducidos en la página 63 de [12] (relojes, flechas y cartas).



Ejercicio 2.7 Fábrica de bicicletas

Este es el último ejemplo de los tomados de [12] y corresponde a la solución del ejercicio 2.7 en la página 315. Se cambiaron los anchos de línea a dos pixeles y se usaron colores distintos para las líneas y las figuras. No se aprecia bien, pero tiene un elemento especial y que afortunadamente se pudo poner. Como la imagen ha sido escalada para que coincida con el ancho de la hoja no se ve bien que en algunos conectores (o arcos, o flujos) se tiene un texto que dice "Δ=20" y "Δ=40". Realmente no sabía si lo podría poner tal cual, pero el elemento de texto soportó perfectamente cuando fue copiada desde Word la letra *delta*.



Sombras nada más

Durante el diseño de los Stick-Ons que implementaríamos optamos por hacer más fuerte la metáfora de la sombra e intentar dejarla tal cual su nombre, como sombras reales, a diferencia de implementaciones anteriores en que se usaban líneas punteadas. Esto por supuesto en parte debido a que la herramienta seleccionada nos permitía hacerlo sin grandes dificultades. Esperábamos que el ojo pudiera discernir las figuras con facilidad debido al parecido con el mundo real. Veamos entonces un ejemplo del sistema con muchas sombras al mismo tiempo en pantalla.

El ejemplo usado está hecho con un elemento que es movido por toda la pantalla y para cada paso se pide que el movimiento sea contenido por un Stick-On. Posteriormente se despegan todos los Stick-On y quedan una serie de sombras. Cabe hacer notar que este ejemplo es uno de los peores casos ya que con el procedimiento descrito tienden a coincidir los bordes de las sombras.

Ilustración 61. 10 sombras

La figura previa tiene 10 sombras y a pesar de ser uno de los peores casos parece sencillo identificar visualmente cada una de las 10 sombras, y por lo tanto saber en qué sector se ha despegado un Stick-On sin que resulte confuso o que de señales inconsistentes o ambiguas como podría ser el caso de una dimensión con sombras representadas por líneas punteadas. En las siguientes figuras hemos aumento el número de sombras a 15 y 23 respectivamente.

Ilustración 62. 15 sombras

Ilustración 63. 23 sombras

Con 15 sombras ya es algo más confusa la figura, pero aún se pueden identificar cada una de las 15 sombras con relativa facilidad. La figura con 23 sombras ya se vuelve un poco desordenada y está en los límites de lo que consideramos entendible. Las partes más críticas son el sector central y la parte superior izquierda. En la parte más oscura del sector central se sobreponen 7 sombras y en la parte superior izquierda 6. Al contarlas se percata uno de que ya es complicada la identificación, por lo que la sugerencia sería no pasar de cinco Stick-Ons sobrepuestos.

Existe un detalle de implementación que no ha sido documentado hasta ahora, pero que se mostró como una herramienta invaluable al momento de discernir entre las sombras. El decir cuántas sombras pasan por el punto más oscuro del sector central en la figura con 23 sombras es un verdadero rompecabezas. Incluso nuestra premisa de que no debería ser tan difícil identificar una sombra rectangular por separado en este punto falla (aunque naturalmente alguna vez tenía que ocurrir ya que era cosa de poner más y más sombras). Sin embargo nos fue fácil saber que había 7 sombras en ese punto y poder saber cuáles eran esas siete sombras. Esto debido a que cuando se están revisando los Stick-Ons o sus sombras, puedo pegar y despegar con sólo un *click* en la figura. Damos un *hint* visual de que elemento vamos a despegar o pegar poniéndole un borde rojo en el caso de los Stick-Ons o tiñendo la sombra de rojo en el caso de las sombras. Sin embargo, puede que una sombra quede cubierta por otra más grande y si la mayor quedó "más arriba" la menor no va a recibir el foco, con lo que no podremos pegar su Stick-On dándole *click* y tendremos que ir a la lista de Stick-Ons a pegarlo. Esto no resulta muy agradable por lo que se implementó un sistema en que si muevo la ruedita del mouse (o *shift*+*click* para los mouse sin ruedita) sobre un punto donde hay varias sombras o Stick-Ons, el foco va rotando por los elementos que allí confluyen. Con esto es fácil identificar sombras en caso de dudas aunque naturalmente en algún punto el esquema se volverá inutilizable como ya lo vimos y mantenemos la recomendación de no exagerar con la cantidad de sombras sobre el mismo punto al mismo tiempo.

Veamos las siete sombras que se sobreponen en el punto central de la figura con 23 sombras, para ello basta poner el mouse sobre el punto más oscuro y mover la ruedita del mouse, obteniendo algo como lo que sigue:

Ilustración 64. Identificando las siete sombras del punto central

Stick-Ons

La manera en que deseábamos satisfacer uno de los objetivos de esta memoria, proveer control local de versiones para los diagramas era usando la herramienta de Stick-Ons propuesta en [1]. En este caso se extenderían los Stick-Ons a dos dimensiones y soportaríamos los diagramas, redes de Petri en particular, que se creen en el sitio web desarrollado. Revisaremos algunos casos simples pero desarrollados en el sitio web creado.

Ilustración 65. Figura original

Partiendo de la figura original en la figura anterior se harán varios cambios sobre ella usando Stick-Ons. Pretendemos mostrar que la implementación funciona y dar una sensación del modo de uso sin necesariamente haber visto el sitio web en persona.

Ilustración 66. Stick-On pegado sobre figura original

En la Ilustración 66 se muestra la situación en que a alguien le pareció que los círculos centrales estaban mal, y decide proponer un cambio. Para ello indica que va a pegar un Stick-On y saca los elementos centrales y pone una flecha que une el primer círculo con el último. Para el sistema es totalmente indiferente si el usuario ha borrado todas las flechas y ha puesto una nueva de iguales características, o si ha dejado de borrar una flecha y esta ha sido ajustada hasta unir los dos círculos. Esto es porque si efectivamente el caso es que ha dejado una flecha para modificar, el sistema al pegar un Stick-On hace una copia de los elementos modificados, en este caso una hipotética flecha que el usuario no borró y es esta copia la que ajusta el usuario. Por eso da lo mismo si la borra y pega una nueva o si simplemente la mueve, ya que entonces el que hace la copia y el pegado es el sistema automáticamente, pero la situación final es la misma.

Ilustración 67. Stick-On despegado de figura original, proyecta sombra.

Al despegar el Stick-On recién pegado volvemos a la situación original. Esto se puede ver en la Ilustración 67.

En la Ilustración 68 partimos desde cero con la figura original (Ilustración 65) y vemos que se ha pegado un nuevo Stick-On de la misma forma que antes, pero esta vez se propone un cambio un poco más complejo en donde la primera flecha y círculo han sido reemplazados por una bifurcación hacia dos cuadrados.

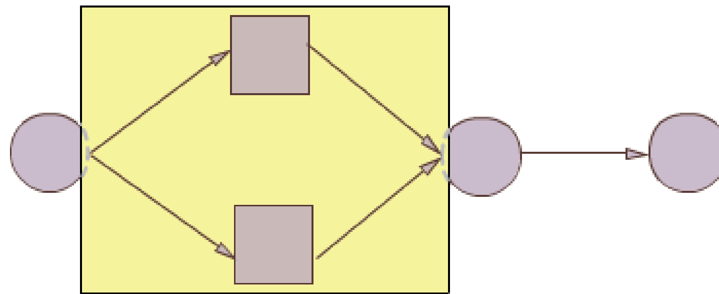


Ilustración 68. Stick-On más complejo

Al igual que antes, si se despega el Stick-On se ve una sombra y la figura original.

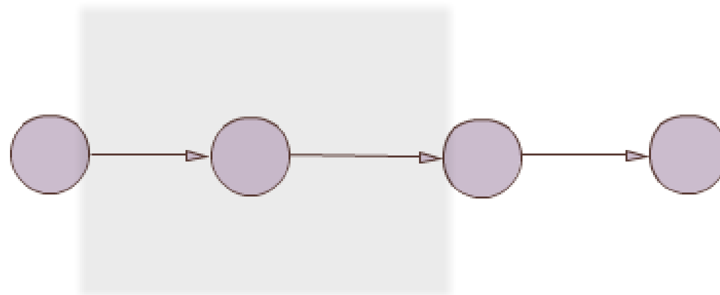


Ilustración 69. Stick-On despegado

Veamos ahora que ocurre cuando se proponen más cambios y tenemos solapamiento de Stick-Ons.

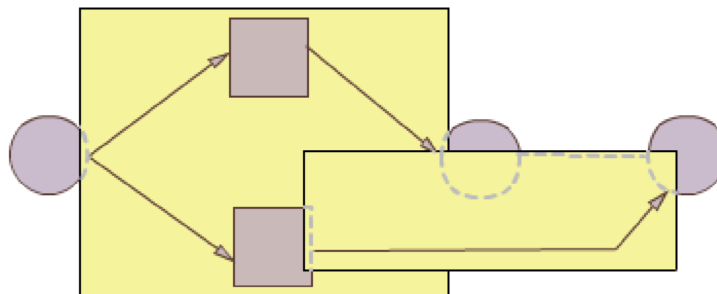


Ilustración 70. Se agrega un segundo Stick-On

En la Ilustración 70 se ha agregado un segundo Stick-On. Este nuevo cambio consiste simplemente en reemplazar la flecha que iba directamente al segundo círculo y se le hace apuntar al último círculo. Vemos que el segundo Stick-On queda sobre el primero. Esto es por definición, los Stick-Ons los vamos pegando sobre los ya existentes. Notamos además que se ha generado del mismo color. Esto también es por una decisión nuestra. Normalmente debería ser de otro color, ya sea para indicar cuál es más nuevo o quién pego el Stick-On. Sin embargo en este caso que estamos implementando, diagramas y redes de Petri esperamos que sea común tener justamente cuadrados de colores que se pueden confundir con los Stick-Ons por lo que

optamos por usar siempre el mismo color y darle al usuario la posibilidad de cambiar el color de un Stick-On en particular. El último detalle que quisiéramos comentar de la Ilustración 70 es lo ocurrido con el segundo círculo. Ha quedado parcialmente cubierto por el segundo Stick-On al igual que un cuarto del último círculo. Ninguno de estos dos círculos se ve afectado por pegar o despegar el segundo Stick-On ya que fue el resultado de mover la flecha solamente. Sin embargo, como lo comentamos en la sección de diseño, debido a que se genera un Stick-On rectangular elementos que no tienen relación directa con el Stick-On pueden quedar tapados, y como solución a este efecto que no deseamos propusimos dibujar una silueta de los elementos que han sido cubiertos "sin querer". Las líneas cortadas indican las siluetas de los objetos cubiertos o semi cubiertos.

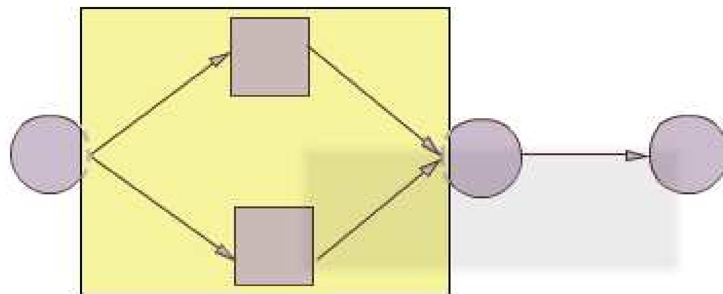


Ilustración 71. Segundo Stick-On despegado

Al despegar el segundo Stick-On tenemos la figura de la Ilustración 71. Vemos que ha vuelto a aparecer la flecha que había sido modificada (apuntando hacia el círculo) y se bosqueja la sombra dejada por el Stick-On. Las otras alternativas son posibles y las mostramos en la Ilustración 72, en donde tenemos el primer Stick-On despegado también y al segundo pegado.

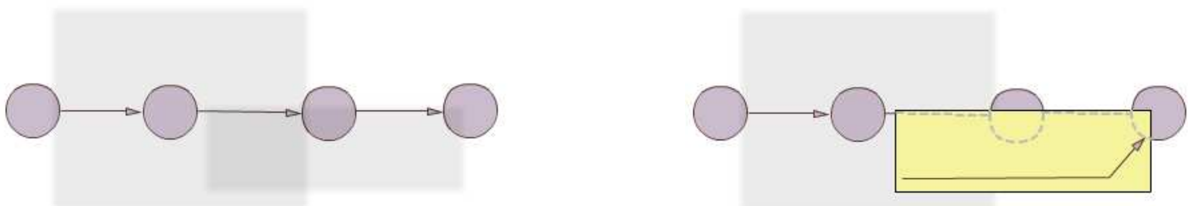


Ilustración 72. Dos Stick-Ons despegados y segundo Stick-On pegado

Agreguemos ahora un tercer Stick-On. Ya no deseamos que los dos últimos círculos estén conectados, y para ello agregamos un cuadrado (Ilustración 73), hacemos apuntar la flecha hacia él y agregamos una segunda flecha apuntando al cuadrado anterior.

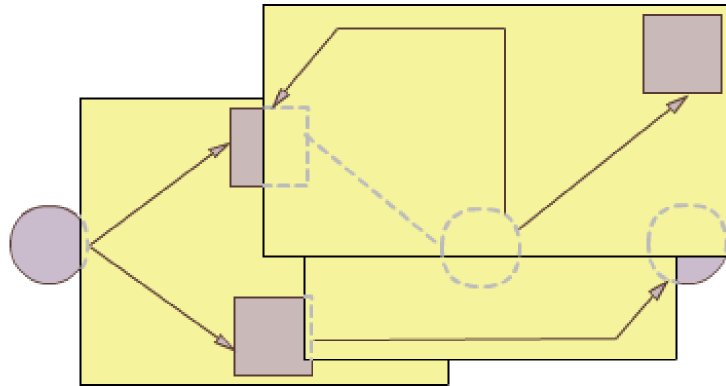


Ilustración 73. Tercer Stick-On

Vemos que la figura resultante no es confusa a pesar de tener 3 Stick-Ons solapándose, gracias a que las siluetas de los elementos activos siempre se proyectan, dando una visión del gráfico que actualmente se tiene. Si se despega o pega cualquiera de los Stick-Ons, las siluetas se ajustan de acuerdo a la nueva configuración de elementos.

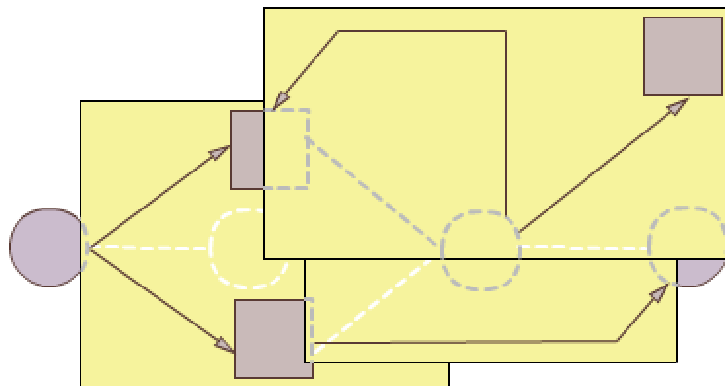


Ilustración 74. Dos tipos de siluetas, la silueta blanca solamente opcional

En la Ilustración 74 mostramos un segundo tipo de silueta, en color blanco. Esta es local a cada Stick-On y representa él o los elementos que este Stick-On ha reemplazado o modificado. Al usarlo la verdad es que no resultó ser tan buen aporte como pensamos y tiende a confundir, así que esta función la dejamos de activación opcional.

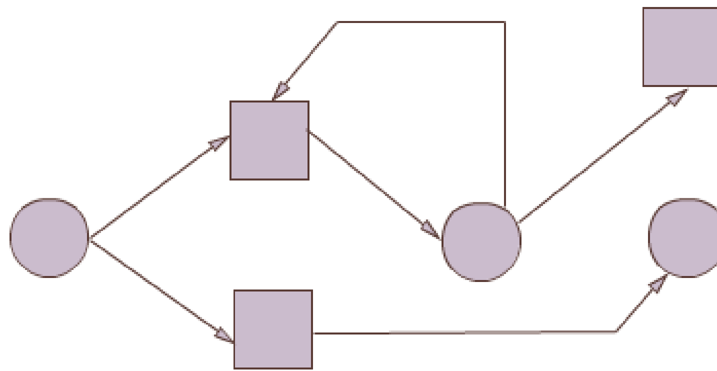


Ilustración 75. Resultado de limpiar los Stick-Ons

La Ilustración 75 muestra el resultado de “limpiar” los tres Stick-Ons que se han pegado, resultando en una versión final de estos cambios. Es interesante comparar este resultado con lo que estaba mostrando la Ilustración 73, que a pesar de tener varios Stick-Ons puestos daba una buena indicación de cómo sería el resultado de aceptar dichos cambios. Por supuesto que había más opciones, podríamos haber rechazado (despegado) los dos últimos Stick-Ons con lo que habríamos tenido la situación mostrada en la Ilustración 76.

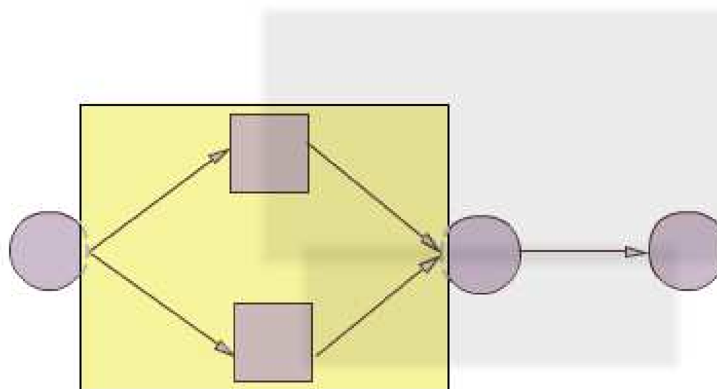


Ilustración 76. Dos Stick-Ons despegados

Al aceptar esta configuración resulta una nueva versión del diagrama que se puede ver en la Ilustración 77.

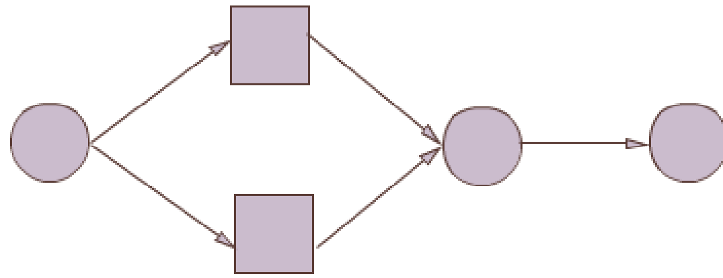


Ilustración 77. Versión alternativa

Finalmente, vemos que al despegar todos los Stick-Ons volvemos a la figura original del ejemplo

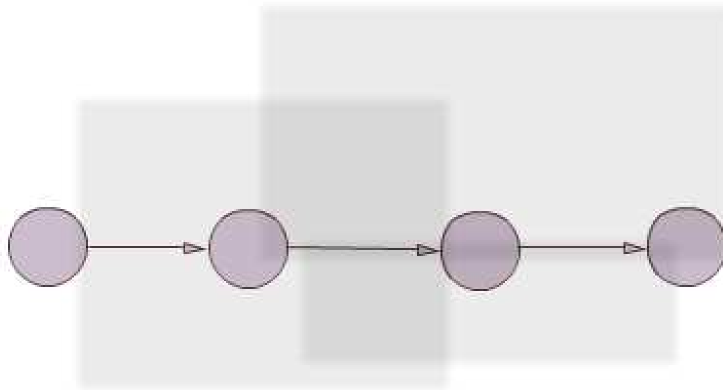


Ilustración 78. Sombras sobre figura original.

Conclusiones

Se ha implementado un sitio web que permite realizar redes de Petri en línea, grabarlos, almacenar distintas versiones, exportarlos a formato PNG y compartir documentos con otros usuarios. En la sección previa, “Resultados”, se han presentado algunas redes de Petri creadas con la aplicación. La plataforma sobre la cual se ha desarrollado es Flex (una variante de Flash enfocada a aplicaciones en Internet) junto a servicios escritos en PHP y una base de datos mySQL. Esto nos ha permitido contar con capacidades gráficas mucho más allá de lo que podríamos haber contado usando solamente HTML. Es necesario tener instalado el *plug-in* para flash, pero esto es muy común y realmente viendo el sistema en perspectiva, no había alternativas. Un desarrollo muy parecido al que realizamos, conservando las distancias por supuesto, sería intentar crear un editor de imágenes como *Photshop* en línea. Mi recomendación, en estos momentos y en base a las tecnologías que conozco, sería también usar Flash o Flex.

Dentro de los orígenes del proyecto se plantearon algunos desafíos especiales, por ejemplo tener distintos niveles de granularidad o tener versiones parciales de las redes de Petri. Para satisfacer estos requisitos se usaron elementos llamados “procesos” que nos permiten entrar a ver mayor detalle en una red de Petri, y para las versiones parciales o locales se usaron Stick-Ons.

Los “procesos” que comentábamos nos dejan poder encapsular un sub diagrama complejo en un solo elemento, el cual es dibujado como un cuadrado de doble borde. Eso está de acuerdo a la extensión jerárquica de las redes de Petri, pero creemos que su uso puede ser mucho más general que un sub proceso en una red de Petri. El concepto de entrar y salir en un elemento y poder ver nueva información realmente le agrega una nueva dimensión al diagrama. Un eje z, por decirlo de alguna manera, que transforma un diagrama simple en un árbol de diagramas. Dentro del sitio la única limitación en estos momentos para tener esta funcionalidad es que se debe usar un elemento de las redes de Petri (proceso).

Se implementaron dos tipos de control de versiones. Por un lado tenemos versiones de los diagramas como un todo generadas cada vez que se graba, y por otro proveemos la funcionalidad para un control local de versiones usando Stick-Ons en dos dimensiones. Este segundo control de versiones es bastante más complejo y se han tomado algunas decisiones que revisaremos ahora que tenemos una perspectiva más acabada del producto. La forma en que pegamos un Stick-On es indicando que se van a hacer cambios, y luego en base a los elementos modificados se dibuja un Stick-On. Creo que esto, o alguna alternativa parecida, es lo correcto dado que estamos trabajando con objetos vectoriales y la información relevante son los objetos y no un sector del espacio. Sin embargo, me parece que si se desean aplicar Stick-Ons a otro tipo de contenidos en dos dimensiones que no sean gráficos vectoriales, se debe volver a analizar. Por ejemplo, si se trabaja con mapas de bits me parecería adecuada la forma tradicional de pegar los Stick-Ons (definiendo una superficie explícitamente).

Al representar la sombra que deja un Stick-On con una simulación de una sombra real pudimos tener una idea intuitiva y sencilla de cómo eran los Stick-Ons que habían sido despegados. Esto en la medida de que no se tuvieran demasiadas sombras al mismo tiempo en el mismo sector.

Nuestra recomendación fue ojalá no pasar de cinco Stick-Ons despegados al mismo tiempo para no perder la claridad, aunque depende mucho del caso. Este enfoque para manejar las sombras tiene la ventaja de que a mi modo de ver es la implementación más natural posible, pero tiene en contra que no es demasiado escalable. Dentro de los rangos normales creo que puede ser una buena opción.

El uso de Stick-Ons permitió poder proponer diferentes cambios dentro de diagramas en una forma satisfactoria. La Ilustración 73 y la Ilustración 75 dan cuenta de una situación en que hay varios Stick-Ons con cambios propuestos en un mismo instante, y se puede saber por simple inspección como es el resultado de la configuración actual de Stick-Ons. Aunque el ejemplo consta de sólo tres Stick-Ons, no es difícil ir agregando más cambios y tener una buena idea de lo que tenemos en pantalla. Después de todo, estoy viendo solamente un diagrama "parchado" pero nunca dejo de tener una noción de la situación actual. En algún momento tuvimos Stick-Ons transparentes, pero resultaban confusos cuando había más de uno. Al volver a los orígenes y tener los Stick-Ons opacos se presentaba el problema de que algunos elementos quedaban tapados y perdíamos la visión global del diagrama. Para solucionar esto se propusieron las *siluetas*, que son las formas de los elementos tapados proyectados sobre la superficie del Stick-On en forma de línea punteada. Da lo mismo si pertenece a otro Stick-On que está tapado o si esta bajo varios Stick-Ons, pero si un elemento está o debe estar visible, proyecta silueta. Esto permite que por más compleja que sea la disposición de los Stick-On podamos tener una vista del diagrama siempre.

Posibles mejoras

¿Por dónde empezar? Partiré diciendo que subestime absolutamente lo que significa hacer una herramienta de diagramación. El enfocarse en redes de Petri es una ventaja porque nos acota las posibilidades de elementos gráficos que tendremos que crear, y a su vez es la primera cosa que se piensa en mejorar: incluir en los diagramas otros tipos como modelos de datos, organigramas, circuitos, etc. La segunda mejora importante debe ser incluir alguna funcionalidad básica a los diagramas que quedo fuera por motivos de tiempo. Por ejemplo:

- *Seleccionar grupos*
- *Copiar estilo*
- *Agrupar & Desagrupar elementos*
- *Traer al frente & enviar al fondo*
- *Poder cambiar el color de fondo (blanco)*
- *Especificar un tamaño de hoja para el área de trabajo*
- *Rotar elementos*
- *Alinear elementos*

Esas como funciones mínimas. De entre ellas pareciera ser que la mayoría son sencillas de implementar, pero hay algunas que me da la impresión de que pueden ser más difíciles, por ejemplo *Agrupar & Desagrupar* y *Rotar elementos*.

Dentro de los cambios o mejoras en aspectos más avanzados, se debe replantear el esquema de "*docking*" cuando un conector se une a una figura. Al diagramar redes de Petri es común por lo que pude ver tener dos conectores paralelos por ejemplo, que llegan a otra figura en forma paralela también. Nuestro *docking,* que atrae las puntas de los conectores a ciertos puntos de las figuras, hace casi imposible este esquema, a tal punto que debí desactivarlo para diagramar las redes de Petri. Como alternativa pensaba en que si bien ayuda que los conectores queden unidos a las figuras, el problema es que se unen a puntos específicos, en nuestro caso ubicados en los puntos cardinales solamente. Creo que deberían quedar ligados de manera más general al borde de la figura.

Otro aspecto que sería fantástico poder implementar es tener la capacidad de exportar al formato "*Open Document*", de tal modo de poder llevar nuestros diagramas en forma de objetos a otras herramientas que soporten este nuevo formato y ya no solamente como imágenes.

Dentro de aspectos relativos al uso del sitio y de los diagramas, sería muy interesante explorar las alternativas de colaboración síncrona. En estos momentos se bloquea el diagrama y solamente un usuario lo puede modificar. Eso no tiene que ser necesariamente así, de hecho creo que es posible que dos personas modificando un mismo diagrama vean en tiempo real los cambios hechos por el otro, con todos los nuevos problemas y definiciones que esto conlleva.

8. Referencias

[1] J.A.Pino, *A visual approach to versioning for text co-authoring*. Interacting with Computers, 8, 1996, pp. 299-310

[2] Pineda, E., Pino, J. A., *Stick-Ons revisited*, 7th International Workshop on Groupware (CRIWG'2001), Sept. 6-8, 2001, Darmstadt, Germany; pp. 26-35.

[3] Petri Nets - Kevin Mcleish
http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C10/10-7.html

[4] Petri, Carl A. (1962). "*Kommunikation mit Automaten*". Ph. D. Thesis. University of Bonn.

[5] Desel, Jörg and Juhás, Gabriel "*What Is a Petri Net? -- Informal Answers for the Informed Reader*", Hartmut Ehrig *et al.* (Eds.): Unifying Petri Nets, LNCS 2128, pp. 1-25, 2001.

[6] Redes de Petri
http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/

[7] Historia de Flash
http://www.devarticles.com/c/a/Flash/Flash-Looking-Back-Looking-Forward/

[8] Chan SCF, Ng VTY, *Real-Time collaborative solid shape design (RCSSD) on the internet*. Concurrent Engineering 10(3), 2002, 229-238.

[9] Chan SCF, Lee PHS, NG VTY, et al., Synchronous collaborative development of UML models on the internet. Concurrent Engineering 9(2), 2001, 111-119.

[10] Descipción de applets de Java.
http://en.wikipedia.org/wiki/Java_applet

[11] Encuesta de Millward Brown, realizada en Marzo 2007.
http://www.adobe.com/products/player_census/flashplayer/

[12] Van der Aalst, W y Van Hee,K.: *Workflow Management – Models, Methods and Systems*. The MIT Press, Cambridge, MA, 2004.

[13] Descripción del lenguaje VML
http://en.wikipedia.org/wiki/Vector_Markup_Language

[14] Descripción del lenguaje SVG
http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[15] Explorer Canvas
http://excanvas.sourceforge.net/

[16] Especificación HTML5
http://www.whatwg.org/specs/web-apps/current-work/multipage/section-the-canvas.html

[17] Librería basada en DIVs de Walter Zone
http://www.walterzorn.com/jsgraphics/jsgraphics_e.htm

[18] Calendario DTHML.
http://www.dynarch.com/projects/calendar

[19] Explorador de primitivas Flex.
http://flexibleexperiments.wordpress.com/2007/03/14/flex-20-primitive-explorer/

[20] Tienda Flex.
http://examples.adobe.com/flex2/inproduct/sdk/flexstore/flexstore.html

[21] Silk Icons
http://www.famfamfam.com/lab/icons/silk/

[22] 101 Damnations.
http://www.softwarereality.com/programming/ejb/index.jsp

[23] EJB query language
http://www.c2.com/cgi/wiki?EjbQueryLanguage

[24] Definición EJB
http://en.wikipedia.org/wiki/Enterprise_JavaBean#Reinventing_EJBs

[25] Críticas a Ruby
http://beust.com/weblog/archives/000382.html

[25] Defínción de webservice
http://en.wikipedia.org/wiki/Webservice

[26] Definición de PHP
http://en.wikipedia.org/wiki/Php