Manual del usuario mxGraph - JavaScript Client

Tabla de contenido

1 Introducción

1.1 Introducción del producto

mxGraph es un componente de JavaScript que proporciona funciones dirigidas a aplicaciones que muestran diagramas y gráficos interactivos . Nota por gráficos nos referimos a gráficos matemáticos , no necesariamente a gráficos (aunque algunos gráficos son gráficos). Vea la sección más adelante "¿Qué es un gráfico?" Para más detalles.

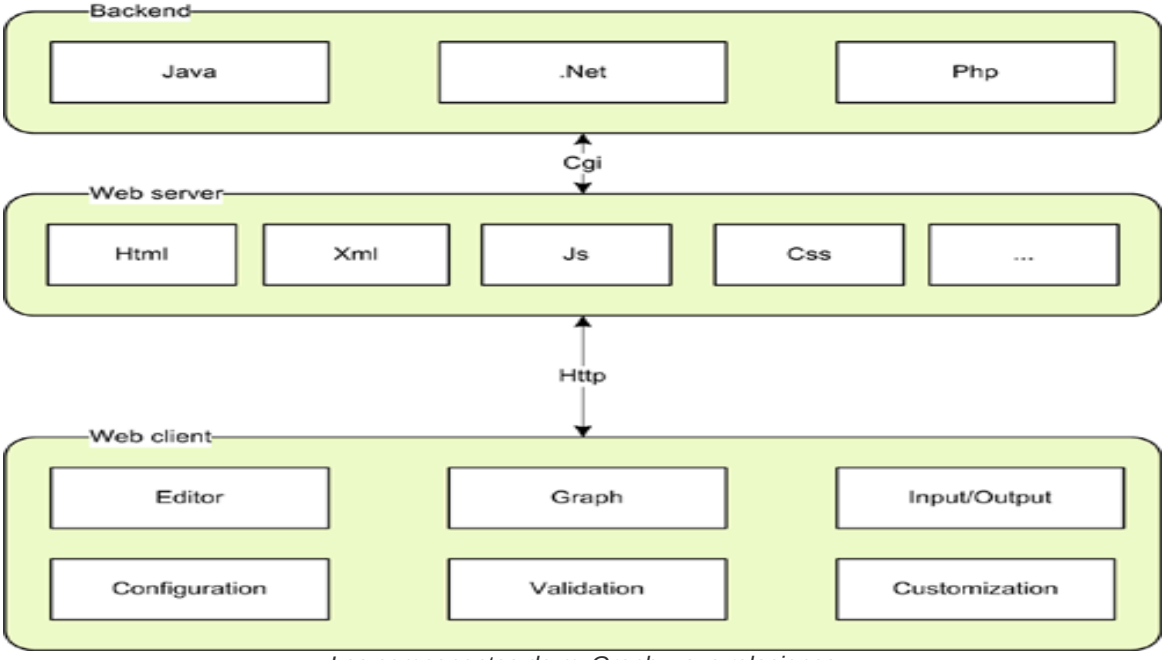
Siendo una biblioteca de desarrolladores, mxGraph no está diseñado específicamente para proporcionar una aplicación lista para usar, aunque muchos de los ejemplos están cerca de ser aplicaciones utilizables. mxGraph proporciona todas las funcionalidades comúnmente requeridas para dibujar, interactuar y asociar un contexto con un diagrama. mxGraph viene con una serie de ejemplos que ayudan a explicar cómo una aplicación básica se reúne y muestra las características individuales de la biblioteca.

Los desarrolladores que integran la biblioteca en su aplicación deben leer la sección "Prerrequisitos" a continuación. Dado que mxGraph es una parte integrante de su aplicación, debe comprender cómo se construyen las aplicaciones web JavaScript a nivel arquitectónico, y cómo programarlas tanto en JavaScript como en cualquier otro idioma utilizado en el servidor.

mxGraph comprende principalmente un archivo JavaScript que contiene todas las funcionalidades de mxGraph. Esto se carga en una página web HTML en una sección de JavaScript y se ejecuta en un contenedor HTML en el navegador. Esta es una arquitectura increíblemente simple que solo requiere un servidor web capaz de servir páginas html y un navegador web habilitado con JavaScript.

Las principales ventajas de esta tecnología son:

- Que no se requieren complementos de terceros. Esto elimina la dependencia del proveedor del complemento.
- Las tecnologías involucradas son abiertas y hay muchas implementaciones abiertas, ningún proveedor puede eliminar un producto o tecnología que deje su aplicación inviable en la práctica.
- Tecnologías estandarizadas, lo que significa que su aplicación se puede implementar en la mayor cantidad posible de usuarios de navegadores sin necesidad de configuraciones o instalaciones adicionales en la computadora del cliente. A los entornos corporativos grandes a menudo no les gusta permitir que las personas instalen complementos del navegador y no les gusta cambiar la versión estándar implementada en todas las máquinas.



Los componentes de mxGraph y sus relaciones

1.2 ¿Para qué aplicaciones se puede usar mxGraph?

Las aplicaciones de ejemplo para una biblioteca de visualización de gráficos incluyen: diagramas de proceso, flujo de trabajo y visualización BPM, diagramas de flujo, tráfico o flujo de agua, visualización de base de datos y WWW, redes y pantallas de telecomunicaciones, aplicaciones de mapeo y SIG, diagramas UML, circuitos electrónicos, VLSI, CAD, financiero y redes sociales, minería de datos, bioquímica, ciclos ecológicos, relaciones entre entidades y causa-efecto y organigramas.

1.3 ¿Cómo se implementa mxGraph?

En el entorno típico de thin-client, mxGraph se divide en la biblioteca de JavaScript del lado del cliente y una biblioteca del lado del servidor en uno de los dos lenguajes soportados, .NET y Java. La biblioteca de JavaScript se incluye como parte de una aplicación web más grande que se entrega al navegador utilizando un servidor web estándar. Todas las necesidades del navegador es la capacidad de ejecutar JavaScript para que esté habilitado.

En la tercera parte de este manual, verá un ejemplo de una página html que incorpora la biblioteca mxGraph, así como una aplicación simple para invocar la funcionalidad de la biblioteca.

1.4 mxGraph Technologies

mxGraph usa JavaScript para la funcionalidad del lado del cliente en el navegador. El código JavaScript a su vez utiliza el lenguaje de gráficos vectoriales subyacente en el navegador activo para representar el diagrama mostrado, actualmente SVG para todos los navegadores compatibles. mxGraph también incluye la característica de renderizar completamente usando html, esto limita el rango de funcionalidad disponible, pero es adecuado para diagramas más simples.

Como desarrollador, no está expuesto a funciones específicas del navegador. Como se mencionó, el lenguaje de gráficos vectoriales varía según el navegador, por lo que mxGraph abstrae sus características en una clase común. Del mismo modo, para el manejo de

eventos y DOMs. Los navegadores difieren en la implementación de estas dos funcionalidades principales del navegador, mxGraph expone una API constante en todos los navegadores y se adapta a las incoherencias entre bastidores.

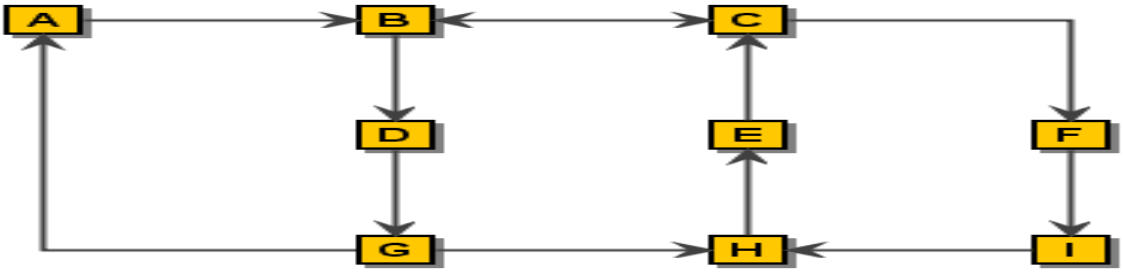
1.5 **mxGraph Licensing**

El cliente JavaScript de mxGraph está licenciado bajo la licencia Apache 2.0. Para preguntas detalladas sobre la licencia, siempre se recomienda consultar a un profesional del derecho.

1.6 **¿Qué es un gráfico?**

La visualización de gráficos se basa en la teoría matemática de redes, teoría de grafos. Si usted está buscando la barra de JavaScript *gráficos* , pastel de *gráficos* , Gantt *gráficos* , echar un vistazo a la Gráficos Google proyecto en lugar, o similar,

Un gráfico consiste en vértices, también llamados nodos, y de bordes (las líneas de conexión entre los nodos). Exactamente cómo un gráfico aparece visualmente no está definido en la teoría de grafos. El término *celda* se usará a lo largo de este manual para describir un elemento de un gráfico, ya sea bordes, vértices o grupos.



Un simple Gráfico

Existen definiciones adicionales en la teoría de grafos que proporcionan antecedentes útiles cuando se trata de gráficos, que se enumeran en los Apéndices si lo que le interesa.

1.6.1 **Visualización de gráficos**

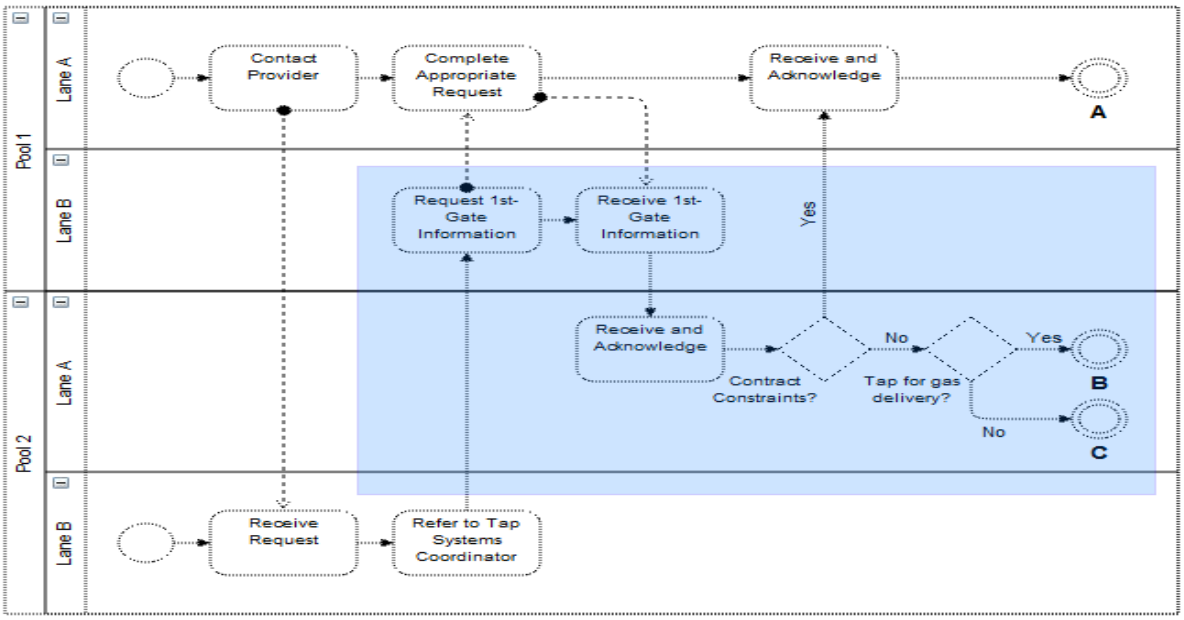
La visualización es el proceso de crear una representación visual útil de un gráfico. El alcance de la funcionalidad de visualización es una de las principales fortalezas de mxGraphs. mxGraph admite una amplia gama de características para permitir que la visualización de las celdas esté limitada únicamente por la habilidad del desarrollador y la funcionalidad de la plataforma disponible. Los vértices pueden ser formas, imágenes, dibujos vectoriales, animaciones, prácticamente cualquier operación gráfica disponible en los navegadores. También puede usar el marcado HTML en ambos vértices y bordes.

Visualización gráfica de un sistema de transporte. (c) Tourizm Maps 2003, http://www.world-maps.co.uk

1.6.2 **Interacción gráfica**

La interacción es la forma en que una aplicación que usa mxGraph puede alterar el modelo de gráfico a través de la GUI de la aplicación web. mxGraph admite arrastrar y clonar celdas, cambiar el tamaño y modificar la forma, conectar y desconectar, arrastrar y soltar desde fuentes externas, editar etiquetas de celdas en el lugar y más. Uno de los beneficios clave de mxGraph es la flexibilidad de cómo se puede programar la interacción.

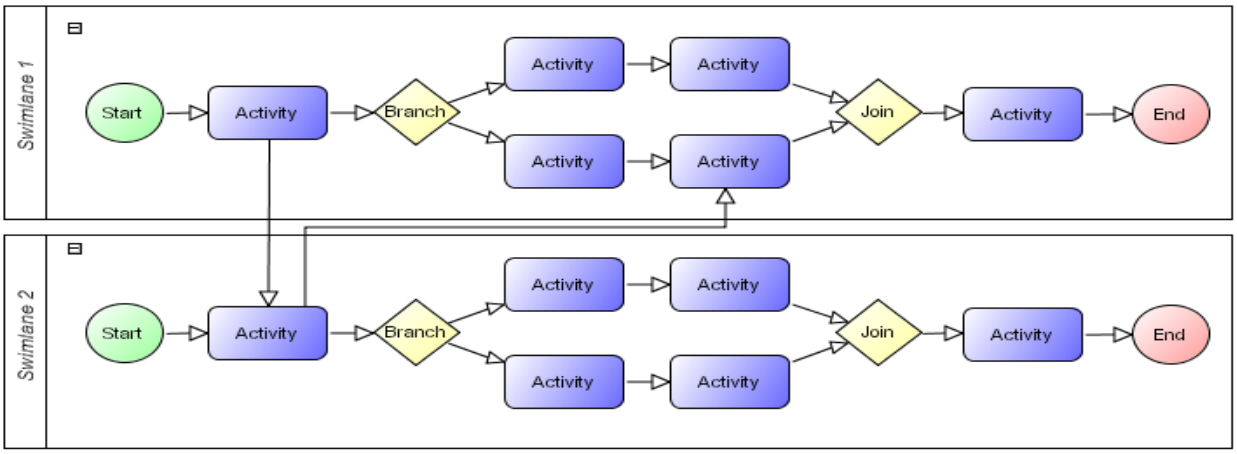
Muchas aplicaciones web gráficas complejas se basan en un viaje de ida y vuelta al servidor para formar la pantalla, no solo la pantalla base, sino también los eventos de interacción. Aunque a esto a menudo se le da el título de funcionalidad AJAX, dicha dependencia del servidor no es apropiada para eventos de interacción. La retroalimentación visual que toma más de 0.2 segundos en una aplicación generalmente afecta seriamente la usabilidad. Al colocar toda la interacción en el cliente, mxGraph proporciona la sensación real de una aplicación, en lugar de parecer un terminal remoto tonto. También permite la posibilidad de uso fuera de línea.



Sombreado de selección al seleccionar un área con la función de arrastrar del mouse

1.6.3 **Diseños de gráficos**

Las celdas de gráfico se pueden dibujar en cualquier lugar en una aplicación simple, incluso una encima de la otra. Ciertas aplicaciones necesitan presentar su información en una estructura generalmente ordenada u ordenada específicamente. Esto podría implicar que las celdas no se superpongan y se mantengan al menos una cierta distancia entre sí, o que las celdas aparezcan en posiciones específicas relativas a otras celdas, generalmente las celdas a las que están conectadas por los bordes. Esta actividad, denominada aplicación de diseño, se puede utilizar de varias maneras para ayudar a los usuarios a configurar su gráfico. Para gráficos no editables, la aplicación de diseño es el proceso de aplicar un algoritmo de diseño a las celdas. Para los gráficos interactivos, es decir, aquellos que se pueden editar a través de la interfaz de usuario, la aplicación de diseño podría implicar solo permitir a los usuarios realizar cambios en determinadas celdas en determinadas posiciones,



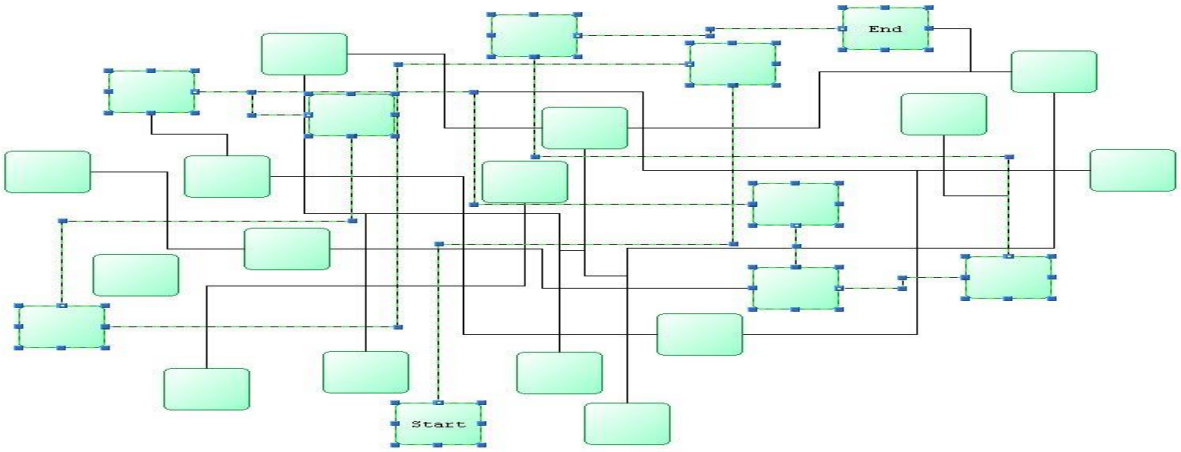
Diseño de un flujo de trabajo usando un diseño jerárquico horizontal

mxGraph admite una gama de diseños jerárquicos, dirigidos por fuerza y de árbol que se adaptarán a la mayoría de las necesidades de diseño. Consulte la sección posterior sobre cómo usar los diseños para obtener más información.

En una arquitectura cliente-servidor, existen dos opciones para ejecutar diseños. Las versiones de Javascript ofrecen la capacidad de ejecutar el diseño por completo en el cliente, mientras que la misma implementación de diseño en Java en el lado del servidor permite la opción de descargar algún procesamiento al servidor, si es necesario.

1.6.4 **Análisis de gráficos**

El análisis de gráficos implica la aplicación de algoritmos que determinan ciertos detalles sobre la estructura del gráfico, por ejemplo, determinando todas las rutas o la ruta más corta entre dos celdas. Existen algoritmos de análisis de gráficos más complejos, que a menudo se aplican en tareas específicas de dominio. Las técnicas como la agrupación en clúster, la descomposición y la optimización tienden a dirigirse a ciertos campos de la ciencia y no se han implementado en los paquetes core mxGraph en el momento actual de la escritura.



Análisis de ruta más corta

1.7 Acerca de este manual

1.7.1 Requisitos previos para mxGraph

Para beneficiarse completamente de este manual, deberá tener una comprensión razonable de las aplicaciones web y de la tecnología de servidor con la que desea implementar. Los ejemplos de implementación están disponibles para cada una de las tecnologías de servidor admitidas, obviamente se requiere cierta familiaridad con esa tecnología de servidor.

El conocimiento básico de XML es útil para cambiar los archivos de configuración del editor que describen los aspectos visuales y de comportamiento del editor. Deberá comprender e implementar la codificación Javascript y estar familiarizado con los principios de programación orientada a objetos y el diseño de software moderno.

No necesita conocer el lenguaje de gráficos vectoriales subyacente que utiliza el navegador, como SVG o lienzo HTML. mxGraph abstrae la descripción del componente visual en una API.

2 Comenzando

2.1 El paquete mxGraph

2.1.1 Obteniendo mxGraph

mxGraph está disponible en el proyecto GitHub . Las versiones lanzadas están etiquetadas como "va.bc", donde a, byc son partes del número de versión que siguen a la versión semántica .

Cada versión formal también está disponible como .zip o .tar.gz en la página de lanzamientos de mxGraph .

2.1.2 Estructura del proyecto y opciones de construcción

Una vez descomprimido, se le presentará una cantidad de archivos y directorios en la raíz de la instalación.

/doc	La raíz de la documentación, incluye este manual del usuario
/punto net	Clases del lado del servidor .NET
/Java	Clases del lado del servidor de Java
/ javascript	Funcionalidad del cliente de JavaScript.
/ javascript / examples	Ejemplos de HTML que demuestran el uso de mxGraph
ChangeLog	Detalles de los cambios entre lanzamientos
index.html	Introducción básica a la biblioteca
licencia.txt	Los términos de licencia bajo los cuales debe usar la biblioteca

Tabla: estructura del directorio del proyecto

2.1.3 npm

mxGraph también está disponible a través del administrador de paquetes npm. Para usar mxGraph como una dependencia, use `npm install`:

```
npm install mxgraph --save
```

El módulo se puede cargar usando `require()`. Esto devuelve una función de fábrica que acepta un objeto de opciones. Las opciones como, por ejemplo, se `mxBasePath`deben proporcionar a la función de fábrica, en lugar de especificarse como variables globales.

```
var mxgraph = require ("mxgraph") ({
  mxImageBasePath: "./src/images",
  mxBasePath: "./src"
})
```

La función de fábrica devuelve un 'objeto de espacio de nombres' que proporciona acceso a todos los objetos del paquete mxGraph. Por ejemplo, el `mxEvent` objeto está disponible en `mxgraph.mxEvent`.

```
var mxEvent = mxgraph.mxEvent;
mxEvent.disableContextMenu (contenedor);
```

2.2 **JavaScript y aplicaciones web**

Las aplicaciones web, específicamente el uso de JavaScript para intentar emular el comportamiento similar a las aplicaciones de escritorio en los navegadores web, sigue siendo un campo relativamente nuevo de ingeniería de software. Hay tres problemas principales con JavaScript que se perciben como una barrera para producir aplicaciones de alta calidad, rendimiento, falta de funcionalidad nativa disponible en aplicaciones de escritorio y API incoherentes entre navegadores.

Ha habido un esfuerzo considerable para desarrollar bibliotecas de frameworks para resolver dos de los problemas, la funcionalidad y los problemas de API. Los requisitos de muchas de estas bibliotecas se basan en la mejora del diseño y la usabilidad del sitio web, así como en la producción de lo que generalmente llamamos funciones de la aplicación (menús, ventanas, diálogos, persistencia, gestión de eventos, etc.). También proporcionan ciertas funcionalidades básicas que faltan en JavaScript que los desarrolladores de aplicaciones de escritorio dan por hecho, como la funcionalidad básica de las matemáticas y las colecciones.

Muchos de estos frameworks de JavaScript tienen soporte IDE para el desarrollo hoy en día y todos los principales navegadores ahora contienen depuradores de JavaScript, ya sea de forma nativa o como un complemento. No existe una fase de compilación con JavaScript (es un lenguaje interpretado), por lo que los errores tipográficos básicos a menudo solo se detectan en tiempo de ejecución, a menos que obtenga una herramienta de comprobación de sintaxis en su IDE. Entonces, aunque no hay un paquete completo para sus necesidades de desarrollo de JavaScript, hay varios proveedores que proporcionan los componentes individuales que necesita para producir aplicaciones de JavaScript de manera efectiva.

2.2.1 **Frameworks de JavaScript de terceros**

2.2.1.1 **Frameworks y bibliotecas nativos de JavaScript**

En lugar de enumerar y comparar cada marco de JavaScript, consulte las entradas de wikipedia para los marcos de aplicaciones web y la comparación de JavaScript . La comparación no debe considerarse autoritativa, más aún, ilustra los tipos de características proporcionadas, como manejo de eventos, animación, widgets, soporte de solicitud de AJAX, etc. Este sitio también es una lista útil de bibliotecas de JavaScript, la mayoría de las cuales son de código abierto / gratuito. .

Tenga en cuenta que muchos frameworks agregan comportamientos implícitos para hacer que JavaScript se parezca más a un lenguaje OO y para aumentar la funcionalidad básica del idioma. Durante la redacción de la parte de diseño de mxGraph, se encontró que este comportamiento implícito rompía un ejemplo de una manera muy difícil de depurar. Tenga en cuenta que esto puede causar problemas y, si selecciona un marco, asegúrese de comprender qué comportamientos implícitos introduce.

Al seleccionar un marco y / o bibliotecas, piense en qué marcos lo vinculan con determinado comportamiento funcional y busque bibliotecas que proporcionen características como la animación como bloques distintos e independientes, que puede usar sin estar vinculado al diseño general.

2.2.1.2 **Integración de los marcos mxGraph y JavaScript**

Esta área a menudo se entiende mal, en pocas palabras, no se requiere *integración* . Las aplicaciones web generalmente comprenden uno o más *div* elementos en los que se coloca el HTML que ajusta el JavaScript de la aplicación. Si crea un div como contenedor para un mxGraph, esa área es una pantalla independiente para la aplicación mxGraph. Se puede comunicar con cualquier servidor de fondo, pero no existe interdependencia entre ese div y el resto de la página, aparte del área que ocupa cada uno. Esto incluye el manejo de eventos, mxGraph puede manejar los eventos para su contenedor, incluso si el resto de la página web usa un modelo de evento completamente diferente. Siempre que ni mxGraph ni las otras bibliotecas y marcos en la página presenten comportamientos implícitos que rompan una parte de la página, el tema de la integración del cliente no es algo que necesite análisis.

La integración de la funcionalidad de fondo mxGraph, que se encuentra en el lado del servidor es el tema de un capítulo posterior.

2.2.1.3 **Extender mxGraph en JavaScript**

En JavaScript, hay varias formas de mapear el paradigma orientado a objetos a construcciones de lenguaje. mxGraph utiliza un esquema particular en todo el proyecto, con las siguientes reglas implícitas:

- No cambie los prototipos incorporados
- No intente limitar el poder del lenguaje de JavaScript.

Hay dos tipos de "clases" en mxGraph; *clases* y *singletons* (donde solo existe una instancia de la clase). Los singletons se asignan a objetos globales donde el nombre de la variable es el mismo que el nombre de la clase. Por ejemplo, mxConstants es un objeto con todas las constantes definidas como campos de objeto. Las clases normales se asignan a una función constructora y un prototipo que define los campos y métodos de la instancia. Por ejemplo, mxEditor es una función y mxEditor.prototype es el prototipo del objeto que crea la función mxEditor. El prefijo *mx* es una convención que se usa para todas las clases en el paquete mxGraph para evitar conflictos con otros objetos en el espacio de nombres global.

Para la creación de subclases, la superclase debe proporcionar un constructor que sea sin parámetros o maneje una invocación sin argumentos. Además, el campo constructor especial debe redefinirse después de extender el prototipo. Por ejemplo, la superclase de mxEditor es mxEventSource. Esto se representa en JavaScript al primero "heredar" todos los campos y métodos de la superclase asignando el prototipo a una instancia de la superclase, por ejemplo.

```
mxEditor.prototype = new mxEventSource ()
```

y redefiniendo el campo constructor usando:

```
mxEditor.prototype.constructor = mxEditor
```

La última regla se aplica para que el tipo de un objeto se pueda recuperar a través del nombre de su constructor usando *mxUtils.getFunctionName (obj.constructor)* .

Constructor

Para la subclasificación en mxGraph, se debe aplicar el mismo mecanismo. Por ejemplo, para subclasificar la clase mxGraph, primero se debe definir un constructor para la nueva clase. El constructor llama al super constructor con cualquier argumento que pueda tener utilizando la función de *llamada* en el objeto de función mxGraph, pasando explícitamente cada argumento:

```
función MyGraph (contenedor)
{
   mxGraph.call (esto, contenedor);
}
```

El prototipo de MyGraph hereda de mxGraph de la siguiente manera. Como de costumbre, el constructor se redefine después de extender la superclase:

```
MyGraph.prototype = new mxGraph ();
MyGraph.prototype.constructor = MyGraph;
```

Es posible que desee definir el códec asociado para la clase después del código anterior (consulte la sección de E / S del manual). Este código se ejecutará en el momento de carga de clases y se asegura de que se use el mismo códec para codificar las instancias de mxGraph y MyGraph.

```
var codec = mxCodecRegistry.getCodec (mxGraph);
codec.template = new MyGraph ();
mxCodecRegistry.register (codec);
```

Funciones

En el prototipo de MyGraph, las funciones de mxGraph se pueden ampliar de la siguiente manera.

```
MyGraph.prototype.isSelectable = function (cell)
{
   var seleccionable = mxGraph.prototype.isSelectable.apply (esto, argumentos);
   var geo = this.model.getGeometry (celda);
   return seleccionable && (geo == null ||! geo.relativo);
}
```

El supercall en la primera línea es opcional. Se lleva a cabo mediante el *aplicar* la función de la *isSelectable* función del objeto del prototipo mxGraph, usando las especiales *esta* ANDN *argumentos* de variables como parámetros. Las llamadas a la función de superclase solo son posibles si la función no se reemplaza en la superclase de la siguiente manera, que es otra forma de "subclasificación" en JavaScript.

```
mxGraph.prototype.isSelectable = function (cell)
{
   var geo = this.model.getGeometry (celda);
   return seleccionable && (geo == null ||! geo.relativo);
}
```

El esquema anterior es útil si una definición de función necesita ser reemplazada por completo.

Para agregar nuevas funciones y campos a la subclase, se usa el siguiente código. El siguiente ejemplo agrega una nueva función para devolver la representación XML del modelo de gráfico:

```
MyGraph.prototype.getXml = function ()
{
   var enc = new mxCodec ();
   devuelve enc.encode (this.getModel ());
}
```

Campos

Del mismo modo, un nuevo campo se declara y define de la siguiente manera:

```
MyGraph.prototype.myField = '¡Hola, mundo!';
```

Tenga en cuenta que el valor asignado a myField se crea solo una vez, es decir, todas las instancias de MyGraph comparten el mismo valor. Si necesita valores específicos de instancia, entonces el campo debe definirse en el constructor. Por ejemplo:

```
función MyGraph (contenedor)
{
   mxGraph.call (esto, contenedor);
   this.myField = [];
}
```

Finalmente, se crea una nueva instancia de MyGraph utilizando el siguiente código, donde container es un nodo DOM que actúa como un contenedor para la vista gráfica:

```
var graph = new MyGraph (contenedor);
```

2.2.2 **Desarrollo general de JavaScript**

2.2.2.1 Ofuscación de **JavaScript**

De forma predeterminada, cuando entrega JavaScript a un cliente de navegador, entrega la fuente completa a ese JavaScript. Entonces, el JavaScript se interpreta y se ejecuta en el navegador. No es posible cifrar el JavaScript en ningún punto del cliente en el punto en que se ejecuta, ya que el intérprete de JavaScript debe ser entendido por el intérprete de JavaScript y los idiomas interpretados no tienen una forma intermedia binaria.

Sería posible encriptar el JavaScript en la transmisión y descifrarlo y ejecutarlo en el cliente, pero el cliente aún podría acceder a la fuente después del descifrado.

No nos ofuscamos porque los nombres de los métodos forman una API pública y la E / S debería comprender la ofuscación en ambos extremos de la comunicación.

2.2.2.2 **Espacios de nombres**

El concepto de espacios de nombres no existe en el JavaScript en pie, así que tenga mucho cuidado al crear nuevos nombres de clase. En mxGraph, todas las clases comienzan con el prefijo "mx-", para evitar conflictos o anular prototipos involuntariamente.

2.3 **¡Hola mundo!**

Hello World en mxGraph consiste en un simple ejemplo del lado del cliente que muestra dos vértices conectados con las etiquetas "Hello" y "World!". El ejemplo demuestra lo siguiente:

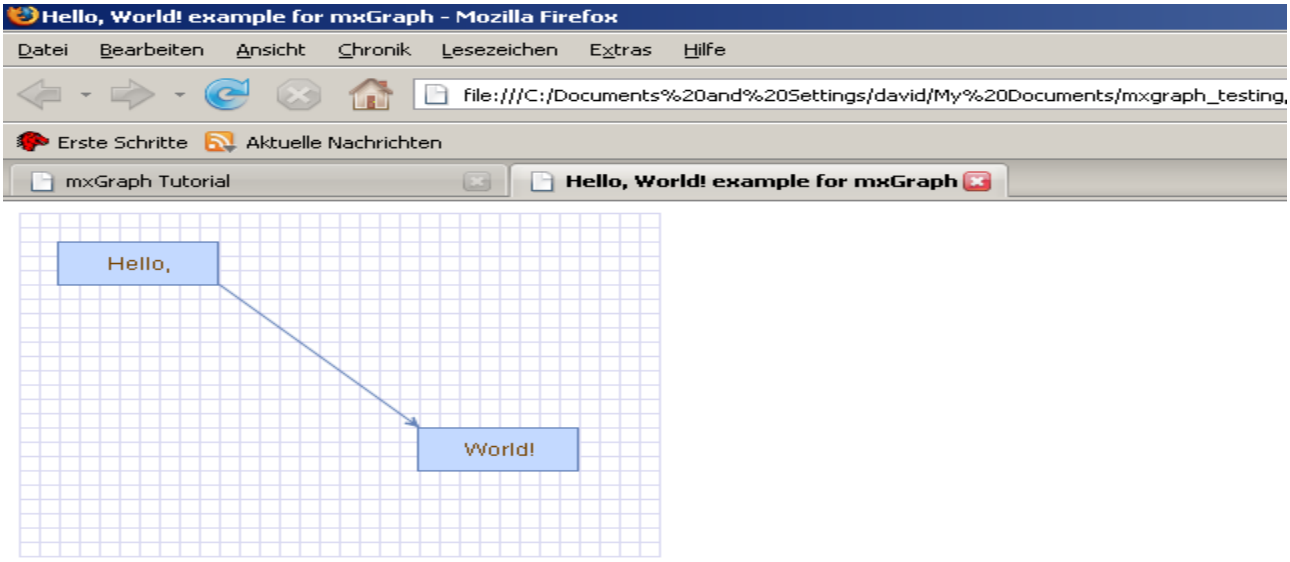
- **Crear una página HTML que vincule el JavaScript de mxGraph,**
- **Creando un contenedor para colocar el mxGraph,**
- **Agrega las celdas requeridas a ese gráfico.**

El código fuente para el ejemplo, helloworld.html, se puede encontrar a continuación y en el directorio de ejemplos tanto de la evaluación como de las versiones completas de mxGraph. La fuente HTML contiene dos secciones principales, la cabeza y el cuerpo. Estos contienen los siguientes elementos principales que puede considerar una plantilla para compilar una aplicación básica de mxGraph:

- **mxBasePath** : esta es una variable de JavaScript que define el directorio dentro del cual se espera que se encuentren los directorios css, images, resources y js. Es un código JavaScript y necesita colocarse en una etiqueta de *script* . Esto debe venir antes de la línea cargando mxClient.js y no debe tener una barra inclinada.
- **mxClient.js** : esta es la ruta a la biblioteca mxGraph. Si el archivo HTML se ejecuta localmente, la ruta puede ser local para la computadora o una ruta pública a Internet. Si la página html se descargó de un servidor web, la ruta generalmente sería una ruta de Internet pública.
- **Creación del contenedor** : en la parte inferior del código, en el elemento del cuerpo, se define la función que se solicita al cargar la página web (el valor de onload). Pasa en un contenedor div como parámetro, que se define debajo. Este div es el contenedor en el que se colocará el componente mxGraph. En este ejemplo, se aplica un fondo de cuadrícula, como se usa comúnmente en las aplicaciones de diagramación. Ninguna otra parte de los gráficos se describe en la creación del contenedor, que no sea el fondo y el ancho y alto del contenedor.

  Tenga en cuenta que el desbordamiento: estilo oculto siempre debe usarse si no desea que aparezcan barras de desplazamiento.

- **La función de entrada** : el código principal del archivo es el método de entrada ejecutado en la carga de la página en este caso. Este es el código de JavaScript y debe estar dentro de un elemento de *script de* JavaScript . Las primeras líneas de cualquier aplicación mxGraph deben ser para verificar que el navegador sea compatible y salir apropiadamente si no es así. Si el navegador es compatible, se crea un mxGraph dentro del contenedor div y se agregan tres celdas al gráfico entre las llamadas de actualización de inicio / finalización.



El ejemplo de mxGraph HelloWorld

```
<html>
<head>
   <title> ¡Hola, mundo! ejemplo para mxGraph </ title>

   <! - Establece el camino base para la biblioteca si no está en el mismo directorio ->
   <script type = "text / javascript" >
```

```
      mxBasePath = '../src';
   </ script>

   <! - Carga e inicializa la biblioteca ->
   <script type = "text / javascript" src = "../ src/ js/mxClient.js" > </ script>

   <! - Código de ejemplo ->
   <script type = "text / javascript" >
      // El programa comienza aquí. Crea un gráfico de muestra en
      // nodo DOM con la ID especificada. Esta función se invoca
      // desde el controlador de eventos onLoad del documento (ver a continuación).
      función principal (contenedor)
      {
         // Comprueba si el navegador es compatible
         if (! mxClient.isBrowserSupported ())
         {
            mxUtils.error ('¡El navegador no es compatible!', 200, falso);
         }
         más
         {
            // Crea el gráfico dentro del contenedor dado
            var graph = new mxGraph (contenedor);

            // Permite la selección del rubberband
            nuevo mxRubberband (gráfico);

            // Obtiene el elemento primario predeterminado para insertar nuevas celdas. Esta
            // normalmente es el primer hijo de la raíz (es decir, la capa 0).
            var parent = graph.getDefaultParent ();

            // Agrega celdas al modelo en un solo paso
            graph.getModel (). beginUpdate ();
            tratar
            {
               var v1 = graph.insertVertex (parent, null,
                        'Hola', 20, 20, 80, 30);
               var v2 = graph.insertVertex (parent, null,
                        '¡Mundo!', 200, 150, 80, 30);
               var e1 = graph.insertEdge (parent, null, '', v1, v2);
            }
            finalmente
            {
               // Actualiza la pantalla
               graph.getModel (). endUpdate ();
            }
         }
      };
   </ script>
</ head>

<! - La página pasa el contenedor del gráfico al programa ->
<body onload = "main (document.getElementById ('graphContainer'))" >

   <! - Crea un contenedor para el gráfico con un fondo de pantalla de cuadrícula ->
   <div id = "graphContainer"
      style = "overflow: hidden; width: 321px; height: 241px; background: url ('editors / images /
grid.gif')" >
   </ div>
</ body>
</ html>
```

Los conceptos importantes a tener en cuenta en este ejercicio son:

- mxClient.js es un archivo JavaScript que combina todo el código fuente de JavaScript de mxGraph. Al realizar la descarga desde un servidor web, obtener todo el JavaScript como un archivo es mucho más rápido que como muchos archivos separados, debido a la sobrecarga de las solicitudes / confirmaciones requeridas para cada archivo. El aumento de velocidad generalmente es de al menos x2, aunque varía según la capacidad del servidor para tener receptáculos paralelos abiertos con un cliente.
- El código JavaScript y sus dependencias están todos colocados dentro de la *cabeza* del elemento.
- Internet Explorer tiene, de forma predeterminada, opciones de seguridad habilitadas que provocan un aviso del usuario al intentar ejecutar JavaScript desde el sistema de archivos local. Esto se puede desactivar en el menú de opciones, pero tenga en cuenta que ejecutar desde el sistema de archivos local no es un escenario de implementación de mxGraph, esto solo ocurrirá durante el desarrollo.
- Su aplicación se puede escribir y vincular en la aplicación dentro del archivo HTML, o en un código fuente de JavaScript separado que esté vinculado al html de la misma manera que el archivo mxClient.js en el ejemplo.

2.4 **mxGraph Deployment and Debugging**

Hay dos versiones del archivo mxclient.js, una para uso de producción y otra para uso de desarrollo / depuración. *javascript / src / js / mxClient.js* es la versión de producción y *javascript / debug / js / mxClient.js* es para desarrollo. La primera versión tiene todos los

avances de línea eliminados para garantizar que el archivo tenga el tamaño mínimo posible. Esto tiene el efecto secundario de romper la mayoría de los depuradores de JavaScript. Durante el desarrollo, se recomienda utilizar la versión de depuración, que tiene las entradas de línea, lo que permite la depuración en los navegadores compatibles.
Ambos archivos mxClient.js son el origen de JavaScript completo de mxGraph, con todos los espacios en blanco y los comentarios eliminados para reducir el tamaño del archivo. Mientras se realiza la depuración, es más fácil utilizar los archivos fuente individuales si necesita depurar en la biblioteca mxGraph. La versión del código fuente de mxGraph contiene la fuente completa en el archivo source.zip en el directorio *javascript / devel* . Descomprimir esto en mxBasePath y eliminar la carga del archivo completo mxClient.js permite una depuración más sencilla de mxGraph. Tenga en cuenta que el archivo mxclient.js en el archivo zip de origen es un archivo de arranque que carga el resto del código fuente de JavaScript.
La velocidad de descarga de la fuente del cliente puede mejorarse aún más al comprimir el código. Todos los navegadores modernos admiten transmisiones de recepción y descompresión comprimidas en el extremo del servidor y todos los buenos servidores web admiten la detección de aquellos navegadores que no lo admiten y envían la versión no comprimida como alternativa.
Por ejemplo, en el servidor web Apache hay un módulo mod_deflate, los detalles de su uso se pueden encontrar a partir de una búsqueda estándar. El servidor jgraph.com usa este módulo y no ha habido informes de problemas en ningún navegador compatible.
El uso de compresión reduce el tamaño del archivo mxClient.js de aproximadamente 600 KB a alrededor de 130 KB. La diferencia no es notada por el usuario en la mayoría de las redes modernas, pero puede haber situaciones donde la versión más pequeña sería preferible.

Modelo y celdas de 3 mxGraph

3.1 Arquitectura Core mxGraph

3.1.1 El modelo mxGraph

El modelo mxGraph es el modelo principal que describe la estructura del gráfico, la clase se llama mxGraphModel y se encuentra dentro del paquete del modelo. Las adiciones, cambios y eliminaciones hacia y desde la estructura del gráfico tienen lugar a través del modelo de gráfica API. El modelo también proporciona métodos para determinar la estructura del gráfico, así como también ofrece métodos para establecer estados visuales tales como visibilidad, agrupamiento y estilo.
Sin embargo, aunque las transacciones para el modelo se almacenan en el modelo, mxGraph está diseñado de tal manera que la API pública principal es a través de la clase mxGraph. El concepto de "agregar esta celda al gráfico" es una descripción más natural de la acción que "agregar esta celda al modelo del gráfico". Donde es intuitivo, las funciones disponibles en el modelo y las celdas se duplican en el gráfico y los métodos en la clase de gráfico se consideran la API pública principal. A lo largo del resto de este manual, estos métodos API clave reciben un fondo rosa:
anExampleCoreAPIMethod ()
Por lo tanto, aunque muchas de las principales llamadas API se realizan a través de la clase mxGraph, tenga en cuenta que mxGraphModel es el objeto subyacente que almacena la estructura de datos de su gráfico.
mxGraph usa un sistema transaccional para realizar cambios en el modelo. En el ejemplo HelloWorld vimos este código:

```
// Agrega celdas al modelo en un solo paso
graph.getModel (). beginUpdate ();
tratar
{
   var v1 = graph.insertVertex (parent, null, 'Hello,', 20, 20, 80, 30);
   var v2 = graph.insertVertex (parent, null, 'World!', 200, 150, 80, 30);
   var e1 = graph.insertEdge (parent, null, '', v1, v2);
}
finalmente
{
   // Actualiza la pantalla
   graph.getModel (). endUpdate ();
}
```

para realizar la inserción de los 2 vértices y 1 borde. Para cada cambio en el modelo, realiza una llamada a beginUpdate (), realiza las llamadas apropiadas para cambiar el modelo y luego llama a endUpdate () para finalizar los cambios y enviar las notificaciones de cambio de evento.
Métodos clave de API:

- **mxGraphModel.beginUpdate ()** - inicia una nueva transacción o una **subtransacción** .
- **mxGraphModel.endUpdate ()** - completa una transacción o una **subtransacción** .
- **mxGraph.addVertex ()** - Agrega un nuevo vértice a la celda principal especificada.
- **mxGraph.addEdge ()** - Agrega una nueva arista a la celda padre especificada.

Nota : Técnicamente, no tiene que rodear sus cambios con las llamadas de actualización de inicio y finalización. Los cambios realizados fuera de este ámbito de actualización tienen efecto inmediato y envían las notificaciones de inmediato. De hecho, los cambios dentro del ámbito de actualización promulgan en el modelo de inmediato, el alcance de la actualización está ahí para controlar el tiempo y la concatenación de las notificaciones de eventos. A menos que el ajuste de la actualización cause problemas estéticos en el código, vale la pena usarlo por hábito para evitar posibles problemas con el evento y deshacer la granularidad.
Tenga en cuenta la forma en que los cambios del modelo se envuelven en un bloque try y endUpdate () en un bloque finally. Esto garantiza que la actualización se complete, incluso si hay un error en los cambios del modelo. Debe usar este patrón siempre que realice cambios en el modelo para facilitar la depuración.
Ignore la referencia a la celda principal por ahora, que se explicará más adelante en este capítulo.

3.1.2 El modelo de transacción

La subtransacción en el bloque azul anterior se refiere al hecho de que las transacciones se pueden anidar. Es decir, hay un contador en el modelo que se incrementa para cada llamada a *beginUpdate* y decrementos para cada llamada a *endUpdate* . Después de

aumentar a al menos 1, cuando este recuento llega a 0 nuevamente, la transacción modelo se considera completa y se activan las notificaciones de eventos del cambio de modelo.
Esto significa que cada sección de código sub-contenida puede (y debería) estar rodeada por la combinación de inicio / finalización. Esto proporciona la capacidad en mxGraph para crear transacciones separadas que se utilizan como "transacciones de biblioteca", la capacidad de crear cambios compuestos y para un conjunto de eventos que se activarán para todos los cambios y solo se creará un deshacer. El diseño automático es un buen ejemplo de dónde se requiere la funcionalidad.
En el diseño automático, el usuario realiza cambios en el gráfico, generalmente a través de la interfaz de usuario, y la aplicación coloca automáticamente el resultado de acuerdo con algunas reglas. El posicionamiento automático, el diseño, es un algoritmo independiente entre las llamadas de actualización de inicio / final que no tiene conocimiento de los detalles del cambio. Debido a que todos los cambios dentro de la actualización de inicio / final se realizan directamente en el modelo de gráfico, el diseño puede actuar sobre el estado del modelo a medida que el cambio está en progreso.
Es importante distinguir entre la funcionalidad que actúa en el modelo de gráfico como parte de un cambio compuesto y la funcionalidad que reacciona a los eventos de cambio de gráfico atómico. En el primer caso, como para el diseño automático, la funcionalidad toma el modelo tal como está y actúa sobre él. Este método solo debe usarse para partes de cambios de modelos compuestos. Todas las demás partes de la aplicación solo deberían reaccionar a los eventos de cambio de modelo.
Los eventos de cambio de modelo se disparan cuando la última llamada a endUpdate reduce el contador a 0 e indica que se ha producido al menos un cambio en el gráfico atómico. El evento de cambio contiene información completa sobre lo que se ha modificado (consulte la sección posterior sobre **Eventos** para obtener más información).

3.1.2.1 **Los métodos de cambio de modelo**

A continuación hay una lista de los métodos que alteran el modelo de gráfico y deben ubicarse, directa o indirectamente, con el alcance de una actualización:

- agregar (padre, hijo, índice)
- eliminar (celda)
- setCollapsed (celda, colapsado)
- setGeometry (celda, geometría)
- setRoot (raíz)
- setStyle (celda, estilo)
- setTerminal (celda, terminal, isSource)
- setTerminals (borde, fuente, destino)
- setValue (celda, valor)
- setVisible (celda, visible)

Inicialmente, solo nos ocuparemos de agregar y eliminar, así como de los métodos de edición de geometría y estilo. Tenga en cuenta que estos no son métodos básicos de API, como es habitual, estos métodos están en la clase mxGraph, según corresponda, y realizan la encapsulación de actualización por usted.
Antecedentes del diseño : algunas personas se confunden por la presencia de información visual almacenada por el modelo. Estos atributos comprenden el posicionamiento de la celda, la visibilidad y el estado contraído. El modelo almacena el estado predeterminado de estos atributos, proporcionando un lugar común para establecerlos por celda, mientras que las vistas pueden anular los valores según la vista. El modelo es simplemente el primer lugar común en la arquitectura donde estos atributos se pueden establecer en una base global. Recuerde, esta es una biblioteca de *visualización de* gráficos , la parte de visualización es la funcionalidad principal.

Insertar celdas

Las tres celdas de gráfico creadas en la `HelloWorld` aplicación son dos vértices y un borde que conectan los vértices. Si no está familiarizado con la teoría gráfica básica y su terminología, consulte la entrada de wikipedia .
Puede agregar vértices y bordes usando el método add () en el modelo. Sin embargo, para el uso general de esta biblioteca, aprenda que mxGraph.insertVertex () y mxGraph.insertEdge () son la API pública principal para agregar celdas. La función del modelo requiere que la celda que se va a agregar ya esté creada, mientras que mxGraph.insertVertex () crea la celda por usted.

Funciones API principales:

- **mxGraph.insertVertex (parent, id, value, x, y, width, height, style)** : crea e inserta un nuevo vértice en el modelo, dentro de una llamada de actualización de inicio / finalización.
- **mxGraph.insertEdge (parent, id, value, source, target, style) :** crea e inserta un nuevo borde en el modelo, dentro de una llamada de actualización de inicio / finalización.

`mxGraph.insertVertex()`creará un objeto mxCell y lo devolverá desde el método utilizado. Los parámetros de la función son:

- **parent** : la celda que es el padre inmediato de la nueva celda en la estructura del grupo. En breve abordaremos la estructura del grupo, pero por el momento la utilizaremos `graph.getDefaultParent();` como su elemento principal predeterminado, tal como se utiliza en el ejemplo HelloWorld.
- **id** : este es un identificador único global que describe la celda, siempre es una cadena. Esto es principalmente para hacer referencia a las celdas en la salida persistente externamente. Si no desea mantener sus identificaciones, pase null a este parámetro y asegúrese de que mxGraphModel.isCreateIds () devuelve verdadero. De esta forma, el modelo administrará los identificadores y se asegurará de que sean únicos.
- **valor** : este es el objeto de usuario de la celda. Los objetos de usuario son simplemente eso, solo objetos, pero forman los objetos que le permiten asociar la lógica comercial de una aplicación con la representación visual de mxGraph. Sin embargo, se describirán con más detalle más adelante en este manual. Para empezar, si utiliza una cadena como objeto de usuario, se mostrará como la etiqueta en el vértice o borde.
- **x, y, ancho, alto** - como sugieren los nombres, estas son la posición xey de la esquina superior izquierda del vértice y su ancho y alto.
- **estilo** : la descripción del estilo que se aplicará a este vértice. Los estilos se describirán con más detalle en breve, pero a un nivel simple, este parámetro es una cadena que sigue un formato particular. En la cadena aparece cero o más nombres de estilo y una cierta cantidad

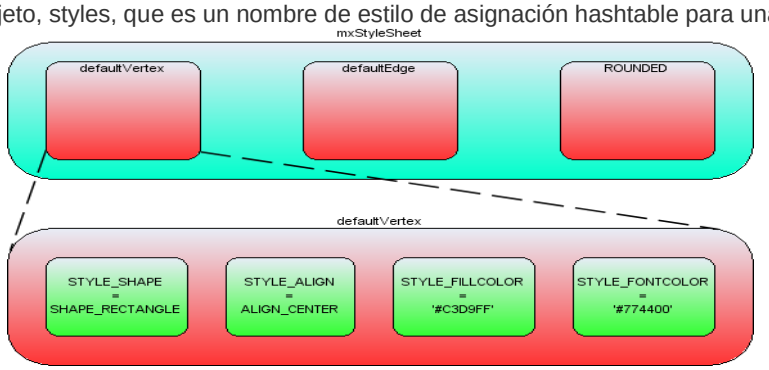
de pares clave / valor que anulan el estilo global o configuran un nuevo estilo. Hasta que creamos estilos personalizados, solo usaremos los disponibles actualmente.
Con el método de adición de bordes, los parámetros nombrados de forma idéntica realizan la misma función que en el método de adición de vértices. Los parámetros fuente y destino definen los vértices a los que está conectado el borde. Tenga en cuenta que los vértices de origen y destino ya deberían haberse insertado en el modelo.

3.1.3 **mxCell**

mxCell es el objeto de celda para ambos vértices y bordes. mxCell duplica muchas de las funciones disponibles en el modelo. La diferencia clave en el uso es que el uso de los métodos modelo crea las notificaciones de eventos apropiadas y deshacer, utilizando la celda realiza el cambio pero no hay registro del cambio. Esto puede ser útil para efectos visuales temporales tales como animaciones o cambios en un mouse sobre, por ejemplo. Sin embargo, como regla general, use la API de edición de modelos a menos que encuentre un problema específico con este mecanismo.
Al crear una nueva celda, se requieren tres cosas en el constructor, un valor (objeto de usuario), una geometría y un estilo. Ahora exploraremos estos 3 conceptos antes de volver a la celda.

3.1.3.1 **Estilos**

El concepto de estilos y hojas de estilo es conceptualmente similar a las hojas de estilo CSS, aunque tenga en cuenta que CSS se usa realmente en mxGraph, pero solo para afectar los estilos globales en el DOM de la página HTML. Abra el archivo util.mxConstants.js en su editor y busque la primera coincidencia en "STYLE_". Si se desplaza hacia abajo, verá una gran cantidad de cadenas definidas para todos los distintos estilos disponibles con este prefijo. Algunos de los estilos se aplican a los vértices, algunos a los bordes y algunos a ambos. Como puede ver, estos definen atributos visuales en el elemento sobre el que actúan.
El mxStylesheet contiene un objeto, styles, que es un nombre de estilo de asignación hashtable para una matriz de estilos:



Arreglos de estilo dentro de la colección de estilos

En la imagen de arriba, el cuadro azul representa la tabla de estilos en mxStyleSheet. La cadena 'defaultVertex' es la clave para una matriz de pares de cadena / valor, que son los estilos reales. Tenga en cuenta que mxGraph crea dos estilos predeterminados, uno para los vértices y otro para los bordes. Si mira hacia atrás al ejemplo de helloworld, no se pasa ningún estilo al parámetro de estilo opcional de insertVertex o insertEdge. En este caso, el estilo predeterminado se usaría para esas celdas.

Establecer el estilo de una celda

Si desea especificar un estilo diferente al predeterminado para una celda, debe pasar ese nuevo estilo a la celda cuando se crea (mxGraph's insertVertex e insertEdge ambos tienen un parámetro opcional para esto) o pasar ese estilo a la celda usando model.setStyle ().
El estilo que aprueba tiene la forma stylename. , tenga en cuenta que los nombres de estilo y los pares clave / valor pueden estar en cualquier orden. A continuación se muestran ejemplos para demostrar este concepto, adaptando la llamada insertEvex que vimos en helloworld:

1. Se ha creado un nuevo estilo llamado 'REDONDEADO' para aplicarlo a un vértice:

```
var v1 = graph.insertVertex (principal, nulo, 'Hola', 20, 20, 80, 30, 'REDONDEADO');
```

2. Para crear un nuevo vértice con el estilo REDONDEADO, anulando los colores de trazo y relleno:

```
var v1 = graph.insertVertex (parent, null, 'Hello', 20, 20, 80, 30, 'REDONDEADO; strokeColor = red; fillColor = green');
```

3. Para crear un nuevo vértice sin estilo global, pero con colores de trazo y relleno locales:

```
var v1 = graph.insertVertex (parent, null, 'Hello', 20, 20, 80, 30, '; strokeColor = red; fillColor = green');
```

4. Para crear un vértice que utiliza el estilo de Vettex predeterminado, pero un valor local del color de relleno:

```
var v1 = graph.insertVertex (parent, null, 'Hello', 20, 20, 80, 30, 'defaultVertex; fillColor = blue');
```

Tenga en cuenta que el estilo predeterminado se debe nombrar explícitamente en este caso, faltando el estilo de salida no se establece un estilo global en la celda cuando el punto y coma inicia la cadena. Si la cadena comienza sin punto y coma, se usa el estilo predeterminado.
De nuevo, la clase mxGraph proporciona funciones de utilidad que forman la API central para acceder y cambiar los estilos de las celdas:
Funciones API principales:

- **mxGraph.setCellStyle (estilo, celdas)** : establece el estilo para la matriz de celdas, encapsulado en una actualización de inicio / finalización.
- **mxGraph.getCellStyle (celda)** : devuelve el estilo de la celda especificada, combinando los estilos de cualquier estilo local y el estilo predeterminado para ese tipo de celda.

Crear un nuevo estilo global

Para crear el estilo global REDONDEADO descrito anteriormente, puede seguir esta plantilla para crear un estilo y registrarlo con mxStyleSheet:

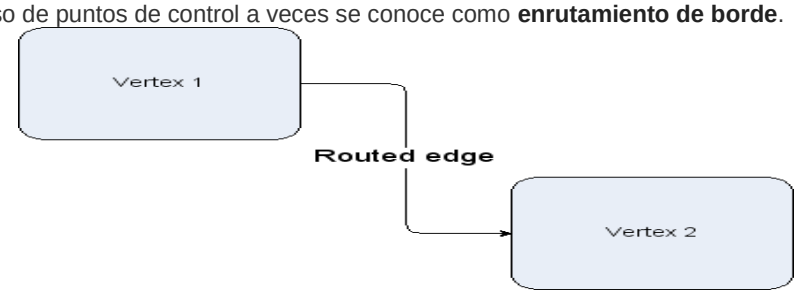
```
var style = new Object ();
style [mxConstants.STYLE_SHAPE] = mxConstants.SHAPE_RECTANGLE;
style [mxConstants.STYLE_OPACITY] = 50;
style [mxConstants.STYLE_FONTCOLOR] = '# 774400';
graph.getStylesheet (). putCellStyle ('REDONDO', estilo);
```

3.1.3.2 Geometría

En el ejemplo helloworld vimos la posición y el tamaño de los vértices pasados a la función insertVertex. El sistema de coordenadas en JavaScript es x es positivo a la derecha e y es positivo a la baja, y en términos del gráfico, el posicionamiento es absoluto para el contenedor dentro del cual se coloca mxGraph.

El motivo de una clase mxGeometry separada, en lugar de simplemente tener la clase mxRectangle almacenando esta información, es que los bordes también tienen información de geometría.

Los valores de ancho y alto se ignoran para los bordes y los valores de xey se relacionan con el posicionamiento de la etiqueta de borde. Además, los bordes tienen el concepto de puntos de control. Estos son puntos intermedios a lo largo del borde en los que se dibuja el borde al pasar. El uso de puntos de control a veces se conoce como **enrutamiento de borde**.

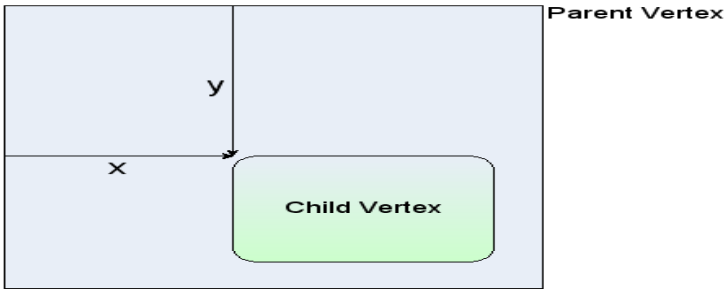


Un borde enrutado por 2 puntos de control

Hay dos conceptos adicionales más importantes en geometría, posicionamiento relativo y compensaciones

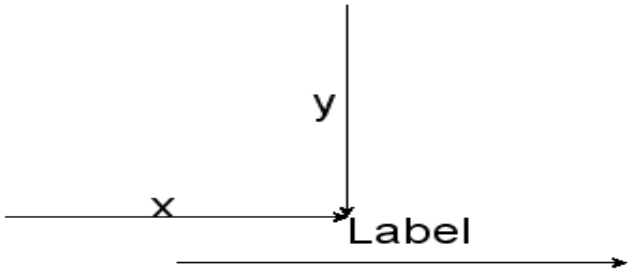
Posicionamiento relativo

De forma predeterminada, la posición xey de un vértice es el desplazamiento del punto superior izquierdo del rectángulo delimitador del elemento primario al punto superior izquierdo del rectángulo delimitador de la celda misma. El concepto de padres y grupos se trata más adelante en este capítulo, pero sin entrar en demasiados detalles, si una celda no tiene célula principal, el contenedor de gráficos es su elemento principal para fines de posicionamiento.



Posicionamiento de vértices no relativos

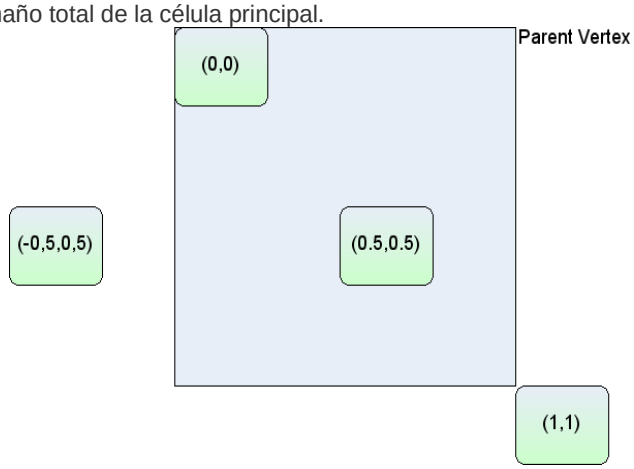
Para un borde, en el modo no relativo, que es el modo predeterminado, la posición de la etiqueta del borde es la desviación absoluta del origen del gráfico.



Posicionamiento de etiquetas de borde no relativo

Para los vértices en modo relativo, (x, y) es la proporción a lo largo de la celda principal (ancho, alto) donde se encuentra el origen de la celda. (0,0) es el mismo origen que el padre, (1,1) coloca el origen en la esquina inferior derecha del padre. El mismo posicionamiento

relativo se extiende por debajo de 0 y por encima de 1 para ambas dimensiones. Este posicionamiento es útil para mantener las células del niño fijadas en relación con el tamaño total de la célula principal.

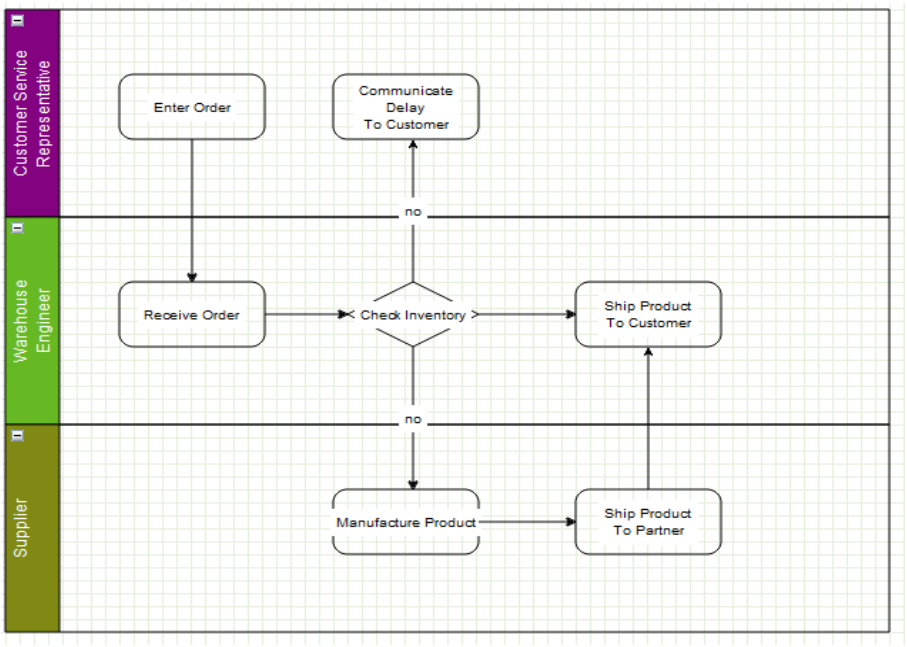


Posiciones vértices relativas

Por último, las etiquetas de borde en modo relativo se colocan en palmas según el posicionamiento desde el centro del borde. La coordenada x es la distancia relativa desde el extremo de origen del borde, en -1, hasta el extremo del borde, en 1. La coordenada y es el desplazamiento del píxel ortogonal desde el borde. El siguiente diagrama muestra los valores de x, y para varias etiquetas de borde en modo relativo. Tenga en cuenta que para una regla, los cálculos son simples. Para los bordes con múltiples puntos de control, el borde debe trazarse a lo largo de sus segmentos (un segmento es la línea entre los puntos finales y / o puntos de control) para encontrar la distancia correcta a lo largo del borde. El valor y es el desplazamiento ortogonal de ese segmento.
Cambiar el posicionamiento relativo en las etiquetas de borde es una preferencia común para las aplicaciones. Navegue a la función mxGraph.insertEdge () en mxGraph, verá que esto llama a createEdge (). En createEdge () la geometría se establece relativa para cada borde creado con este prototipo. Este es en parte el motivo de la cantidad de funciones auxiliares en mxGraph, que permiten un cambio fácil del comportamiento predeterminado. Debe intentar utilizar la API de la clase mxGraph tanto como sea posible para proporcionar este beneficio en sus aplicaciones.
Compensaciones
El campo de compensación en mxGeometry es un desplazamiento x, y absoluto aplicado a la **etiqueta de** la celda . En el caso de las etiquetas de borde, el desplazamiento siempre se aplica después de que se haya calculado la etiqueta de borde de acuerdo con la bandera relativa en la sección anterior.
Funciones API principales:

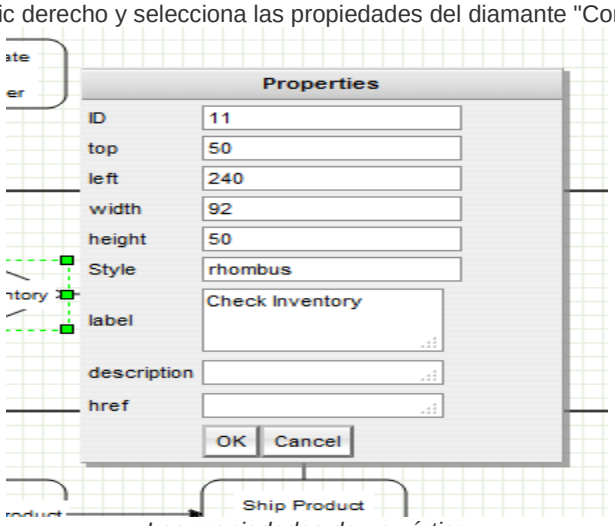
- **mxGraph.resizeCell (celda, límites)** - Cambia el tamaño de la celda especificada a los límites especificados, dentro de una llamada de actualización de inicio / finalización.
- **mxGraph.resizeCells (cells, bounds)** - Cambia el tamaño de cada una de las celdas en la matriz de celdas a la entrada correspondiente en la matriz de límites, dentro de una llamada de actualización de inicio / finalización.

3.1.3.3 **Objetos de usuario**
El objeto Usuario es lo que le da a los diagramas mxGraph un contexto, almacena la lógica comercial asociada con una celda visual. En el ejemplo de HelloWorld, el objeto de usuario acaba de ser una cadena, en este caso simplemente representa la etiqueta que se mostrará para esa celda. En aplicaciones más complejas, estos objetos de usuario serán objetos en su lugar. Algunos atributos de ese objeto generalmente serán la etiqueta que mostrará la celda visual, el resto del objeto describe la lógica relacionada con el dominio de la aplicación.
Usando el ejemplo de un flujo de trabajo simple o una aplicación de proceso, digamos que tenemos el siguiente gráfico (este ejemplo está disponible en línea , seleccione el ejemplo de Swimlanes en la ventana de tareas):



Un flujo de trabajo simple

Normalmente, este flujo de trabajo existirá en algún servidor de aplicaciones y / o base de datos. El usuario del navegador se conecta a ese servidor, o algún servidor de aplicaciones para el usuario vinculado al servidor de aplicaciones y la aplicación web del usuario solicita el flujo de trabajo de "orden". El servidor obtiene los datos de ese flujo de trabajo y los transmite al cliente.
mxGraph admite el proceso de llenar el modelo en el lado del servidor y transmitirlo al cliente, y viceversa. Consulte el capítulo posterior sobre "E / S y comunicación del servidor".
Los datos transmitidos serán tanto el modelo visual (el diagrama) como la lógica de negocios (en su mayoría contenida en los objetos del usuario). El cliente inicialmente mostrará el diagrama de arriba. Si el usuario tiene permiso para editar este flujo de trabajo, normalmente podrá hacer dos cosas: 1) editar el diagrama, agregar y eliminar vértices, así como cambiar las conexiones, y 2) editar los objetos de usuario de las celdas (vértices y / o bordes).
En la demostración en línea, si hace clic derecho y selecciona las propiedades del diamante "Comprobar inventario", verá este diálogo:



Las propiedades de un vértice

Estas propiedades muestran la geometría, etiqueta, ID, etc., pero un diálogo podría mostrar fácilmente el objeto de usuario de la celda. Puede haber una referencia a algún proceso en el motor de flujo de trabajo sobre cómo se verifica realmente el inventario. Este podría ser un mecanismo específico de la aplicación para que tanto el servidor como el cliente asignen alguna identificación a las llamadas a métodos remotos. Otro valor podría ser el tipo de objeto que devolvió el proceso, tal vez un booleano o un entero para indicar el nivel de stock en este caso. Dado ese tipo de devolución, es posible imponer restricciones con el diagrama y proporcionar alertas visuales de si, por ejemplo, la verificación de decisión de los bordes salientes no se corresponde con el tipo de retorno del vértice.
A continuación, como ejemplo, los objetos de usuario de los bordes de salida pueden contener una etiqueta y un estado booleano. De nuevo, el editor basado en mxGraph podría proporcionar los medios para alterar el valor booleano. En el servidor, al ejecutar el proceso, puede seguir los bordes que corresponden al valor booleano devuelto por el nodo de decisión.
Tenga en cuenta que el ejemplo anterior es muy específico del dominio, está ahí para explicar cómo el objeto del usuario se correlaciona con la lógica comercial de la aplicación. Visualiza cómo mxGraph crea lo que llamamos un **gráfico contextual** . El contexto está formado por las conexiones entre los vértices y la lógica de negocios almacenada dentro de los objetos del usuario. Una aplicación típica recibe

la lógica visual y de negocios de un servidor, puede permitir la edición de ambas y luego la transmite al servidor para su persistencia y / o ejecución.

3.1.3.4 **Tipos de células**

Como se describió anteriormente, mxGraph es la API principal para usar esta biblioteca y el mismo concepto se aplica a las celdas. Un estado básico de la celda no expuesta en el gráfico es si una celda es un vértice o un borde, esta llamada se realiza en la celda o en el modelo.

Hay dos indicadores booleanos en mxCell, vertex y edge, y los métodos auxiliares establecen uno de estos como verdadero cuando se crea la celda. isVertex (), isEdge () en mxIGraphModel es lo que el modelo usa para determinar el tipo de celda, no hay objetos separados para ninguno de los tipos. Técnicamente, es posible cambiar el tipo de una celda en tiempo de ejecución, pero tenga cuidado de invalidar el estado de la celda (consulte la sección posterior) después de cambiar el tipo. Además, tenga en cuenta que la variable de objeto geometría significa cosas diferentes a vértices y bordes. Generalmente, no se recomienda cambiar un tipo de celda en tiempo de ejecución.

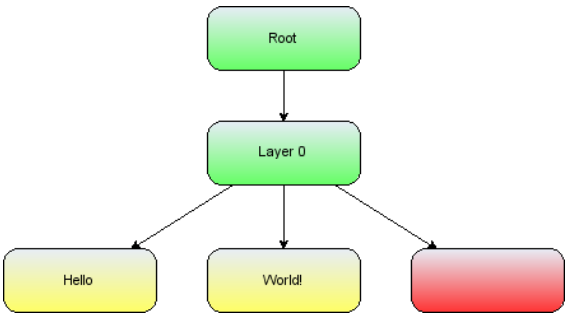
3.1.4 **Estructura del grupo**

Agrupar, dentro de mxGraph, es el concepto de asociar lógicamente celdas entre sí. Esto se conoce comúnmente como el concepto de sub-gráficos en muchos juegos de herramientas de gráficos. Agrupar implica uno o más vértices o bordes que se convierten en hijos de un vértice principal o borde (generalmente un vértice) en la estructura de datos del modelo de gráfico. Agrupar permite a mxGraph proporcionar varias funciones útiles:

- Sub-gráficos, el concepto de un gráfico lógicamente separado que se muestra en el gráfico de nivel superior como una celda por sub-gráfico.
- Expandiendo y colapsando. El colapso es la capacidad de reemplazar una colección de celdas agrupadas visualmente solo con su celda principal. Expandir es el reverso de esto. Este comportamiento se puede ver haciendo clic en el pequeño "-" en la esquina superior izquierda de las celdas de grupo del ejemplo de las piscinas en el ejemplo de workfloweditor en línea . Esto se describe en la sección *Gestión de la complejidad a* continuación.
- Layering. La estratificación es el concepto de asignar celdas a una determinada capa de orden z dentro de la pantalla gráfica.
- Profundizar, intensificar. Estos conceptos permiten que los subgráficos se visualicen y editen como si fueran un gráfico completo. En la sección *Objetos* del *usuario* , vimos el vértice "Comprobar inventario" como una sola celda. Tomemos, por ejemplo, el caso en el que un desarrollador describe cada uno de los vértices del proceso a medida que el software procesa la tarea. La aplicación puede tener una opción para profundizar en el vértice del inventario de cheques. Esto daría como resultado la aparición de un nuevo gráfico que describe en detalle cómo exactamente el sistema verifica el inventario. El gráfico puede tener el título del vértice principal de "verificar inventario" para indicar que es un niño, así como la opción de subir de nuevo al siguiente nivel.

  En la agrupación, a las celdas se les asigna una celda principal. En el caso más simple, todas las celdas tienen el padre predeterminado como su padre. El elemento primario predeterminado es una celda invisible con los mismos límites que el gráfico. Esta es la celda devuelta por graph.getDefaultParent () en el ejemplo helloworld. La posición x, y de un vértice es su posición relativa a su principal, por lo que en el caso de la agrupación predeterminada (todas las celdas que comparten el elemento primario predeterminado), la posición de la celda también es la absoluta en el componente del gráfico. En el caso de que todas las celdas se agreguen a la raíz predeterminada, la estructura del grupo se parece lógicamente, en el caso del ejemplo helloworld, al diagrama siguiente.

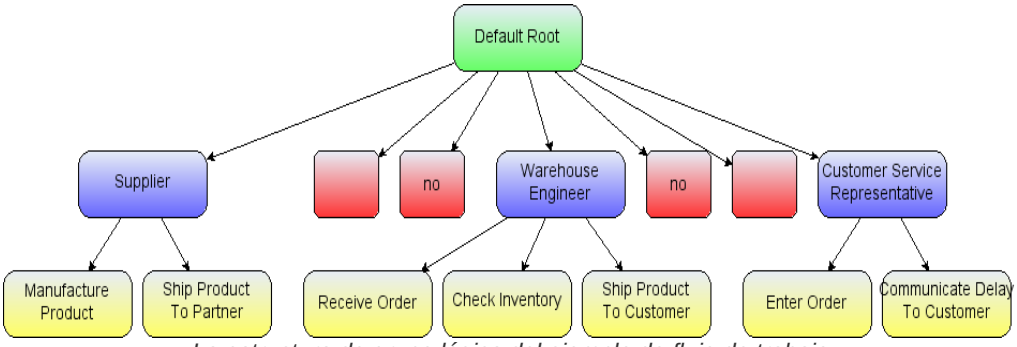
  Tenga en cuenta la adición de la celda Layer 0, esta es la indirección predeterminada en la estructura del grupo que permite cambios de capa con el requisito de celdas adicionales. Lo incluimos a continuación para corregirlo, pero en los diagramas de grupo posteriores se omitirá.



La estructura de grupo del ejemplo helloworld

Además, tenga en cuenta que la posición de la etiqueta de borde (x, y en geometría) es relativa a la celda principal.

Si volvemos al ejemplo del flujo de trabajo simple en la sección Objetos del usuario, podemos ver qué agrupamiento podría tener un aspecto visual. En el ejemplo, las celdas del grupo representan a las personas y los vértices secundarios representan las tareas asignadas a esas personas. En este ejemplo, la estructura del grupo lógico se ve así:



La estructura de grupo lógica del ejemplo de flujo de trabajo

Los vértices de acción del flujo de trabajo son los hijos amarillos y los vértices del grupo del carril de baño están marcados en azul.
Insertar celdas en la estructura del grupo se logra usando el parámetro padre de las funciones insertVertex e insertEdge en la clase mxGraph. Estas funciones establecen la célula padre en el niño en consecuencia y, de manera importante, informa a la célula padre de su nuevo hijo.
Alterar la estructura del grupo se realiza mediante las funciones mxGraph.groupCells () y mxGraph.ungroupCells ().
Funciones API principales:

- **mxGraph.groupCells (grupo, borde, celdas)** : agrega las celdas especificadas al grupo especificado, dentro de una actualización de inicio / finalización
- **mxGraph.ungroupCells (cells)** - Elimina las celdas especificadas de sus padres y las agrega al padre de sus padres. Cualquier grupo vacío después de la operación se elimina. La operación ocurre dentro de una actualización de inicio / finalización.

3.1.5 Gestión de complejidad

Hay dos razones principales para controlar la cantidad de celdas mostradas en un momento dado. El primero es el rendimiento, el dibujo de más y más células alcanzará los límites de usabilidad de rendimiento en algún punto de cualquier plataforma. La segunda razón es la facilidad de uso, un humano solo puede comprender cierta cantidad de información. Todos los conceptos asociados con la agrupación, enumerados anteriormente, se pueden utilizar para reducir la complejidad de la información en la pantalla para el usuario.

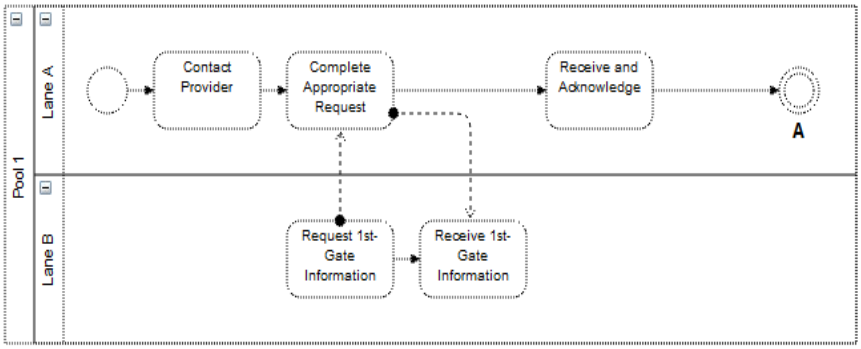
3.1.5.1 Plegable

Plegable es el término colectivo que usamos para expandir y colapsar grupos. Decimos que una celda se pliega al hacer que sus vértices hijos sean invisibles. Hay una serie de funciones relacionadas con esta función:
Función API principal:

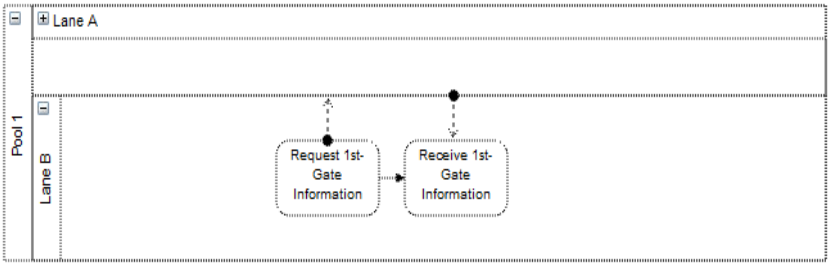
- **mxGraph.foldCells (colapso, recurse, celdas)** : indica el estado contraído de las celdas especificadas, dentro de una actualización de inicio / finalización.

Plegable funciones relacionadas:
mxGraph.isCellFoldable (celda, colapso) - Por defecto verdadero para celdas con niños.
mxGraph.isCellCollapsed (celda) - Devuelve el estado plegado de la celda
Cuando una celda de grupo se contrae, tres cosas ocurren por defecto:

- Los hijos de esa célula se vuelven invisibles.
- Se usan los límites de grupo de la celda de grupo. Dentro de mxGeometry hay un campo alternativoBounds y en celdas de grupos, por defecto almacena un límite separado para sus estados colapsados y expandidos. El cambio entre estas instancias es invocado por mxGraph.swapBounds () y esto se maneja para ti dentro de una llamada a foldCells (). Esto permite cambiar el tamaño de los grupos colapsados, mientras que cuando se expande de nuevo, el tamaño parece correcto usando el tamaño pre-contraído.
- Edge promotion ocurre, por defecto. La promoción de borde significa mostrar los bordes que se conectan a los niños dentro del grupo colapsado que también se conectan a las células fuera del grupo colapsado, haciendo que parezcan conectarse al padre contraído.



Carril expandido



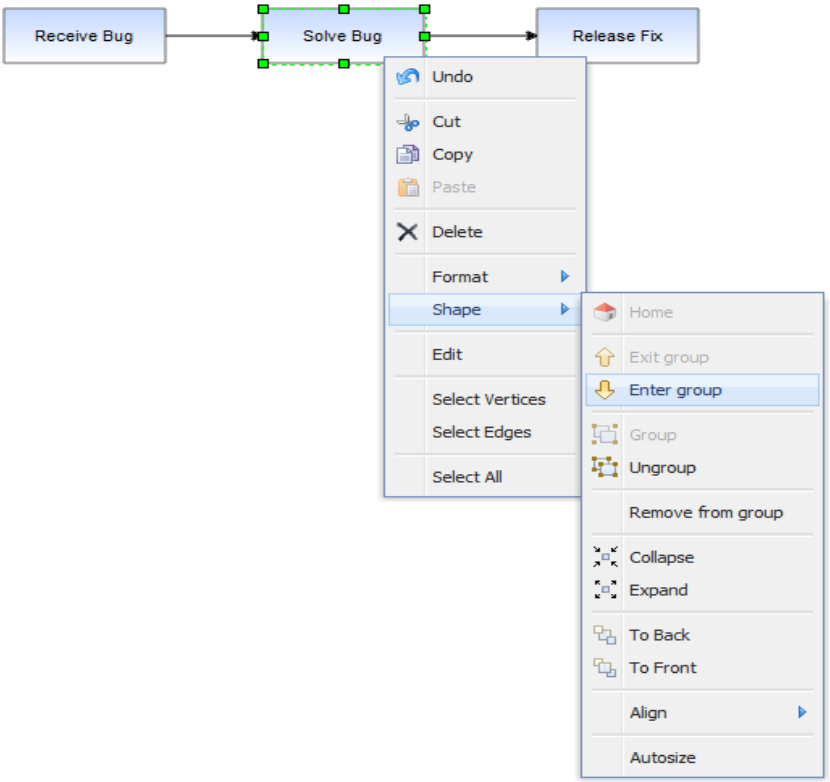
Colapsar colapsado

Las dos imágenes anteriores demuestran estos tres conceptos. En su estado expandido, la celda del grupo superior muestra una pequeña caja en la esquina superior izquierda con un carácter "-" dentro. Esto indica que al hacer clic en este cuadro se colapsa la celda del grupo. Al hacer esto, obtenemos la imagen inferior donde la celda del grupo adquiere su tamaño colapsado. Los vértices y los bordes infantiles que no salen de la celda del grupo se vuelven invisibles. Finalmente, los bordes que salen de la celda de grupo se promueven para parecer estar conectados a la celda de grupo colapsado. Al hacer clic en el carácter "+" que ahora aparece dentro del cuadro, se expande la celda del grupo y la devuelve a su estado original de la imagen superior.

Usando la función mxGraph.foldCells (), puede obtener el mismo resultado programáticamente que haciendo clic en los símbolos expand / collapse. Un uso común de esto es cuando la aplicación reduce una cantidad específica, los grupos de celdas se agrupan y la celda agrupada se contrae (muy a menudo sin el recuadro "-" dado que la aplicación controla el plegado). De esta forma, el usuario puede ver menos celdas más grandes y cada una representa lógicamente las celdas de sus hijos. A continuación, puede proporcionar un mecanismo para acercarse a un grupo, que lo expande en el proceso. También puede proporcionar desglose / intensificación, que se explica a continuación.

3.1.5.2 **Subgráficos, Drill-Down / Step-Up**

A veces, como una alternativa para expandir / colapsar, o posiblemente en combinación con ella, su gráfica estará compuesta por una cantidad de gráficos, anidados en una jerarquía. A continuación vemos un ejemplo simple:



Un ejemplo de flujo de trabajo de alto nivel

Este flujo de trabajo simple consta de tres pasos de alto nivel. Obviamente, los pasos individuales contienen una serie de subpasos y veremos un sub-gráfico de la celda *Resolver errores* .

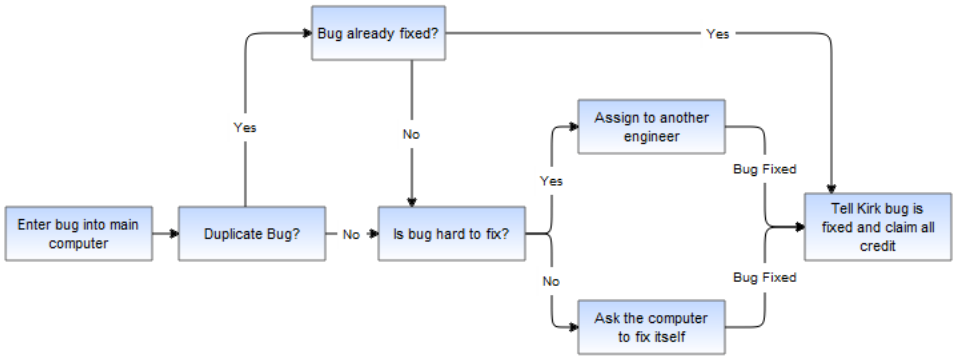
Bajo el vértice *Solve Bug* hemos creado varios niños para representar el proceso de solución de un error con más detalle, en este caso, el proceso de resolver un error en la nave espacial Enterprise .

En este ejemplo, que usa el ejemplo GraphEditor, la opción de menú que se muestra seleccionada en la imagen anterior invoca mxGraph.enterGroup (celda), que es uno de los dos pares de funciones API para subgráficos.

Funciones API principales:

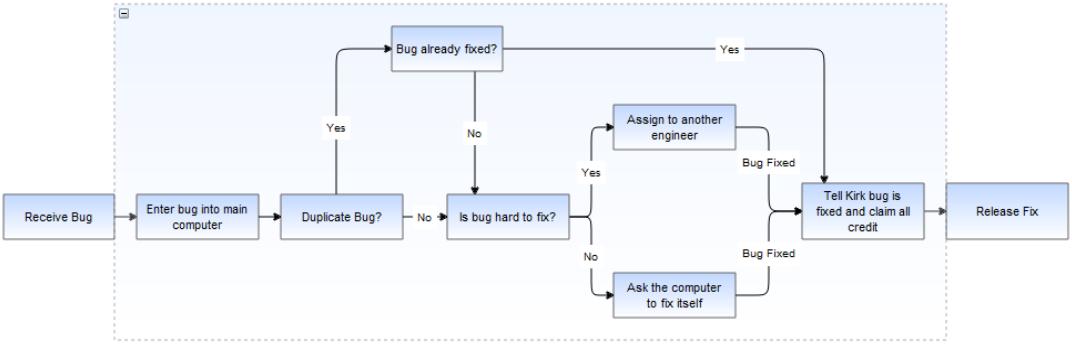
- **mxGraph.enterGroup (celda)** : convierte la celda especificada en la nueva raíz del área de visualización.
- **mxGraph.exitGroup ()** - Hace que el padre de la celda raíz actual, si lo hay, la nueva celda raíz.
- **mxGraph.home ()** - Sale de todos los grupos, haciendo que el elemento primario predeterminado sea la celda raíz.

Hasta el momento, la celda raíz del gráfico ha sido el vértice principal predeterminado para todas las celdas de primer nivel. Con estas funciones, puede hacer que cualquier celda de grupo en la estructura de grupo sea la celda raíz, de modo que los elementos secundarios de esa matriz aparezcan en la pantalla como el gráfico completo.



Resultado de la perforación hacia abajo en el vértice Solve Bug

El mismo gráfico expandido usando plegamiento se ve así:

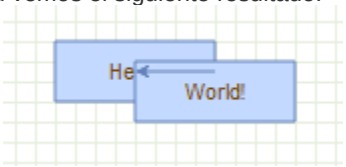


Al salir del grupo con la opción de *grupo forma-> salir* , que invoca mxGraph.exitGroup, vuelve al gráfico de nivel superior de 3 vértices original.

3.1.5.3 **Estratificación y filtrado**

En mxGraph, al igual que muchas aplicaciones gráficas, existe el concepto de z-order. Es decir, el orden de los objetos mientras miras en la dirección de la pantalla. Los objetos pueden estar detrás o delante de otros objetos y si se superponen y son opacos, el objeto que está más atrás se oscurecerá parcial o completamente. Mire de nuevo a la estructura de gráfico de la ilustración de HelloWorld anterior. Las celdas de los niños se almacenan bajo los padres en un orden determinista (por defecto, el orden en que las agrega).

Si movemos las celdas en el ejemplo de HelloWorld vemos el siguiente resultado:



Vértices superpuestos

Se puede observar que el *Mundial* vértice se encuentra en frente de la *Hola* vértice. Esto se debe a que el vértice *Mundial*tiene un índice secundario más alto que el vértice *Hola* , en las posiciones 1 y 0, respectivamente, en la colección ordenada que contiene los elementos secundarios de la celda raíz.

Para cambiar el orden, usamos mxGraph.orderCells.

Función API principal:

- **mxGraph.orderCells (atrás, celdas)** : mueve la matriz de celdas al frente o a la parte posterior de sus hermanos, según el indicador, dentro de una actualización de inicio / finalización.

Una celda hermana en mxGraph es cualquier celda que comparte el mismo padre. Entonces, al invocar esto en el vértice de *Hello* , se superpondría con el *World* Vertex.

El orden y la agrupación se pueden ampliar para formar grupos lógicamente estratificados. Las celdas se dibujan mediante una búsqueda en profundidad. Vuelva a tomar el ejemplo HelloWorld e imagine que tanto los vértices *Hello* como *World*tienen cierta jerarquía de niños debajo de ellos. El vértice *Hola* y todos sus hijos se dibujarán antes del vértice del *Mundo* o cualquiera de sus hijos. Si *Hello* y *World* eran células de grupo invisibles, entonces tienes dos jerarquías de celdas, una de las cuales se dibuja completamente antes que la otra. También puede cambiar el orden de las jerarquías simplemente cambiando el orden de las celdas invisibles del grupo.

El concepto de capas se muestra en el ejemplo layers.html. Aquí los botones se utilizan para establecer la visibilidad de las celdas de la capa de grupo. Este ejemplo se relaciona muy estrechamente con el concepto de filtrado.

En el filtrado se muestran celdas con algún atributo particular. Una opción para proporcionar funcionalidad de filtrado es verificar algún estado antes de representar las celdas. Otro método, si las condiciones de filtrado son simples y conocidas de antemano, es asignar celdas filtrables por grupos. Hacer que los grupos sean visibles e invisibles realiza esta operación de filtrado.