Mark Judd: La Integración del JavaScript

Entrevistado por Susan Florio, Traducido por Anita L. Paci

[Editor's note: This article resides in "Iris Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]

Conozcan a Mark Judd: Su lema: "Traiga la Web a Notes y Notes a la Web!" Su meta para la versión 5.0: Integrar el JavaScript con Notes. A continuación vamos a aprender como la creación de aplicaciones para Web se ha hecho mucho mas fácil.

Cuales son tus metas para la programabilidad, y específicamente el JavaScript, para la versión 5.0?
Bien, nuestra meta para la versión 5.0 fue difuminar la distinción entre el cliente Notes y cualquier navegador de Web (browser). Cualquier cosa que se puede hacer en una aplicación para Web, se debe poder hacer en una aplicación Notes y viceversa. Para lograr esto, necesitamos tener una programabilidad del lado del cliente igual en Notes como desde la Web. Entonces, como el JavaScript es esencialmente la lingua franca de programación de scripts para la Web, lo incorporamos a Notes.

La meta a largo plazo es "escribir una vez, correr en todos los partes" -- una sola aplicación para múltiples clientes. Los desarrolladores de Notes deben poder escribir una plantilla que corra igual de bien en un cliente Notes que desde un navegador de Web. Con la versión 5 tenemos un buen adelanto en esto. En el futuro, esto incluye incorporar todo el procesamiento del lado de cliente que ocurre en el navegador de Web, incluyendo la capacidad de manipular una página HTML y los datos XML mediante el JavaScript -- toda la tecnología de Web que exista -- dentro de Notes. Y continuaremos trabajando con esto en las versiones futuras.

Porque el JavaScript es tan importante para los navegadores de Web?
Es importante descargar la mayor cantidad del procesamiento posible del servidor y pasarlo al cliente. Una vez que los datos están en el cliente, una vez que un documento se carga del servidor, debes hacer lo posible para mantener lo máximo del procesamiento de los datos en el cliente. El JavaScript te permite crear aplicaciones altamente interactivas que hacen mucho del procesamiento del lado del cliente. Con esto el tráfico de la red se reduce, y el servidor se libera para atender los otros requerimientos.

La versión 5.0 soporta cual versión de JavaScript y que significa esto para los usuarios?
Las gente pregunta esto todo el tiempo, pero depende de lo que quiere decir. Quiere decir el lenguaje, o el Document Object Model (DOM), o las clases, o las librerías? Si estás hablando del lenguaje, el Domino soporta la versión 1.3 de JavaScript. Hemos adquirido la licencia del intérprete de JavaScript de Netscape y lo hemos incrustado en el Domino Designer y el cliente Notes. Para los usuarios, esto define la sintaxis del lenguaje y el conjunto de clases propias del lenguaje (built-in classes), tales como las clases de String (Cadena) y Date (Fecha).

Otro punto acerca del lenguaje es que el JavaScript es verdadamente un lenguaje orientado a objetos. Sin embargo, se basa en un modelo de "prototypical instance" (objeto prototipo) en vez de un modelo de "clases". El modelo basado en prototipos (instance-based model) puede ser muy poderoso. Pero vamos a dejar esta discusión para otro día. Lo que las personas realmente quieren decir cuando preguntan "Que versión..." es que soportamos con respecto al DOM.

Que es el DOM?
El DOM significa Document Object Model (Modelo de Objeto de Documento). Cuando un navegador de Web reproduce una página HTML, primero analiza el HTML y crea una representación de árbol en memoria de esa página. El árbol y el API que lo accesan y lo manipulan una vez creado, se llaman el DOM.

El DOM declara exactamente cuales son los objetos en el modelo. También declara los objetos que pueden "contener" ciertos objetos, y la lista de métodos, propiedades y eventos de cada objeto. Por ejemplo, declara que un documento contiene formularios, que un formulario contiene elementos, y que un elemento de texto tiene un "valor". En JavaScript, accesarías un valor de un campo de texto (o elemento) utilizando el siguiente sintaxis:

```
document.forms[0].textFieldName.value
```

Lo malo es que los distintos navegadores de Web soporta distintos DOMs. El Netscape lo hace diferentemente que el Internet Explorer de Microsoft. Además, las versiones 3.0 de los navegadores de Web soporta distintos DOMs que las versiones 4.0, las cuales soportan distintos DOMs que las versiones 5.0. El cliente Notes se va a añadir a esta confusión, ya que solamente proveemos un subconjunto de la versión 4.0 de HTML del DOM. Nuestro soporte actual de DOM se acerca mas a la definición de la versión 3.2.

Lo bueno es que existe un esfuerzo de parte del W3C para estandarizar el DOM. Estamos comprometidos en soportar su definición. Para mi, el aspecto mas importante en la guerra de los navegadores de Web no es sobre el nivel de soporte HTML, sino sobre el nivel de soporte DOM. Siempre estoy buscando el DOM mas programable. Hoy en día, IE gana, pero estés pendiente del nuevo navegador de Netscape (llamado Gecko).

Cual es la relación entre el HTML Dinámico (Dynamic HTML - DHTML) y el JavaScript?
El HTML Dinámico son muchas cosas. Sus componentes mayores son el DOM, las Hojas de Estilo en Cascada (Cascading Style Sheets - CSS) y el JavaScript. Pero es el DOM que realmente hace que el DHTML sea dinámico -- el hecho que un API existe para manipular una página después que ha sido reproducido o aun para crear la página inicialmente. Actualmente, la única asignación de lenguaje ubicuo para el DOM es el JavaScript. Busque el lenguaje Java que empezará a aparecer en los navegadores de Web futuros y en el cliente Notes también.

Cuales son los usos mas comunes del JavaScript en una aplicación Notes?
En el Domino Designer, puedes utilizar el JavaScript para programar las acciones, los botones, las zonas activas (hotspots), y los eventos de los formularios, páginas y campos. Frecuentemente vas a ver las aplicaciones tomar ventaja de estos objetos para realizar validaciones de los formularios y los campos, crear efectos de mouse-over, hacer numerosos cálculos, y mostrar las cajas de dialogo.

Que cambió en el Domino Designer entre la versión 4.6 y 5.0?
Con la versión 5.0, puedes programar el JavaScript dentro del IDE (ambiente integrado de desarrollo). Ahora, cuando diseñas una aplicación, puedes escribir el JavaScript directamente sin tener que utilizar texto HTML como tenias que hacer en la versión 4.6. Acuérdate que el texto HTML es procesado solamente por los navegadores de Web. Por otra parte, el JavaScript escrito dentro del IDE es accesado por tanto los navegadores de Web como los clientes Notes.

Los objetos que están en el DOM están expuestos mediante el panel de programación. En la versión 4.x, cuando se creaba un campo dentro de un formulario, el objeto del campo aparecía en el InfoList con sus eventos correspondientes (Entrada y Salida estaban soportados). Ahora, cuando creas un campo dentro de un formulario, vas a ver los eventos de JavaScript que corresponden al objeto DOM apropiado también. A medida que vas

creando objetos dentro de un formulario que corresponde a objetos DOM, vas a ver los eventos DOM listados al lado de los eventos NotesUI.

Varían los eventos a los cuales puedes anexar el JavaScript, dependiendo del objeto con el cual estás trabajando. Para ver cuales eventos están disponibles para cuales objetos, simplemente examines los objetos expuestos en el panel del InfoList bajo la pestaña de Objetos (Objects tab).

También, cuando estás programando un evento (event handler) en JavaScript, la pestaña de Referencia (Reference tab) en el InfoList listará los objetos DOM junto con sus propiedades, métodos y eventos.

La versión 5.0 soporta la versión 4.0 de HTML?
En el Domino Designer, puedes crear aplicaciones para Web que toman ventaja completa de los navegadores HTML 4.0. El servidor de Web Domino puede servir estas páginas de la versión 4.0. El cliente Notes todavía no maneja los eventos onMouse y onKey, entonces actualmente no soporta el HTML 4.0. Si deseas crear una sola aplicación que corra tanto en el navegador de Web como el cliente Notes, deberías programar con la especificación del HTML 3.2.

Sin embargo, aunque en el cliente Notes no se procesan los eventos de onMouse escritos en JavaScript, todavía puedes programar ciertos efectos del mouse. El Designer te permite programar los estados de "normal," "mouse over," y "pressed" de las entradas del outline, de las acciones de botones, y de los recursos de imágenes sin obligarte programar estos cambios de estado en el JavaScript. El cliente Notes respeta estos cambios de estado, haciéndose menos importante la necesidad de soportar el evento onMouse.

Va a ser mucho trabajo para los usuarios actualizarse?
En la versión 5.0, puedes correr tus aplicaciones de la versión 4.6 tal como están programadas y trabajarán perfectamente bien. Sin embargo, como dije anteriormente, el texto HTML y el JavaScript no son reconocidos por el cliente Notes. Entonces, cuando corres una aplicación en el cliente Notes, el texto HTML no funcionará. En la versión 5.0, si mueves el texto HTML al sitio apropiado dentro del IDE, el cliente Notes sí lo reconocerá.

Entonces, mientras puedes dejar tus aplicaciones como están cuando te actualizas, vale la pena migrar el código. Cuando se controla el JavaScript dentro del panel de programación, se revisa la sintaxis y se codifica con colores. También vas a poder buscar-y-reemplazar dentro de un script, dentro de todos los scripts para un objeto en

particular, o dentro de todos los scripts para todos los objetos dentro de un formulario. No vas a tener esta capacidad si tu código JavaScript es servido como texto HTML.

Que es el JS Header y como ayuda a los usuarios?
Esto realmente es un "evento" de un formulario Domino que es el sitio en el IDE en el cual puedes colocar las funciones JavaScript para tus aplicaciones. La información contenida en el JS Header es exactamente lo que los desarrolladores de Web colocan dentro una etiqueta <SCRIPT> dentro del área <HEAD> de un documento. Es un buen sitio para colocar definiciones de funciones o declaraciones globales que necesitarías en otros sitios del formulario.

Me puedes dar un ejemplo de algo que se puede hacer con el JavaScript que no se puede hacer con el LotusScript?
Bien, una cosa es correr en la Web. Pero, aparte de eso, puedes utilizar la combinación del JavaScript y el DOM para crear interfaces dinámicamente para el cliente. Por ejemplo, suponemos que estás exponiendo un interfaz de Crear consulta ("query builder") para un usuario. Cuando el usuario quiere añadir otra "cláusula" ("clause") a la consulta, la conexión JavaScript/DOM puede crear campos nuevos dinámicamente. Si utilizas la versión 4.0 o mayor del Internet Explorer y deseas ver un ejemplo de esto, véase acá "**Ejemplo del "Query Builder"**.

Esto es el fin de LotusScript?
De ninguna manera! Algunos desarrolladores pueden escoger utilizar el JavaScript para crear aplicaciones que corran en la Web. Algunos desarrolladores pueden escoger utilizar el LotusScript porque están familiarizados con el lenguaje y aprecian sus capacidades. El LotusScript es todavía un gran lenguaje de programación de scripts del cliente Notes y nosotros continuaremos soportándolo y extendiéndolo. Es particularmente práctico para el acceso a los objetos OLE, o para hacer la transición del Visual Basic.

Además, independiente del tipo de cliente al cual quieres apuntar, el LotusScript es también un gran lenguaje de programación de scripts para el lado del servidor. Lo increíble de nuestra arquitectura es que cuando añadimos nuevos objetos a las clases internas (back-end classes), todos nuestros lenguajes heredan esa funcionalidad.

Que es el "LiveConnect"?
El LiveConnect es lo que le permite el JavaScript accesar los componentes. Igual como puedes usar la "automatización" para manipular un OCX escrito en C++ utilizando el Visual Basic, puedes utilizar el "LiveConnect" para manipular un applet escrito en Java utilizando el JavaScript. Si conoces el interfaz del programador de aplicaciones (API) para un componente, puedes manejar el componente utilizando el JavaScript sin preocuparse por saber con cual lenguaje fue escrito ese componente.

Por ejemplo, si existe un applet de calendario en una página, entonces el JavaScript pudiera configurar el calendario para mostrar la fecha "actual", o abrir el mes de enero del año 2000, o cambiarse a una vista semanal, etc. En esencia, el JavaScript tiene acceso al API público entero del componente. Otra manera de ver esto es cuando colocas un applet en una página, has extendido el DOM automáticamente.

El LiveConnect forma parte de los tres clientes mayores: El Netscape Navigator, el Internet Explorer, y ahora, el Notes R5.

Puedes hablar sobre el Common Object Request Broker Architecture (CORBA) en relación al JavaScript? O como puedes accesar las clases internas de una aplicación JavaScript?
Bien, me gusta mas la segunda pregunta. Esto es simplemente una de las cosas mas increíbles de nuestro soporte de JavaScript en la versión 5.0 de Notes.

En la versión 4.6, comenzamos construyendo un interfaz de Java a las clases internas (back-end classes). Con la versión 5.0, hemos mejorado las clases internas, les hemos dado un API completo de Java, y hemos creado un applet (llamado lotus.domino.AppletBase) que te permite accesar las clases internas desde Java. Entonces, combinando este applet con el milagro del LiveConnect y un poco de JavaScript, obtienes acceso desde el cliente a los objetos del back-end. Los programadores de LotusScript hacen esto todo el tiempo; constantemente están

recorriendo el back-end sin preocuparse por el hecho que están corriendo scripts del cliente. Pues, ahora se puede hacer lo mismo con el JavaScipt!

Pero vamos a contestar tu pregunta sobre CORBA. Da la casualidad que cuando estás corriendo el applet AppletBase en el cliente Notes, nosotros utilizamos el protocolo RPC nativo de Notes para accesar el back-end. Cuando estás usando un navegador de Web, utilizamos el protocolo IIOP para hablar con un ORB del servidor para accesar el back-end. Esto simplemente sucede automágicamente. El Domino sabe cual de los protocolos hay que utilizar en cada caso.

Déjeme confirmar lo que dije. El applet AppletBase no es un applet de CORBA. Solamente provee el acceso fácil a los objetos del back-end. Algunas veces utilizará el protocolo de transporte de CORBA; algunas veces utilizará otro protocolo. Después de la versión 5.0, estamos preparados para aumentar el conjunto de protocolos que utilizamos, pero los desarrolladores de Notes no tienen que preocuparse por esto. El API para los Objetos Domino se mantienen constante; el protocolo puede variarse atrás del escenario para tomar ventaja de cualquier plataforma o para entonar el rendimiento.

Cual fue tu papel en el esfuerzo de JavaScript?
Yo era la espina. Andaba todo el tiempo pontificando: "Quien necesita el LotusScript? Quien necesita el lenguaje de fórmulas? No funcionan en la Web. Dame el JavaScript!" La gente se cansó mucho de mi. Pero el punto era ponerles las anteojeras, ser el defensor del desarrollador de la Web, y ayudar en definir una experiencia de JavaScript para los desarrolladores de Notes en general. Me pusieron el apodo de "Fanático de JavaScript", lo cual no solamente es mucho menos pomposo que el apodo "Evangélico de JavaScript", sino que también reconoce libremente el efecto de las anteojeras.

El esfuerzo de JavaScript tuvo una cantidad tremenda de trabajo. El equipo de Lenguajes tuvo que meter el intérprete de JavaScript dentro del Domino Designer y el cliente Notes, y luego conectarlo mediante el LiveConnect a nuestra Maquina Virtual de Java (Java Virtual Machine). El equipo de Designer tuvo que incorporar el JavaScript dentro del IDE y crear las conexiones para los eventos del W3C HTML 4.0. El equipo Domino tuvo que servir estos eventos constantemente y generar mas código JavaScript para que todo funcionara correctamente -- no es una tarea fácil dado los distintos sabores de navegadores de Web que hay en el mercado. El equipo del cliente Notes, con el reto mas grande, tuvo que asignar el DOM al editor existente de Notes. El grupo de Applets tuvo que desarrollar los componentes de Java los cuales el JavaScript los conecta para crear una experiencia de Notes dentro un navegador de Web. El equipo de back-end de Notes tuvo que ajustar los Objetos Domino para hacerlos accesibles utilizando el Java y el JavaScript desde tanto el cliente Notes como el navegador de Web. Los desarrolladores de las plantillas tuvieron que hacer los ajustes necesarios para lograr un paradigma mas céntrico de Web. Y, hay el esfuerzo del equipo de Ingeniería de Calidad (QE), del equipo de documentación y del HelpDesk, etc., etc.

Entonces, como puedes ver, Yo tuve el trabajo mas fácil. Yo me metí en el medio y seguí empujando a todo el mundo para que se lograra todo.

Que estamos haciendo con el JavaScript después de la versión 5.0?
Lo primero es mejorar el nivel de soporte DOM en el cliente Notes para que soporta el HTML 4.0 (y mas allá). Ambos el Designer y el cliente Notes necesitarán soportar el CSS para poder hacer esto. Habrá mejor soporte de librerías JavaScript en el Designer igual como el soporte de JavaScript directo (inline). También vamos a crear mas applets de Java y librerías de JavaScript que imitan la funcionalidad de Notes.

Esperamos que el DOM (como está definido por el esfuerzo de estandarización del W3C) llegará a ser la manera en que los usuarios manipulan los documentos del lado del cliente. Vamos a estar combinando las clases del NotesUI con el modelo DOM. Por ejemplo, el UIView de Notes va a ser un objeto/applet que soporta un API que es la unión del NotesUIView actual y el ViewApplet. Vas a accesar un NotesUIView en el DOM como si fuera una etiqueta <OBJECT> en una página HTML.

Mas y mas de nuestros componentes internos van a ser expuestos externamente. Los clientes van a poder subclasificar (subclass) nuestros componentes para personalizar mejor sus propias aplicaciones. Si no te gusta nuestro Control de Tiempo (Time Picker), por ejemplo, podrás incluir tu propio control. El acceso de JavaScript al componente personalizado se mantiene constante.

En adición, los interfaces del XML solamente han sido utilizados internamente en la versión 5, pero van ser disponibles externamente. El cliente Notes llegará a soportar un DOM XML accesado naturalmente utilizando el JavaScript. Y, lo mas importante para mi, porque soy un neófito de LotusScript y de los lenguajes de fórmulas, es que me gustaría ver aparecer el JavaScript en el motor de scripts del lado del servidor.

Algunas otras palabras?
Bien, aquí está el pronóstico final: el DHTML ya no es solamente para la Web. Desde mi perspectiva, ya está llegando a ser una capa de ventana para todas las aplicaciones en todas las plataformas. Sea el Windows o el UNIX, pronto los interfaces gráficos de usuario (GUI) estarán escritos en DHTML, y no el C++. Y acuérdate, el JavaScript va de mano en mano con el DHTML. (Pero, Yo soy solamente el "Fanático de JavaScript".)

Biografía:
Aquí en Iris, Mark Judd es el "Fanático de JavaScript". En las vidas anteriores, el fue un "Fanático de Java" (actualmente, sigue siéndolo), y un "Fanático de Ada" (pero ya no). Tiene 2 años trabajando en Iris. Primero trabajó con el grupo de Java Applet antes de concentrarse en el esfuerzo de JavaScript y DHTML. Antes de entrar en Iris, construía software para los servicios integrados digitales sobre una red de fibra óptica (hacen 20 años), sistemas videotex (hacen 15 años), IDEs de Ada (hacen 10 años), IDEs 4GL (hacen 5 años), y (mas recientemente) aplicaciones Java/Web.

Fuera del trabajo, Mark y su esposa Terri hacen el papel de acompañantes de sus hijos Zak, Zoe, y Alexis a sus juegos de fútbol. Aunque Mark ha perdido la esperanza de tener una Navidad con nieve, todavía la espera para poder inundar su jardín y patinar con los hijos.

Copyright 1999 Iris Associates, Inc. all rights reserved.