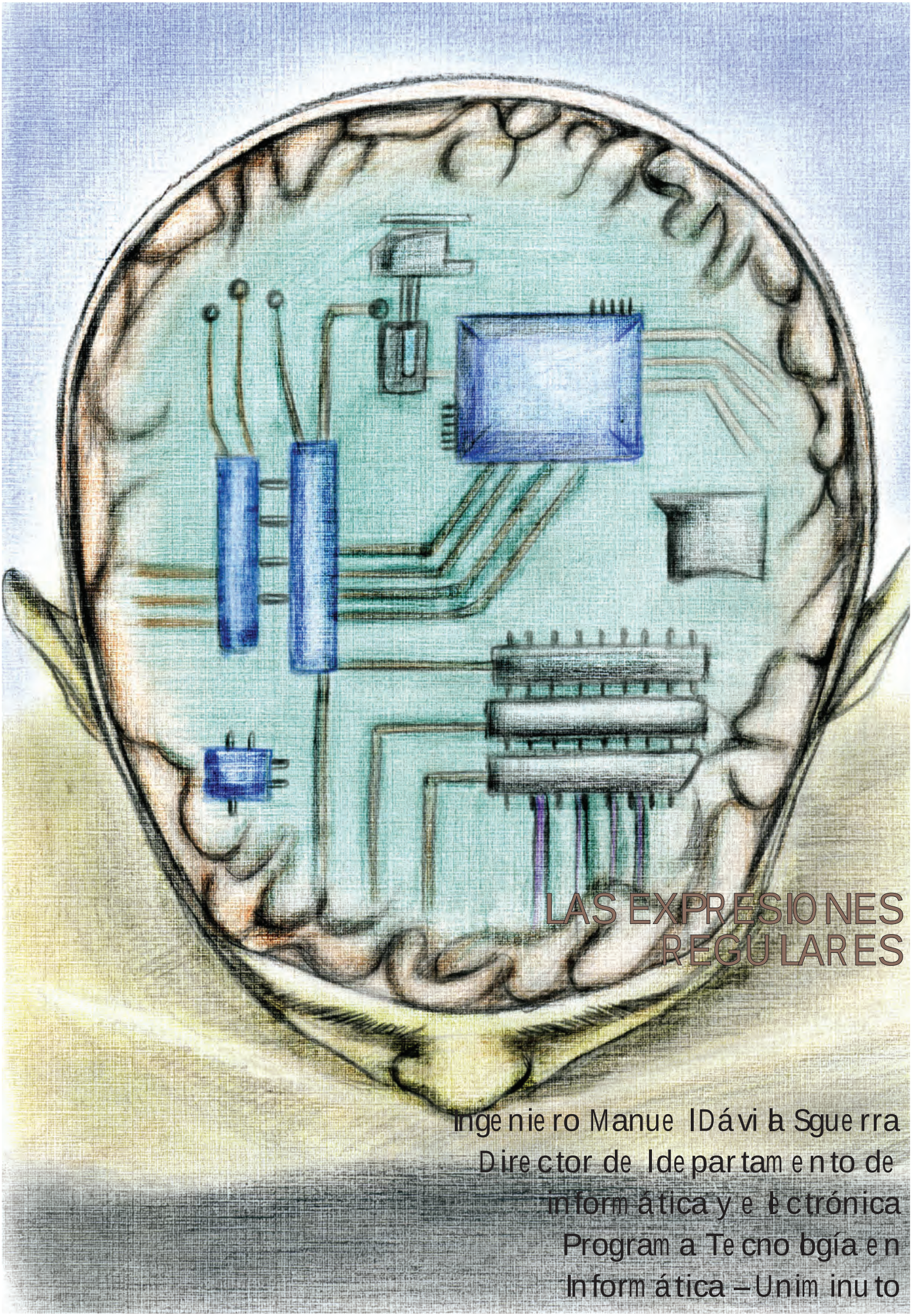

LAS EXPRESIONES REGULARES

Ingeniero Manuel Dávila Sguerra
Director del Departamento de
Informática y electrónica
Programa Tecnología en
Informática – Unimínuto

Las expresiones regulares
/^\/[\/\-[A-Za-z0-9]+\/$/)

"La expresión debajo del título no es un error de la impresora. Es una expresión regular la cual entenderemos después de leer este documento."

Introducción

Los sistemas de información antes de los años 90, se caracterizaban por el diseño de archivos, que aquí llamaremos repositorios de datos o simplemente repositorios, definidos como registros de información que contenían campos de datos "bien definidos".

Se entiende por "bien definido" el hecho de tener un tipo de información de características preestablecidas, como campos Alfanuméricos, numéricos de tipo entero o de doble precisión y cuyo contenido casi siempre obedecía a diseños preestablecidos o en el caso de los campos alfanuméricos descripciones no muy estrictas en su contenido.

Estos repositorios se acceden a través de proposiciones sql en el caso de las Bases de datos relacionales, que permiten conocer los registros o filas de las tablas que contienen datos con las características buscadas.

Es así que una proposición estándar de sql vigente aun hoy en día puede leerse como:

select Nombre from Clientes where saldo > 1000000;

la cual expresa el deseo de seleccionar del archivo o tabla de Clientes aquellos Nombres cuyo saldo es mayor de un millón de pesos.

Aun hoy en día este tipo de búsqueda es habitual y la tecnología de las Bases de datos relacionales se usan bajo este mismo contexto.

Pero, después de los 90's aparece Internet. Y el mundo se hizo pequeño para acercar a la gente y acelerar las posibilidades de publicar contenidos para ser consultados por todos.

En esta fase de la informática, cobra valor la información publicada en forma netamente textual combinada por la magia de los hiper vínculos y toda la tecnología html, hoy más caracterizada por el xml.

Para los sistemas de información basados en repositorios de datos este tipo de información es simplemente un campo más dentro de los registros clasificados como campos "memo" en los años del famoso dBASE o simplemente un campo que contiene texto libre.

Pero no se visualizan igual cuando se trata de los sistemas de búsqueda.

¿Por qué ?

Por una característica fundamental.

Los datos de texto libre podríamos decir que son datos de tipo anárquico, es decir que su contenido puede tener cualquier cosa. Útil o no, consistente o inconsistente, por su naturaleza de libre expresión.

Pero los sistemas de búsqueda tienen una responsabilidad muy grande y es desentrañar, descubrir, y encontrar información con significado dentro de los datos anárquicos.

Y esto creó un cambio fundamental en el diseño de los sistemas de información orientados a la web.

Y fueron así que las Expresiones regulares reaparecieron en el escenario para ser las salvadoras.

Qué son las expresiones regulares.

Dos neurofisiólogos, Warren McCulloch (http://en.wikipedia.org/wiki/Warren_McCulloch), y Walter Pitts (http://en.wikipedia.org/wiki/Walter_Pitts), describieron en 1940 el sistema nervioso haciendo un modelamiento de las neuronas como si fuera un autómata es decir como una máquina abstracta con un estado finito de memoria. Forma esta, de modelar, muy usada en la Ciencia de la Computación.

Recordemos a Alan Turing que creó la máquina de Turing para definir matemáticamente un algoritmo usando "procedimientos mecánicos" y que es un ejemplo importante de lo que significa "modelar" en las Ciencia de la Computación..

Alan Turing 1912-1954

Pero regresemos al estudio de los Neurofisiólogos.

Más tarde un matemático llamado Stephen Kleene describe estos modelos usando una notación matemática llamada conjuntos regulares, hasta que esta notación aparece en un proyecto relacionado con la Computación en la persona de Ken Thompson, en 1968, el creador del lenguaje B y el precursor del lenguaje C de Dennis Ritchie, ambos creadores de Unix.

Ken Thompson y Dennis Ritchie

Lo que hizo Thompson fue construir la notación mencionada dentro de un editor de texto llamado ged, el cual fue después llevado a Unix en donde se implementó en múltiples programas de utilidad como grep, expr, awk, Emacs (de Richard Stallman el creador del movimiento GNU del Software Libre), vim, ex y el lenguaje Perl.

En el caso de las expresiones regulares de Perl estas se derivan de regex que fue escrito por otro pionero llamado Henry Spencer (http://en.wikipedia.org/wiki/Henry_Spencer). De aquí nació la librería pcre Perl o "Compatible Regular Expressions" usada en las herramientas modernas.

Pero que hacen las expresiones regulares?

Su objetivo es buscar de manera inteligente información dentro de datos anárquicos para extraer de ellos lo que siempre se busca con la informática: información bajo contexto.

Como se puede observar, las expresiones regulares han existido desde Unix de tal manera que no estamos hablando de tecnologías recién creadas.

Nuestro trabajo consistirá en indagar su importancia en los nuevos paradigmas de desarrollo de software en este caso orientado a la web.

Los primeros pasos.

Imaginémonos que nos llega a las manos un texto anárquico pero que nuestro primer interés es encontrar en él algún indicio de haber sido escrito con amabilidad. Primero definimos que un texto amable es aquel que involucra palabras amables, como "por favor", o "si usted tiene la amabilidad", y otras posibles formas de expresión que califiquen de amable la manera como fue escrito.

En este caso supongamos que la palabra "por favor" es la más indicativa de esa personalidad de un texto.

Aclaro que usaré el lenguaje Perl para estas explicaciones por ser uno en los que se ha implementado regex de la mejor manera.

El programa es:

Nombre del programa: amable.pl

Objetivo: Indicar si el texto es amable.
Indicativo de amabilidad: la existencia de las palabras "por favor".

```
#!/usr/bin/perl
print ("Hágame una pregunta amable:\n");
$pregunta = <STDIN>;
if ($pregunta =~ /por favor/) {
                print ("Gracias por ser
amable!\n");
                }
                else {
                print ("Esto no fue amable!\n");
                }
```

Por ser el primer programa mostrado en Perl dentro de este documento, doy la siguiente explicación en la secuencia en que aparecen las instrucciones:

Indicar en donde está el interpretador de perl dentro de un sistema Linux o Unix : #!/usr/bin/perl

Pedir un texto por el teclado: $pregunta = <STDIN>;

Y analizar si es amable:

```
if ($pregunta =~ /por favor/) {
                print ("Gracias por ser amable!\n");
                }
                else {
                print ("Esto no fue amable!\n");
                }
```

Observe el operador =~ que indica la existencia de una expresión regular.

Comienzo y final de línea.

Es corriente necesitar hacer una búsqueda de algún texto sabiendo que este debe estar o bien al comienzo de la línea o al final.

Este ejemplo muestra cómo comienzan a aparecer caracteres especiales que indican el tipo de búsqueda pero que a la vez hace críptica la manera de expresarlo. Los caracteres ^ y $, por ejemplo, no significan nada en este renglón, pero vamos a ver como, unos renglones abajo su significado cobra vida.

Para seleccionar el texto de comienzo de línea se usa el carácter ^

Para seleccionar el texto de final de línea se usa el carácter $

Por ejemplo si se desea buscar las líneas que comiencen por la palabra computador la especificación correcta es ^computador

Si esta palabra existe en varias líneas pero no comienzan por ella, no son encontradas por esta expresión regular.

Si se desea encontrar las líneas en las cuales esta palabra aparece al final, la especificación correcta es computador$

Y si se desea encontrar las líneas en las cuales solo exista esta palabra, la especificación correcta es ^computador$

Clases de caracteres

Es muy común que se necesite buscar una palabra que tenga varias formas de escribirse de manera correcta. Es el caso de la búsqueda de un nombre de persona en donde podría aparecer comenzando con minúscula o con mayúscula. Por ejemplo el nombre Pedro o pedro.

Existe un "constructor" llamado clase de caracteres que permite manejar estas acepciones.

Obsérvese que en este caso lo que necesitamos es encontrar la palabra edro iniciando por P o p

La manera de especificarlo usando Clases de caracteres es [Pp]edro

Otro caso es cuando se busca una cadena de caracteres del estilo H1, H2, H3, H4, H5, H6 etc; es decir la letra H seguida por un número entre 1 y 6, como es frecuente en las hojas html.

Una especificación sería H[123456]
Sin embargo aparece un metacaracter de rango que es el – que permite especificar un rango de tal manera que en lugar de 123456 podemos decir 1-6

De esta manera otra forma de especificar lo anteriores:

H[1-6]

De la misma manera es posible indicar la presencia de minúsculas [a-z] o Mayúsculas [A-Z]

Negación de caracteres

En ocasiones nuestro interés es encontrar texto en el cual no debe existir alguna cadena de caracteres.

Por ejemplo si se necesita que el texto no contenga los números de 10 a 20, la especificación correcta sería:

[^10-20]

Obsérvese que el carácter ^ significa algo diferente fuera del constructor Clase de caracteres []

Alternación.

Para encontrar alguna de varias expresiones dentro de un texto se usa la alternación.

Es el caso de querer encontrar Pedro o pedro, cadena de caracteres en los que ya indagamos una forma de hacerlo a través de las Clases de caracteres.

Otra forma de lograrlo es (Pedro|pedro), en donde el carácter | tiene el significado de la expresión lógica o

Es decir encuentre las palabras pedro o Pedro dentro del texto. Los paréntesis solo encuadran la alternación para delimitarla.

Es muy común al desarrollar software que tiene que leer correos electrónicos en los cuales se necesita analizar por ejemplo aquellas líneas de texto que vienen con From, Subject o Date como primeras cadenas de texto.

Una forma de expresar esa búsqueda es:

^(From|Subject|Date:)

Lo cual significa que busque líneas que comiencen por:

1. From seguido de :o
2. Subject seguido de :o
3. Date seguido de :

Palabras límite.

En ocasiones es necesario buscar palabras completas dentro del texto que no estén embebidas dentro de la línea.

Para esto se usa la cadena de metacaracteres \< y \> rodeando la palabra.

Por ejemplo \<computador\> buscará aquellas líneas en donde se encuentre la palabra computador aislada.

Cuantificadores.

Estos símbolos permiten indicar si se desea obtener un resultado basado en un símbolo que debe aparecer opcionalmente, una vez, o varias.

En los sistemas operacionales es conocido el * que indica por ejemplo, en casa* la cadena de caracteres cas y la letra a cero o muchas veces, de tal manera que seleccionará cas casa casaa casaaa etc

El metacaracter + indica uno o más del carácter precedente. Así, casa+ indica la selección de casa, casaa, casaaa etc

Y por último el metacaracter ?Que indica uno opcionalmente, de tal manera que casa? Seleccionará aquellas líneas que contienen cas o casa solamente.

Un ejemplo muy útil para quienes desarrollan software para la web es el caso de html en donde es común tener que buscar sobre expresiones del estilo <HR SIZE = 9 > que de acuerdo con la sintaxis de lhtml permite uno o varios espacios entre las marcas. Por ejemplo <HR SIZE = 9 > o <HR SIZE = 9 > son válidas.

Cómo, entonces, usando estos cuantificadores podemos crear una expresión que detecte esta sintaxis?

Veamos esta expresión regular:

<HR(+SIZE *= *[0-9]+)?>

y analicémosla.

Para claridad usaremos el símbolo b para expresar la existencia de un espacio en blanco.

Esta expresión encontrará aquellas líneas en donde aparezca:

la cadena <HR
seguido de:

b+ mínimo un espacio, o más. Nótese que antes de + hay un espacio en blanco

seguido de la cadena SIZE

b*= con ningún espacio seguido o muchos de ellos y luego el carácter =. Nótese que antes del * hay un espacio en blanco.

b* seguido de cero espacios o muchos de ellos. Nótese que antes del * hay un espacio en blanco

[0-9] Seguido de un número que debe estar entre 0 o 9
+ el cual debe ser mínimo uno o varios de ellos
y esta expresión de lestib SIZE = 9 puede:

?existir o no, pero una sola vez si existe

b* seguido de ninguno o muchos espacios. Nótese que antes de * hay un espacio

y terminado en >

Para finalizar

La expresiones regulares van mucho más allá de lo que aquí hemos visto, pero como este artículo no pretende ser un tutorial de ellas sino una base para aquellos que quieren profundizar en ellas, no profundizaremos más por ahora.

Terminemos examinando la expresión regular del lTítulo:
/^[\/\-[A-Za-z0-9]+\/$/]

^\/ el texto debe comenzar por / La presencia del carácter \ se denomina escape y se aplica a aquellos caracteres que son de control pero que queremos interpretar los como carácter puro.

[\/\-[A-Za-z0-9] esta es una clase de caracteres debido a la presencia de [] rodeando a la expresión. Dentro de ella aparecen:
\/ Un slash
\- carácter menos
[A-Za-z0-9] Una letra mayúscula es decir A-Z, o minúscula es decir a-z o un número 0-9
+ Una o más veces
\/ Otro slash,
$/ finalizando el texto

Caben dentro de este texto cadenas de caracteres de lestib:

/usr/ o /programas/contabilidad/ o lo que en general se denominan directorios dentro de un sistema operacional.

Conclusión

La expresiones regulares son "el mejor amigo del hombre" cuando este es programador. O de las mujeres también, claro está. Su uso facilita muchas soluciones dentro de los algoritmos de los programas y hace muy poderosas dichas soluciones.

Es críptico y enredado, pero como todo en la vida, es difícil hasta que se sabe cómo funcionan, por lo que invitamos a los programadores a seguir avanzando en su aprendizaje.

BIBLIOGRAFIA

[1] Jeffrey E.F. Friedl, Mastering Regular Expressions, O'Reilly,1997
[2] David Till, Teach yourself Perl5, Sams publishing, 1996
[3] Ellen Siever, Stephen Spainhour, Nathan Patwardhan, The PerlCD Bookshelf, O'Reilly, 1998
[4] Larry Wall, Tom, Christiansen, Randal L. Schwartz, Programming Perl- Second Edition, O'Reilly,
1996
[5] Sriram Srinivasan, Advanced PerlProgramming, O'Reilly,1997
[6] Shishir Gundavaram, CGIProgramming on the World Wide Web, O'Reilly,1996