(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 559 638**

(51) Int. CI.:

H04N 21/84 (2011.01)
H04N 21/8543 (2011.01)
H04N 21/8545 (2011.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **01.02.2002 E 02703287 (9)**

(97) Fecha y número de publicación de la concesión europea: **11.11.2015 EP 1356381**

(54) Título: **Método y equipo para la compilación de un lenguaje interpretativo para televisión interactiva**

(30) Prioridad:

02.02.2001 US 265986 P
02.02.2001 US 266210 P
09.02.2001 US 267876 P
15.02.2001 US 269261 P
28.03.2001 US 279543 P
12.10.2001 US 328963 P

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
15.02.2016

(73) Titular/es:

OPENTV, INC. (100.0%)
275 Sacramento Street
San Francisco, CA 94111, US

(72) Inventor/es:

WILLARD, PIERRE

(74) Agente/Representante:

TOMAS GIL, Tesifonte Enrique

DESCRIPCIÓN

Método y equipo para la compilación de un lenguaje interpretativo para televisión interactiva

Aviso de derechos de autor

[0001] Una parte de la divulgación de este documento de patente contiene material (listados de código y listados de mensaje) sobre los que se hace la reivindicación de protección de derechos de autor.
El propietario de derechos de autor no tiene objeción a la reproducción de facsímil por cualquier persona del documento de patente o la divulgación de patente, como aparece en el documento de o registro de U.S. Patent and Trademark Office, pero se reserva cualquier otro derecho.

Antecedentes de la invención

Campo de la invención

[0002] La presente invención se refiere al campo de muestra de contenido de televisión interactiva y específicamente a la extracción de un lenguaje interpretativo, por ejemplo, JavaScript de lenguaje de marcado de texto, por ejemplo, páginas HTLM y la compilación del JavaScript a un servidor para descarga y a un dispositivo de cliente para ejecución en la muestra de contenido proporcionada por un emisor, Internet o memoria en un espacio de muestra de televisión interactiva.

Resumen de la técnica relacionada

[0003] Sistemas de televisión interactiva pueden utilizarse para proporcionar una amplia variedad de servicios a espectadores.
Sistemas de televisión interactiva son capaces de entregar corrientes de programa de vídeo típicas, aplicaciones de televisión interactiva, texto e imágenes gráficas, páginas web y otros tipos de información.
Sistemas de televisión interactiva son también capaces de registrar acciones de espectador o respuestas y se pueden usar para este tipo de fines como marketing, entretenimiento y educación.
Usuarios o espectadores puede interactuar con los sistemas pidiendo productos publicitados o servicios, compitiendo contra los participantes en un concurso televisivo, solicitando información especializada con relación a programas particulares, o navegando a través de páginas de información.

[0004] Típicamente, un proveedor de servicio de radiotransmisión o controlador de red genera una señal de televisión interactiva para transmisión a una televisión del espectador.
La señal de televisión interactiva puede incluir una parte interactiva consistente en código de aplicación o información de control, al igual que una parte audio/vídeo consistente en un programa televisivo u otros monitores informacionales.
El proveedor de servicio de radiotransmisión combina las partes audio/vídeo e interactivas en una única señal para transmisión a un receptor conectado a la televisión del usuario.
La señal es generalmente comprimida antes de transmisión y transmitida a través de típicos canales de emisión, tal como líneas de televisión por cable (CATV) o sistemas de transmisión por satélite directos.

[0005] Típicamente, un decodificador conectado a la televisión controla la funcionalidad interactiva de la televisión.
El decodificador recibe una señal de emisión transmitida por el proveedor de servicio de radiotransmisión, separa la parte interactiva de la parte audio-vídeo y descomprime las partes respectivas de la señal.
El decodificador usa la información interactiva, por ejemplo, para ejecutar una aplicación mientras la información audio/vídeo se transmite a la televisión.
El decodificador puede combinar la información audio/vídeo con gráficos interactivos o audio generado por la aplicación interactiva antes de transmisión de la información a la televisión.
Los gráficos y audio interactivos pueden presentar información adicional al espectador o pueden dar pie el espectador para contribución.
El decodificador puede proporcionar contribución de espectador u otras informaciones al proveedor de servicio de radiotransmisión vía una conexión de módem o cable.

[0006] Conforme a sus naturaleza agregada, sistemas de televisión interactiva proporcionan contenido en varios protocolos de comunicación diferentes que preferiblemente pueden ser entendidos por el cliente o espectador que recibe la información del controlador de red/proveedor de servicio de radiotransmisión.
Típicamente el cliente es un decodificador con un procesador poseyendo potencia de tratación y ancho de banda de comunicación limitados.

Traducción de los protocolos varios va más allá que el tratamiento limitado de capacidad disponible en el procesador de decodificador de señales digitales típico.
Además, existen fuentes múltiples que utilizan una multitud de herramientas de creación web para crear contenido.
Estas fuentes tienden a utilizar el lenguaje de marcas de hipertexto (HTLM) como un estándar con JavaScript introducido en las páginas HTLM.
JavaScript son típicamente interpretados.
Dispositivos de cliente típicamente poseen potencia de tratación y ancho de banda limitados, incapaz de interpretar y han de ejecutar un lenguaje interpretativo en una manera rápida y eficaz.
Así, hay una necesidad de una arquitectura de cliente y de servidor robusta, que elimina la necesidad para interpretación de JavaScript introducido en el código de HTLM de modo que contenido codificado de HTLM se puede visualizar por el cliente o procesador de decodificador de señales digitales sin requerir una cantidad desorbitada de potencia de tratamiento o ancho de banda de comunicación.

[0007] WO 01/0301 divulga un sistema y método para diversas plataformas de cliente de acoplamiento y servicios de información diversa sin requerir necesariamente módulos separadamente configurados para acoplar cada combinación de plataforma de cliente y servicio de información.

[0008] EP-A-0778522 divulga un ordenador que compila programas accionado por una parte de compilación e incluye unos completadores que, cuando la firma digital de la parte originante de un programa neutral de arquitectura ha sido verificada, compila el código de arquitectura del programa neutral del programa neutral de arquitectura en el código de programa específico de arquitectura en el lenguaje específico de arquitectura identificado por el recopilador de información en el programa neutral de arquitectura, y adjunta al código de programa específico de arquitectura una firma digital de la parte de compilación para generar un programa específico de arquitectura.

[0009] US-A-5432937 divulga un método que habilita liberación única de aplicaciones para arquitecturas múltiples y sistemas operativos y para proporcionar facilidad de uso de aplicaciones en múltiples ambientes de arquitectura.

[0010] Según la presente invención se proporciona un método para compilar un script para ejecución en un dispositivo de cliente en una configuración distribuida que comprende: recibir una página de HTLM que contiene al menos un script de un proveedor de servicio en un servidor; extraer el script de la página de HTLM en un filtro; compilar el script en un código compatible de cliente para ejecutar en un dispositivo de cliente; transmitir el script compilado al dispositivo de cliente; y ejecutar el script compilado en el dispositivo de cliente.

[0011] La presente invención también proporcionó un equipo para un equipo para compilar scripts para ejecución en un dispositivo de cliente en una configuración distribuida que comprende: una memoria de servidor para recibir una página de HTLM que contiene al menos un script de un proveedor de servicio en un servidor; un componente de extracción para extraer el script de la página de HTLM en un filtro; un componente de compilador para compilar el script en un código compatible de cliente para ejecución en un dispositivo de cliente; un enlace de comunicación para la transmisión del script compilado al dispositivo de cliente; y un dispositivo de cliente para la ejecutando del script compilado.

Resumen de la invención

[0012] La presente invención aborda las necesidades del entorno de televisión interactiva mencionadas anteriormente.
La presente invención proporciona un método y equipo que comprende software y hardware para aceptar y extraer un lenguaje interpretativo, tal como JavaScript de un lenguaje de contribución, tal como HTLM y compilar el lenguaje interpretativo para mostrar en un dispositivo de cliente.
Una página de JavaScript compilada común a una pluralidad de páginas es enviada solo una vez y referenciada como una página externa por páginas compiladas, reduciendo así la cantidad de datos a ser enviada y ancho de banda asociado.
El espectador o cliente puede ser un espectador a un (STB), un usuario de teléfono móvil, unos asistentes digitales, un pocket PC o cualquier otro dispositivo de recepción electrónica.

[0013] La presente invención proporciona clara ventaja sobre sistemas conocidos.
La presente invención permite extraer scripts de una página de HTLM y compilar los scripts en un servidor, para su ejecución en un dispositivo de cliente, lo que es una mejora sobre sistemas conocidos donde un navegador interpreta y ejecuta los scripts en la misma máquina.
La presente invención también reúne scripts para cada lenguaje, por ejemplo JavaScript o cualquier otro lenguaje de script, de un documento de HTLM en un único paquete, para paso al compilador, de modo que algunos o todos los scripts son compilados juntos.
Scripts se introducen en los documentos de HTLM de muchas formas.
Por ejemplo, entre un par de etiquetas <SCRIPT> y </SCRIPT>, o como valor de atributo para controladores de evento.

La presente invención analiza el documento de HTLM, agrega el script (para un lenguaje especifico, por ejemplo, JavaScript) en un paquete, y luego llama al compilador.
Algunas piezas de scripts (por ejemplo, scripts incluidos) se pueden compilar separadamente en módulos diferentes.
Todos módulos se envían al dispositivo de cliente para ejecución.
Esto permite, por ejemplo, que scripts compilados sean compartidos a través de páginas HTLM diferentes.
Esto significa que el compilador tiene una interfaz para compilar script incluido que quizá diferente del de compilar los scripts de documento principales.
Así, es más fácil y más simple crear un único módulo compilado.

[0014] El dispositivo de cliente sabe cómo enlazar y ejecutar piezas de scripts compilados, como si fuera un único script.
Scripts compilados pueden ser guardados, de modo que el compilado no ocurren si el script está ya en la caché.
La presente invención también proporciona un marcador temporal de compilación en los scripts para que código sea ejecutado solo en los navegadores de HTLM usual (es decir, nunca ejecutados en el dispositivo de cliente).
En sistemas previos permiten al script probar en el momento de ejecución qué navegador está funcionando (Internet Explorer, Netscape, etc).
En la presente invención, el compilador preferido reconoce una clave, por ejemplo, "JS2O".
Así, en una pieza de código como: "if (JS20) DO_A; else DO_B;" el compilador no compilará "DO_B" ya que sabe que "if (JS20)" es siempre verdad.
Así, toda pieza de código es en realidad compilado como "DO_A "; el compilador de script no compila código que es nunca ejecutado en el dispositivo de cliente, así reduciendo el tamaño de código compilado, y mejorando el tiempo de ejecución de código.
El compilador reemplaza algunas referencias de enlace tardías al compilar enlaces temporales para ejecución más rápida en el dispositivo de cliente.
Sistemas previos resuelven referencias en el momento de ejecución.
Por ejemplo, en el JavaScript, cuando se accede a una etiqueta "foo", el motor de script buscará el nombre "foo" en una cadena de ámbito.
Esto lleva tiempo por supuesto.
Aquí el compilador enlaza (en algunos casos) la etiqueta "foo" a una dirección específica para acceso rápido a tiempo de ejecución.
Por ejemplo, en JS2O, a (algunas) propiedades se asigna espacio de memoria como variables global regular en C lenguaje, y a (algunas) otras propiedades se asignan los intervalos en la pila como variables local regulares en C.
El compilador reemplaza algunas referencias de enlace tardías por valores absolutos para ejecución más rápida en el dispositivo de cliente.
En esta forma de realización en vez de enlazar a una dirección, el compilador reemplaza la etiqueta directamente por un valor.
Por ejemplo, en el JavaScript, la etiqueta "NaN" es una propiedad de un objeto.
Aquí el compilador reemplaza directamente referencias a "NaN" por su valor.

[0015] El compilador preferido porta una interfaz para definir clases nuevas de objetos.
Esta habilita al compilador a acceder y/o optimizar creación y manipulación de aquellos objetos, y permite el tiempo de ejecución a enlazar con el código para esta clase.
El código de clase se puede implementar como un módulo de TV abierta.
En una forma de realización preferida, H2O define una sintaxis de HTLM para declarar clases nuevas de objetos.
El compilador sostiene una interfaz para definir ejemplo predefinido/externo de objetos que habilita al compilador a acceder y/o optimizar manipulación de aquellos objetos, y permite al tiempo de ejecución a acceder aquellos objetos.
Aquellos objetos son creados fuera de la ejecución de script.
En una forma de realización preferida, H2O define una sintaxis de HTLM para declarar ejemplo predefinido/externo de objetos.

Breve descripción de los dibujos

[0016] Otros objetos y ventajas de la invención se harán aparentes al leer la siguiente descripción detallada y en referencia a los dibujos anexos donde:

La Figura 1 es una ilustración de una cabecera que proporciona contenido a un dispositivo de cliente;
La Figura 2 es una ilustración de una cabecera que proporciona contenido a un dispositivo de cliente;
La Figura 3 es una ilustración de una cabecera que proporciona contenido a un dispositivo de cliente;
La Figura 4 es una ilustración de una cabecera que proporciona contenido a un dispositivo de cliente;
La Figura 5 es un diagrama de arquitectura de transcodificador de HTLM H2O;
La Figura 6 es un diagrama de circulación de datos y procesador de navegador H2O;
La Figura 7 ilustra una interfaz entre H2O y JS20;
Figuras 8 ilustran los componentes de compilador JS2O;

La Figura 9 ilustra el compilador como un compilador de línea de mando JS20;
La Figura 10 ilustra ejecución del módulo de código principal compilado JS en el cliente;
La Figura 11 ilustra un segmento de datos preferidos; y
La Figura 12 ilustra un segmento de pila preferida.

[0017] Mientras la invención es susceptible a modificaciones varias y formas alternativas, formas de realización específicas de las mismas se muestran por medio de ejemplo en los dibujos y se describirán aquí en detalle.
Debe entenderse, no obstante, que los dibujos y descripción detallada de los mismos no se destinan a limitar la invención a la forma particular descrita, sino al contrario, la invención cubrirá todas las modificaciones, equivalentes y alternativas que entren en el espíritu y alcance de la presente invención tal y como se define por las reivindicaciones anexas.

Descripción detallada de una forma de realización preferida

[0018] Pasando ahora a figura 1, la plataforma de servicio 50 comprende un grupo de aplicaciones aproximadamente dividido en tres categorías, conversión de contenido 204, funciones control de transacción/negocios 206 y conversión de transporte 207.
La plataforma de servicio habilita a servicios 200 a interactuar con un cliente 212.
Los servicios 200 comunican a través de un enlace de comunicación 202 a la plataforma de servicio 50.
La plataforma de servicio 50 comunica sucesivamente con un cliente 212.
El cliente 212 puede ser un STB, unos asistentes digitales, un teléfono móvil, o cualquier otro dispositivo de comunicación capaz de comunicación con la plataforma de servicio a través de enlace de comunicación 230.
La conversión de contenido 204 y servicios de conversión de transporte 207 proporcionan el transporte y función de comunicación, y los servicios de función de negocio proporcionan las funciones de control de negocio.

[0019] Como se muestra en figura 2, funciones control de transacción/negocios 206 están distribuidas entre la plataforma de servicio y el cliente 212.
Por ejemplo, un cliente puede desempeñar algunas funciones de negocios (por ejemplo, implementar reglas de campaña publicitaria y filtros de publicidad/negocio para seleccionar publicidades vistas) y seleccionar contenido, que son más adecuados para el cliente 212 (por ejemplo, seleccionar una publicidad o programa que encaje al perfil de usuario).
Las funciones de la figura 2 se expanden en la figura 3.
Como se muestra en figura 3, las funciones de negocio 206 comprenden cuatro componentes funcionales mayores: director de servicio 238, director de espectador 240, director de transacción 242, y director de publicidad (Ad) 244.
Sigue ejemplo de un flujo de operación de alto nivel para una forma de realización preferida.

[0020] Haciendo referencia ahora a figura 3, un servicio 200 negocia con un controlador de red para ofrecer un servicio a abonados vía la plataforma de servicio del controlador de cabecera.
La red o controlador de cabecera usa el director de servicio 238 para grabar los servicios y las reglas de negocio negociado 222 (por ejemplo horario, requisitos de ancho de banda, acceso de servicio a información de espectador) asociadas al servicio.
El director de servicio 238 almacena datos de servicio 216 (por ejemplo dirección URL, contenido).
Basado en las reglas de negocio 222 y datos de servicio 216, director de servicio 238 comunica con la función de comunicación emisión 234 para recuperar el contenido del proveedor de contenido.

[0021] Cuando el contenido es recuperado del servicio 200, se puede procesar por la conversión de contenido 204 y filtros de contenido 224 para convertir el contenido en una forma adecuada para el dispositivo de cliente 212.
La función de emisión 234 convierte el contenido en una forma adecuada para el la red de emisión 234.
El cliente 212 recibe el contenido convertido por enlace de emisión 211.
Cliente 212 y servicio 200 interactúan vía enlace punto a punto 210 y función punto a punto 232, que son parte de conversión de transporte 207.
El servicio 200 puede comprender compras, audio/vídeo, juego, sondeo, publicidad, mensajería o cualquier otro servicio.

[0022] Cliente 212 comunica a través de enlace de comunicación punto a punto 232 a la plataforma de servicio 50 y servicio 200.
Equilibrador de carga 236 interactúa con las funciones de negocios 206 para determinar el reparto del peso óptimo entre el emisión 234 enlace de comunicación 211 y el enlace de comunicación 210 punto a punto 232.
Los agentes de negocios de plataforma 226 usan reglas de negocio 222 para controlar la interacción e intercambio de información entre el servicio 200 y el cliente 212.
Por ejemplo, el controlador de red puede elegir prevenir acceso de servicio 200 a información de usuario.
Servicio 200 preferiblemente paga un honorario basado en las reglas de negocio 222 y datos de servicio 216 para acceder a la información de usuario.

[0023] Director de espectador 240 almacena información de cliente/usuario en los datos de usuario 220.
Agentes de negocio de plataforma 226 controlan el flujo de información de espectador al servicio 200.
Director de transacción 242 registra información transaccional cambiada entre el servicio 200 y cliente 212.
Basado en las reglas de negocio 222 y los datos de usuario 220, director de publicidad 244 determina qué publicidades y qué tipo de publicidades serán presentados al cliente vía enlace de difusión 211 y enlace punto a punto 210.

[0024] La Figura 4 ilustra otro ejemplo de una implementación preferida de plataforma de servicio 50.
Servicios 200 proporcionan compras, chat, y otros servicios bien en Internet o en otra red o canal de comunicación accesible al controlador de red.
Usando la plataforma de servicio, el controlador de red accede a aquellos servicios.
Funciones de negocio 206, que comprende director de servicio 238, interactúa con director de carruseles 254 para recuperar contenido de un servicio 200.
El carrusel comprende un flujo de repetición de datos audio/vídeo/interactivos emitidos a clientes de plataforma de servicio 50.
Director de carruseles 254, director de transacción 242 y director de servicio 238 controlan la inserción de contenido y deleción del carrusel de emisión.
Contenido de servicio es recuperado, convertido en un formato adecuado de cliente por H2O 248.
H2O 248 es una implementación posible de conversión de contenido 204 y filtro de contenido 224.
H2O convierte contenido HTLM en contenido legible o plataforma de servicio/cliente.
El contenido convertido es formateado en un carrusel de datos y multiplexado por la unidad de transmisión abierta 256 para emisión al cliente 212.
Cliente 212 interactúa con los servicios y si es necesario comunica con la plataforma de servicio y los servicios 200.
Comunicación punto a punto va a través de puerta de enlace de servicio 246.
Puerta de enlace de servicio 246 ejecuta conversión de transporte para convertir el protocolo STB en unos agentes de negocio de plataforma de forma 226 y H2O248 esperan y entienden.
Equilibrador de carga 236 interactúa con funciones de negocio 206, director de carrusel 254, y puerta de enlace de servicio 246 para determinar la carga óptima entre el enlace de emisión 241 y el enlace de comunicación punto a punto 210.
Funciones de negocio 206 interactúan con los agentes de negocio de plataforma 226 para controlar acceso e intercambio de información entre los servicios 200 y cliente 212.

[0025] En una forma de realización preferida de la presente invención, H2O es una solución de cliente/servidor, que permite a desarrolladores de contenido de internet a construir aplicaciones de TV interactiva y servicios para controladores de red que manejan la plataforma de servicio.
La plataforma de servicio habilita acceso de espectador al mayor conjuntos de talento y contenido de internet puesto en el gran creciente mercado mundial de aplicaciones de TV interactiva.
El H20 proceso de servidor convierte contenido de internet (páginas HTLM, scripts ECMA, y formateo de página HTLM) en los activos de plataforma de servicio.
El proceso de cliente H2O hace los activos e interactúa con el cliente 212.
En un contexto de t-Commerce/E-Commerce, H2O habilita tiendas de comercio electrónico a utilizar herramientas web existentes para crear servicios de compras y para interrelacionarse con la plataforma de servicio preferido (controlador), usando protocolo de web de estándar.

[0026] H2O actúa como un proxy a la puerta de enlace de servicio y las herramientas de radiodifusión para convertir contenido web.
H2O recibe HTLM de tanto fuentes difusiones como en línea.
La presente invención habilita sitios web a usar sus servidores HTTP y servidores de aplicación actuales para generar contenido de TV interactiva.
En una forma de realización preferida, H2O convierte HTLM, JavaScript, y gráficos de internet en código compatible con el cliente, Ocode preferiblemente, un código de OpenTV basado en C que funciona en una máquina virtual en el decodificador de señales digitales.
Cualquier otro protocolo conocido o desarrollado puede también ser añadido a la funcionalidad de H2O.
H2O habilita la plataforma de servicio a comunicar con STB del cliente que no son capaces de navegador completo y para crear interfaces originales de usuario.
H2O habilita conexión de plataforma de servicio en cualquier motor de comercio que usa solo HTLM.
H20 es responsable de contenido de web de conversión tal como páginas HTLM, dibujos JPG, ficheros audio de onda, etc. en recursos aptos para clientes que pueden ser fácilmente utilizados con capacidad de tratamiento mínima y ancho de banda en el cliente.

[0027] El lado de servidor de H20 es un HTTP proxy, H2OS.

Para otros fines, ese se puede empaquetar como una herramienta DLL o de lote.
El lado de cliente de H20, una aplicación STB OCOD, es H2OC.
H2OC es construido encima de otros componentes de cliente de plataforma de servicio, como la biblioteca de puerta de enlace de servicio o la biblioteca de carga de carrusel.
H2O habilita que URL sean usados para referenciar documentos y servicios.
H2O habilita el rastreo en los ambientes de difusiones y en línea.
H2OS proporciona funcionalidad de proxy HTTP.
Aplicaciones de plataforma de servicio solicita un documento a través de H2O.
H2O recupera el documento, lo analiza, lo compila, y devuelve el documento al solicitante.
Esta funcionalidad H2O habilita uso del mismo motor para usos diferentes, en línea y emisión, facilita escalabilidad, y habilita uso flexible de H20.
El análisis depende del tipo de documento, por ejemplo, análisis H2O puede ser análisis de HTLM, una imagen GIF, o imágenes JPEG, etc. Para hacerlo expansible, H2O proporciona función a "plug-in" y utiliza nuevos filtros de terceras partes.

[0028] H2O porta etiquetas especiales que comprenden: control A/V, control de canal; control en la muestra de pantalla (OSD); y Triggers.
Etiquetas W3C portadas por H2O comprenden: posición controlada de elementos gráficos (x, y, z).
Bibliotecas Javascript comprenden matemáticas, DOM, y fecha.
El lado de cliente de H2O, H2OC, compone activos de gráficos en el cliente o STB.
H2O habilita muestra actualizada de una página vista de usuario ante la recepción.
H2OC utiliza bibliotecas (comunicación, carrusel, et al.) proporcionadas por otros componentes de plataforma de servicio.
H2O habilita actualizar una página en una actualización única, pero también proporciona una opción para elegir entre actualizaciones parciales conforme activos son cargados, en lugar de esperar a que todos o algunos activos se carguen.
H2O habilita conexión/desconexión dinámica de clases de tercera parte.

[0029] En el modo emisión, preferiblemente, se proporciona un objeto permanente global que es no esclarecido cuando se inicia una página nueva.
El objeto permanente mantiene contexto entre páginas.
Otros objetos de base proporcionados por la plataforma de servicio son hechos permanentes también en la transición (por ejemplo, control de estación, OSD).
Aparatos son métodos definidos por el cliente.
Aparatos se definen a través de un lenguaje de definición de interfaz para permitir creación de nuevos aparatos, modificación de aparatos y para permitir añadir métodos sin modificación del compilador 4000 de JS2O de la presente invención.

[0030] Ahora retornando a las figuras 5 y 6, una discusión de los componentes H2O principal sigue.
Transcodificador H2O 2001 convierte contribuciones 2062 de HTLM en transcodificaciones H2O que pueden ser eficaz e interactivamente visualizadas por un navegador H2O en un dispositivo de cliente, por ejemplo, un OpenTV STB.
Las transcodificaciones H2O de vistas de navegador H2O en forma limitada Dynamic HTML4.
El huésped, tal como HTTP proxy2003, invoca transcodificador H2O 2001.
Transcodificador H2O 2001 se interrelaciona con controlador MIME 2036 e información MIME 2028 para información de contenido en el tipo MIME no HTLM.
El compilador JS20 4000 es seguro para multihilo.
Transcodificador H2O invoca Controlador de javaScript H2O 2000 para coger scripts como contribución, coger scripts externos o clase URI, e invoca compilador JS2O y JavaScript pre-enlace.
El controlador JS 2000 procesa clases JS personalizadas 2070 y JS scripts compartidos 2066.

[0031] Yendo ahora a figura 6, el navegador H2O 2100 se interrelaciona con tiempo de ejecución de JavaScript JS20 2129 para soporte de JavaScript.
El transcodificador H2O funciona como una tarea, coge contenido MIME a través de HTTP proxy huésped, y contenido MIME de procesos come se solicita.
Preferiblemente una interfaz dispone del huésped de proxy para controlar mecanismo de guardado para mejor eficiencia.
El transcodificador H2O ejecuta sincrónicamente.

[0032] El transcodificador H2O sostiene un subconjunto seleccionado del HTML4.01 W3 estándar.
Transcodificador H2O sostiene un subconjunto de texto: párrafos, líneas, frases; subconjunto de lista; subconjunto de tabla y subconjunto de enlaces.
El transcodificador H2O sostiene un subconjunto de objeto e elementos de imagen.

El transcodificador H2O también sostiene un objeto de lado de la autoría de filtro (por ejemplo, objeto personalizado IE (Internet Explorer)) y sostiene objetos personalizados de tiempo de ejecución para el STB; tanto el lado de la autoría (PC) y desarrollo de clase de objeto personalizado STB; un subconjunto de mapa de imagen (lado del cliente, lado del servidor); un subconjunto de forma y controles de forma; elemento de script; y un subconjunto de JavaScript.

[0033] En el transcodificador H2O, los nodos de HTLM contienen información CSS computada una vez en los nodos del elemento.
Muy poco, si lo hay, CSS adicional es conservado.
Esto es fundamentalmente diferente de las reglas de cascada dinámica de CSS.
Estilo dinámico se limita ya que un cambio de estilo preferiblemente se aplica a solo un nodo.
Esto significa que para buscar a través de DOM una propiedad del estilo de un elemento particular y esperar que devuelva un valor válido, la propiedad de estilo (por ejemplo estilo= color: rojo) es explícitamente específico para el elemento dentro de un estilo en línea, o explícitamente creado en un código JavaScript para que emulación PC se comporte de forma similar.

[0034] La presente invención invoca una tarea de controlador de JavaScript H2O para procesar datos del elemento de script.
La tarea de controlador JavaScript coge el script URI, e invoca el compilador de JavaScript JS20.
JS2O devuelve un paquete de datos con el módulo de código de JavaScript compilado.
JS20 puede guardar scripts de JavaScript compilados y encuentra scripts de JavaScript similares repetidos e idénticos.

[0035] La presente invención genera transcodificaciones H2O usando recursos compatibles con cliente de modelos de datos internos del motor de programa analizador sintáctico con información de disposición y de estilo limitada, y códigos JavaScript.
Una especificación general de formato de transcodificación H2O comprende secciones de transcodificación de recursos compatibles de cliente que representan el documento y códigos JavaScript para el cliente o STB.
El formato de transcodificación contiene información acerca del documento en la estructura de ramificación.
El formato es eficaz, compacto, y expansible.
El formato proporciona un número de versión para asegurar conformidad en el cliente o STB.

[0036] Un modelo de objeto compatible de cliente, por ejemplo, en una forma de realización preferida, el modelo de objeto OTV (OOM) se proporciona para que un programador cree clases personalizadas.
El OOM proporciona un mecanismo para desarrollar una clase de objeto personalizada, clase de carga/registro/cancelación de registro/descarga, y acceder a códigos de interfaz de clase para manipular objetos personalizados
El diseño e implementación de OOM usa el modelo de objeto JavaScript para evitar duplicación de esfuerzo.
El OOM proporciona una interfaz para permitir a JavaScript y C programas manipular objetos.
HTLM, OOM, y C programas también pueden funcionar sin JavaScript.
La presente invención proporciona registro ligero de clase de lado del client con esquema de nombración seleccionado.
El registro habilita asociar un módulo de clase con un nombre y consulta.
OOM carga un módulo de clase y registra el módulo.
Hay solo una interfaz para una clase.
La interfaz contiene solo métodos.
Cada método se asocia con un nombre de cadena único en este alcance de interfaz.
El OOM proporciona un mecanismo para crear un ejemplo de objeto de una clase y un mercanismp de referencoa de clase proporcionar.
OOM se encarga de la descarga de clase.
OOM define flujo de control y flujo de datos entre el H2O HTLM y el motor JavaScript.
OOM instancia objetos de navegador para el motor JavaScript.
OOM expone acceso de objetos DOM al motor JavaScript.
Un objeto de una clase es un objeto cuyas propiedades de método son los métodos de la interfaz de clase.
El OOM define interfaces, pautas, y un subconjunto del SDK API para que objetos personalizados sean integrado en la estructura de vista, actualización de vista, manipulación de evento, foco, y acceden para recursos de sistema compartidos.

[0037] Un controlador dirige comportamiento de navegador y componentes, y controla eventos.
El controlador proporciona respuesta de tiempo de puesta en marcha rápida, conseguido por técnicas varias tal como muestra visual durante el inicio.
El controlador preferido de la presente invención usa recursos de sistema: evento, descarga, mpeg, osd A/V, etc. y controla modelo de datos y vista.
El controlador también se encarga de raw y procesa eventos de cliente (preferiblemente OpenTV) 2146.

El controlador se encarga de eventos de objeto primitivos, que a su vez generan un evento DOM 2140,2134.
La presente invención sostiene encargarse de evento bubbling DOM 2138 y usa nodo focalizado como un nodo de inicio.
Eventos DOM pueden ser manejados por los códigos de controlador de evento JavaScript 2128.

[0038] El controlador preferido pre-define clases del elemento y así es más eficaz que clases personalizadas.
El controlador usando recuperador URI obtiene contenido URI, hace HTTP: obtiene y publica (publicación de forma) peticiones, obtiene respuestas y proporciona funciones abiertas y atrás/adelante de documento.
En una forma de realización preferida, modelos de datos son principalmente tiempo de ejecución HTLM DOM, y otras categorías de modelos de datos de tiempo de ejecución.
HTLM DOM también contiene información de vista que refleja sus efectos destinados.

[0039] Vista soporta preferiblemente estilos de color diferentes, y trata la tabla de color como un recurso compartido.
En una forma de realización preferida, un conjunto limitado de controladores tipo MIME son proporcionados dependiendo en la configuración para ahorrar memoria.
Navegadores populares de HTLM para ambientes PC típicamente exponen objetos de huésped específicos de navegador para que tiempo de ejecución JavaScript se encargue de ellos.
La presente invención sostiene un subconjunto de objetos huésped de Internet Explorer de Microsoft 5 (IE5) para tiempo de ejecución JavaScript.

[0040] La presente invención invoca el controlador de tipo JavaScript H2O (JS controlador).
El controlador JS atraviesa la ramificación del elemento para recopilar código global, funciones, referencias de enlace a scripts externos, referencias de enlace de clase de objeto, y códigos huésped (de atributos de controlador de evento) JavaScript.
H2O invoca el controlador de JavaScript H2O y le pasa esta colección.
La tarea de controlador JavaScript coge script y contenido clase MIME, e invoca el compilador de JavaScript JS20 que devuelve un módulo de código compatible con cliente que contiene los códigos de JavaScript compilados.
La recogida de script lleva de vuelta controladores a códigos.
La función se invoca como mostrado en el apéndice de software.

[0041] El transcodificador atraviesa la ramificación de nodo del elemento y genera cliente transcodificaciones compatiblemente con cliente, por ejemplo, OTV H2O que utilizan el generador de recurso compatible de cliente interno.
La presente invención configura componentes, pantalla, puesta en marcha, etc. Configuraciones estáticas y dinámicas son estáticamente determinado para compilar tiempo.

[0042] Pasando ahora a las figuras 7 y 8, el compilador JS20 3904 se interrelaciona 3902 con el motor H2O 3900 para compilar código de JS en un módulo de código compatible de cliente 3906, preferiblemente, un módulo de ocode OpenTV.
El compilador JS 4000 se interrelaciona con el motor H2O que utiliza un C/C++ API como descrito por debajo.
El compilador JS20 es reentrante y se divide en los componentes siguientes: un compilador del elemento de página JS20 4002, una caché JS2O 4004, una interfaz JS20 4006 y un JS2O extremo posterior 4008 ensamblador binario de ocode 4010 y generador de módulo de código 4012.
Todos los componentes de compilador son reentrantes y usan búfers de memoria para paso de datos de uno a otro.

[0043] Pasando ahora a figura 8 y también refiriendo a figura 5, un filtro H2O 2064 extrae Javascripts entrantes de páginas HTLM entrantes 2062 y las manda al compilador de elementos de página 4002 para agregación dentro de una página de muestra única.
H2O recibe páginas HTLM de tanto el enlace difusión como de conexión en línea.
El compilador de elementos de página 4002 recibe elementos JS de H2O 2030 (que vienen de una página HTLM), reúne internamente todos elementos para una página HTLM en un bloque único de código JS, y luego llama al JS compilador 4000 para compilar la página JS en un módulo de código.
Una página JS es la agregación de todos código JS de una o más páginas HTLM.
La interfaz 3902,3906 entre H2O y el compilador del elemento de página JS20 es descrito por debajo.

[0044] La caché opcional JS20 4004 recibe una página JS, y compara esa página con otras páginas compiladas almacenadas en caché.
Si almacenada, la página almacenada es usada.
Si no almacenada ya, JS20 llama al compilador JS 4006.
Almacenamiento en caché es preferiblemente realizado para las últimas 100 páginas compiladas.
Todos los hilos de compilador JS20 comparten la caché, así, el acceso a la caché es protegido.
La interfaz de compilador JS 4006 compila el código JS primero en un formato intermediario, y luego en un módulo de código compatible con cliente que utiliza el generador de código específico 4012 para JS20.

[0045] El extremo posterior Compilador JS 4008 recibe el la página JS completa como un árbol pre-analizado y pre-compilado.
El extremo posterior de compilador JS genera los códigos de operación compatibles con cliente, por ejemplo códigos de operación de ocode, directamente en el formato binario usa la salida del ensamblador binario 4010.
El extremo posterior de compilador JS genera un módulo de código compatible con cliente en el generador de módulo de código 4012.
El ensamblador binario compatible con cliente 4010, por ejemplo, el ensamblador binario Ocode (específico a OTV) proporciona un API para generar códigos de operación binarios compatibles de cliente.
Este ensamblador sostiene los códigos de operación necesitados por el compilador JS.

[0046] La presente invención también crea, cuando una opción es especificada, un fichero de ensamblaje de fuente.
Este fichero contiene toda la información de debug (.stabs) y el mismo código/datos exacto que el compilador binario.
Este fichero de ensamblaje se puede compilar separadamente para aquellos que quieren debug páginas JS de código de fuente.
El generador de módulo de código 4012 proporciona un API para contribuir los códigos de operación binarios y sacan un módulo de código compatible con cliente 4102.

[0047] Preferiblemente, la presente invención también genera una cabecera de módulo compatible con cliente (en la memoria) para sostener Meta información.
El compilador JS20 habilita prueba y aceptación del compilador JS, y también habilita prueba del generador de ocode compatible con el cliente y del compilador de elementos de página 4002.
Pasando ahora a figura 9, en un modo de operación, el compilador de línea de comando JS20 4100 funciona como una simple interfaz de línea de mando que toma un fichero JS de fuente como contribución y emite un fichero de ensamblaje de Ocode compatible con cliente 4112.
El compilador JS20 puede también opcionalmente contribuir una definición de clase de objeto externo.

[0048] El interfaz de línea de comando 4104 comunica con el compilador de elementos de página JS2O 4004 con el mismo API que H2O, es decir, (js2o_compile... API como definido en JS20 FFS).
Las opciones siguientes son disponibles para el compilador de línea de comandos JS20.
Js2o -g -O -i inputfile –o outputfile -c classmodule -p name classmodule
-i : input file, -o : output file, -g: genera información de debug, -O: optimizar,
-c: define clase de objeto externo, "classmodule" es un módulo de código compatible con cliente con información de cabecera especial, -p: define un objeto predefinido llamado "name" (nombre) y de clase definida por el módulo de código "classmodule".

[0049] El fichero de salida se compila con un compilador compatible con cliente, por ejemplo, un ensamblador de ocode del OpenTV Software Development Kit (SDK#) o con un interfaz de compilador C (por ejemplo, gcco).
El ensamblador binario de Ocode y el generador de módulo de código funcionan en el mismo formato de objeto binario.
Las herramientas SDK proporcionan un formato ' a.out' para los ficheros de objeto de Ocode.

[0050] El tiempo de ejecución H2O 4202 y tiempo de ejecución JS2O 4204 pueden ser parte del mismo módulo de código compatible con cliente.
En una forma de realización preferida, los módulos de clase predefinida 4200 son el código para las clases de objeto OOM incorporado y DOM.
El tiempo de ejecución JS2O incluye las clases incorporadas JS.
El compilador JS2O 4000 genera el módulo de código principal JS 4206 y el módulo de código externo JS 4208.
Módulo de clase externa 4210 y otros módulos son preferiblemente módulos C.

[0051] JS2O proporciona un cliente compatible, por ejemplo, biblioteca de Ocode que se enlaza con todos los módulos de código compilado JS.
Esta biblioteca está compuesta de funciones pequeñas.
Funciones grandes se almacenan en el módulo de tiempo de ejecución JS20.
Por ejemplo una función "add two integers" (añadir dos enteros) está en la biblioteca de Ocode, pero una función "generic add" (añadir genérico) de cualquier tipo está en el módulo JS de tiempo de ejecución.

[0052] Todos valores empujados en la pila preferiblemente son tipo js2o para alojar recogida de inmundicia.
Es posible en algunos casos empujar algunos valores non-js2o cuando es seguro que aquellos valores surgen antes de cualquier recogida de inmundicia pueda ocurrir.
Esto pasa cuando el compilador emite un conjunto de instrucciones atómicas, como mostrados en el apéndice de software.

Recogida de inmundicia

[0053] El colector de inmundicia (GC) administra asignación de memoria para objetos y cadenas dinámicos JS2O.
El GC comienza de cero cada vez que una página nueva es comenzada (toda la memoria previamente asignada es preferiblemente descartada).
El GC usa una marca y algoritmo de barrido.
El valor de marca aumenta en uno cada vez que una recogida de inmundicia nueva es comenzada.
Así, GC hace no necesita un segundo paso para vaciar las marcas.

[0054] El motor de tiempo de ejecución JS20 crea objetos y cadenas dinámicamente.
Esos son los que son recogidos como inmundicia.
Los objetos dinámicamente creados y cadenas se referencian a través de variables (locales o globales), a través de valores temporales en la pila, y a través de propiedades de objeto.
La marca y algoritmo de barrido tiene que ir a través de todos ellos para marcar los artículos usados.

[0055] Objetos persistentes de página son aquellos que son mantenidos vivos a través de páginas múltiples.
Objetos persistentes de página preferiblemente no sostienen una referencia a un valor JS (objeto o cadena u otro) para evitar problemas cuando se cambian las páginas cuando los módulos son desenlazados.
Los objetos persistentes de página preferiblemente hacen copia propia de datos.

[0056] Pasando ahora a figura 11 y figura 12, variables globales JS usadas en una página y objetos referenciada en una página (objetos aún predefinidos) son asignadas (en momento de compilar) a una ranura en el segmento de datos 4300 del módulo de código JS principal de código.
La variables local y parámetros tienen cada uno una ranura en la pila de ejecución.
El compilador también genera valores intermedios o "temp values" en la pila.
El algoritmo de marca va a través de todos aquellos intervalos y marca todos objetos dinámicos y cadenas, al igual que recursivamente marca todos objetos y cadenas referenciados en propiedades de cualquier objeto.

[0057] El basurero proporciona un API, por ejemplo,
void js2o_mark_val (js2o_val_jsval, int markvalue);
void js2o_gc_mark (int *js2o_fp, int *js2o_sp, int *js2o_first_fp, int markvalue).

[0058] La función js2o_mark_val marca el valor js2o con el valor especifico.
Esta función lleva cuidado de llamar el objeto MarkProperties vtable it función si el valor es un objeto, y lleva cuidado de marcar el valor de prototipo, si lo hay.
La función js2o_gc_mark va a través de todos los globales JS y todas las pilas de llamadas JS y marca todos valores temporales, locales, argumentos y globales, como se muestra en figura 12.
El GC también verifica intervalos 'número de args', pero no hace una diferencia en GC ya que son de tipo js2o_int.

[0059] La presente invención proporciona es un API genérico para memorizar propiedades dinámicas en una cadena única.
El API incluye el nombre de propiedad y índice de propiedad en esa cadena para acceso rápido a ambos.
Este código asume un número máximo de 255 propiedades, y una longitud de nombre de propiedad máxima de 127 bytes (no incluyendo el último carácter cero).

[0060] Cada nombre de propiedad se almacena dentro de una cadena única en el formato, "N-name/" donde N es un único byte, que sostiene el número de propiedad característica +1.
Name (nombre) es el propio nombre de propiedad.
Este formato asume que ' /' y ' -' son caracteres ilegales para nombres.
N contiene un índice de 1 a 255.
Los caracteres "/" y "-" se pueden sustituir por cualquier carácter no usado en nombres y números válidos.

[0061] La siguiente función establece un nombre de propiedad a un valor específico.
Si la propiedad no existe, es añadida.
void prop_set (js20_prop* prop, char *name, js2o_val val);

[0062] La siguiente función obtiene una propiedad, especificando el nombre.
Si la propiedad no existe, devuelve JS2O_NULL.
Js2o _val prop_get (Js2o_prop* prop, char *name);

[0063] La siguiente función retira una propiedad, especificando el nombre.
Si la propiedad no existe, nada ocurre.
Void prop_remove {js2o_prop* prop, char *name);

[0064] La siguiente función obtiene el nombre de una propiedad, especificando un índice. (0 a n). La función devuelve JS2O_NULL si este índice no existe.
Esta función asume que se llama con número de índice consecutivo, empezando por 0.
js2o-val prop-index_get_name (js2o_prop* prop, int index).

[0065] La siguiente función marca todas propiedades con el valor de marca específico, usado para recogida de inmundicia.
void prop_mark (js2o_prop* prop, int markvalue);

[0066] La siguiente función libera toda memoria usada internamente por esta lista de propiedad.
void prop_free (js2o_prop* prop).

[0067] El tiempo de ejecución js2o proporciona los siguientes principios activos:
js2o_obj *js2o_funcobj_create (js2o_obj *this, void *func);

[0068] Esta función crea un objeto de función con el cursor específico 'this' y dirección de función.
Cada vez que la función es llamada, el eel cursor específico 'this' se pasa como el primer parámetro.
void js2o_funcobj_delete (js2o_obj *objf);

[0069] Esta función elimina un objeto de función.
En realidad no lo borra.
La eliminación será hecha por la recogida de inmundicia más tarde si nadie referencia este objeto.
La función restaura el objeto a valores nulos, de modo que es seguro acceder este objeto (genera opcionalmente advertencia de tiempo de ejecución o error, pero no triturará de prueba a ejecutar código inexistente).
void js2o_to_null_object (js2o_obj *obj, size_t size);

[0070] Esta función se utiliza para cambiar un objeto existente en un objeto de tipo JS2O_NULL_OBJECT.
El tamaño asegura que el objeto original es suficientemente grande para ser transformado en un objeto nulo.
Esta función también puede usarse para 'quitar' de forma segura un objeto, que se puede referenciar por alguna variables.
js2o_val js2o_dstr_create (char *str);

[0071] Esta función crea un ipo de cadena JS2O con la cadena carbonizan específica.
La cadena es copiada aquí.
js2b_val js2o_dstr_create_static (char *str);

[0072] Esta función crea un tipo de cadena JS20 con el car de cadena específicado.
La cadena no es copiada aquí.
Se supone que el puntero str es válido.

[0073] El lenguaje JavaScript en JS20 es un subconjunto de Lenguaje de ECMAScript Language ECMA-262.
Algunas características no soportadas en JS20 son aquellas que requeriría compilación en el dispositivo de cliente.
Otras consideraciones de soporte incluidas son velocidad de tiempo de ejecución, incluyendo posibilidades de optimización de compilador, tamaño de tiempo de ejecución, y utilidad en un entorno del dispositivo de STB/cliente.
Las siguientes características JS no se soportan en JS20 en el presente ejemplo de una forma de realización preferida.

[0074] Clase RegExp, función Eval, definición de funciones anidadas, objeto de llamada, objeto de argumento, instrucción "with", constructores de función con argumentos dinámicos o cuerpo, métodos Watch/unwatch, _parent_and _proto_ (características de navegador), ImplicitThis atributo, ImplicitParents atributo, instrucciones Try/Catch/Throw (intentar, coger, lanzar) y todos objetos de error, otros métodos seleccionados y propiedades del predefinido objetos JS.

[0075] En una forma de realización preferida, las siguientes condiciones se implementan en JS20, no obstante, parámetros diferentes o valores de limitación se pueden seleccionar como se desee.
Longitud máxima de nombre de propiedad: 127 bytes (no incluyendo último carácter nulo); número máximo de propiedades por un único objeto: 255; valores de número entero son firmados 29 bits (-268435456, #+268435455); y valores flotantes son 31 bits.

[0076] El valor de regreso de unos constructores (si hay) es descartado si los constructores no devuelven un objeto diferente.
Objetos dinámicos no ensombrecen objetos predefinidos.
El nombre de aquellos objetos predefinidos son palabras reservadas.
Funciones estáticamente definidas no pueden ser ensombrecidas por otras funciones (lo que significa que aquellos métodos son propiedades de solo lectura del objeto global).
Un conjunto limitado onomástico es reservado.
Todos nombres que empiezan por JS20, con o sin prefijo de guiones bajos son reservados.
Algunas funciones no será accesible como objetos JS, por ejemplo, funciones globales predefinidas.

[0077] La ejecución de tiempo de ejecución de un script JS20 tiene comportamiento preferido:
Manipulación de excepción: un error de tiempo de ejecución puede bien hacer nada, o parar la ejecución JS, o incluso paran la prestación de página.
Rebosamiento de computación: el tiempo de ejecución no prueba el rebosamiento de número entero.

[0078] Los siguientes objetos JS se soportan por JS2O.
Objeto; Matemáticas; Cadena; Número; Booleana; Colección; y Función. (Object; Math; String; Number; Boolean; Array; and Function).
JS20 incluye soporte para objetos específicos compatibles de cliente predefinido, como OSD, canales, etc. El objeto de ventana tiene propiedades "event", "document" y "navigator" que son objetos en sí mismas.
El objeto de ventana tiene muchos métodos (por ejemplo, back 0).
Todas propiedades y métodos del objeto de ventana son directamente accesibles.

[0079] El H2O (HTLM para código compatible con cliente, por ejemplo, Ocode) del motor convierte Páginas HTLM múltiple para formato compatible con cliente, por ejemplo, formato OpenTV en paralelo.
Tener un compilador JS reentrante simplifica la interfaz y mejora el rendimiento.
El compilador es reentrante y para desempeñar tantas optimizaciones como posible, el compilador compila todo el código JS de una única página HTLM en una compilación.

[0080] Para desempeñar la optimización para JS2O y otras funciones, el compilador proporciona las siguientes características: el compilador da mensajes de error específicos para características no soportadas and.limitations, y analiza el fichero entero antes de empezar la generación de código (necesitado en particular para función llamada antes de su definición).
Una referencia de compilador a una función diferencia preferiblemente entre funciones definidas en la actual compilación, funciones definidas globalmente y otras.

[0081] Objetos de función: para diferenciar como para acceder al objeto o una llamada a la función, referencia a un variable se diferencia en cuanto a variables locales, variables globales, y otra variables.
El compilador habilita determinación en cuanto a si un nombre se refiere a una función conocida, o una variable conocida.
Una opción debug sirve preferiblemente para encender o apagar mensajes de debug.
Opciones de optimización se proporcionan para encender o apagar varias optimizaciones compiladoras.
El compilador emite advertencias si a una llamada de función le faltan algunos argumentos.

[0082] Tipos variables: en algunos casos, el compilador sabe el tipo de una variable que está usada.
El compilador mantiene el tipo de variables en el código.
El tipo puede también ser ' Unknown' (desconocido).
El compilador mantiene Meta datos (fichero de fuente, número de línea, etc) de modo que el generador de código puede sacar información de debug para programa y datos.
En particular, el compilador proporciona acceso estáticamente compilado lo siguiente: variables locales, variables globales (definidas en la compilación corriente), funciones (definidas en la compilación actual), y objetos predefinidos y sus propiedades.
El compilador proporciona una interfaz para importar definición de objetos predefinidos.
Nótese que como no hay 'Block Scope' dentro de un función JS (toda variables local definida en una función son válidas en toda la función, independietemente de dónde son declaradas), el compilador reserva espacio para todas la variables locales dentro del prólogo de función.
Por defecto todas las variables tienen el valor JS2O_UNDEFINED.
El prólogo de compilador establece el valor de todas las variables locales.
Todos variables globales 'uninitialized' (sin inicializar) se fijan a ese valor.

[0083] El compilador JS20 proporciona un API con el analizador H2O, como se muestra en el apéndice de software.

[0084] Este API es seguro multihilo.
El orden en que aquellos API son llamados es obviamente importante a asegurar generación de código ordenadamente.

[0085] El método "js2o_compile_create" crea un controlador de compilación; filename es el nombre de fichero de fuente de JavaScript.
Este filename se usa solo para mensajes de error.
Opciones compiladas son para el compilador (por ejemplo, optimización, fichero externo, debug, etc).
La función devuelve NULL si hay error.
El método "js2o_compile_destroy" destruye el controlador y libera toda memoria relacionada.
El método "js2o_compile_generate" ejecuta la compilación real de todas las piezas registradas en un trozo compilada.
Después de esto, solo js2o_compile_error_msg o js2o_compile_destroy deberían ser llamados.
Los datos compilados preferiblemente comprenden un módulo de código OPENTV (en formato binario).
El método "js2o_compile_error_msg" devuelve el último mensaje de error, adecuado para un fprintf en el stderr.
El mensaje de error es liberado después de que una llamada a js2o_compile_destroy.

[0086] Método "js2o_compile_warning_callback" registra una función de devolución de llamada para mensajes de advertencia, 'callback' es un puntero a una función de devolución de llamada, y 'datos' es un valor, que será pasado a la función de devolución de llamada.
El prototipo para la función de devolución de llamada es oid func (void *data, char *msg, js2o_handle jh)'.
El mensaje de advertencia es destruido después de la llamada de devolución de llamada.
Registro una devolución de llamada nula para eliminar una devolución de llamada.
Por defecto, no hay función de devolución de llamada.

[0087] El método "js2o_compile_add_src" añade una pieza de código de texto a la compilación, "linenum" es el número de línea de la primera línea de ruta este código, y asumimos que es del fichero especificado injs2o_compile_create.
El parámetro de control se usa para código controlador de eventos: en ese caso el control es un puntero usado para devolver un valor de control.
Este valor de control es preferiblemente usado en el tiempo de ejecución para ejecutar este código (véase también js2o_execute).
En otros casos, es decir, no un controlador de evento, el control es nulo.
Esta función mantiene una copia interna del código.

[0088] Optimización: para controladores de eventos múltiples que usan el mismo código, js2o genera un ejemplo del código de controlador.
El código preferiblemente tiene todas las líneas nuevas (\n caracteres) del fichero de fuente original, de modo que números de línea son significativos.

[0089] El método "js2o_compile_add_bin" incluye un fichero de js2o pre-compilado.
Cualquier instrucción global en el fichero precompilado se ejecutan a esta posición en el tiempo de ejecución.
El nombre es usado por JS20 para localizar el módulo externo en el tiempo de ejecución.
Si el mismo fichero se incluye en diferentes posiciona en una página HTLM, este API es preferiblemente llamado cada vez con lo mismo nombre.
El módulo señala al código compatible con cliente, por ejemplo, un módulo de código OpenTV (incluyendo cabecera).

[0090] El método "js2o_compile_add_obj" define un objeto predefinido disponible. "objname" es el nombre del objeto.
La clase se define por el código compatible con cliente, por ejemplo, módulo .otv (incluyendo cabecera) señalado por "classmodule".
El módulo no se referencia en el tiempo de ejecución por JS20 (el objeto se crea por H2O).
Nótese que esta API no define una clase nueva accesible de código de fuente JS.

[0091] El método "js2o_compile_add_class" define una clase predefinida disponible.
La clase se define por el código compatible con cliente, por ejemplo, módulo .otv señalado por classmodule.
El nombre de clase (a ser usada en la 'nueva' instrucción JS) es también específico dentro del módulo (módulo cabecera).
El nombre de clase se usa en el tiempo de ejecución por JS20 para obtener los constructores de función de esta clase.

[0092] El método "js2o_compile_add_element" define un elemento DOM nuevo.
Este añadirá una variable global con el nombre específico.
La función devuelve un control variable.
Este control se usa en el tiempo de ejecución por H2O para inicializar la dirección de este objeto.

Un fichero de fuente de JavaScript externo se puede incluir en la página HTLM.
H2O compila este fichero de fuente JS antes de compilar la propia página HTLM. Lo que H2O finalmente pasa en el js2o_compile_add_bin API (para compilar el código JS dentro de la página HTLM) es el ya compilado fichero JS.
Para compilar un fichero de fuente externo, el motor H2O llama al js2o_compile_API con la siguiente advertencia: una opción especial 'external ref' (referencia externa) se usa en js2o_compile_create.
El API no incluye un fichero externo (no llama a js2o_compile_add_bin).

[0093] El compilador JS20 preferiblemente mantiene una caché de páginas JS previamente compiladas (en la memoria), por ejemplo, las últimas 100 páginas.
Almacenamiento en caché es realizado dentro de JS2O porque algunas páginas tendrán HTLM diferente pero el mismo script JS HTL introducido dentro de HTML.
Así la presente invención guarda la compilación de tal código JS dentro de JS20.
Aquí un página JS es simplemente la agregación de todos los código JS de una página HTLM.
Nótese que el proxy http dentro de H2O también implementa el almacenamiento en caché.
Almacenamiento en caché de una página JS habilita uso de un JS común entre una pluralidad de páginasmuestra.
Esto reduce la cantidad de datos requeridos para ser enviada al STB o cliente.

[0094] Código JS se encapsula dentro de un módulo de código compatible con cliente, por ejemplo, un Módulo de código de OpenTV.
En el caso de HTML+JS, el módulo de código compatible con cliente, por ejemplo, módulo de código de Open TV es preferiblemente introducido dentro del recurso H2O (como una colección "big char").
El formato de módulo de código habilita depuración de código de fuente.
El compilador emite la información de debug en un fichero .odb.
Nótese no obstante que el debugger no conoce los tipos JS20.
Preferiblemente soporte se proporciona en gdbo para los tipos JS20.
Código JavaScript se usa dentro de una página HTLM en diferentes formas: código JavaScript se puede introducir dentro de página HTLM usando uno o varios par de etiquetas <SCRIPT></SCRIPT>.
La sintaxis es preferiblemente: <SCRIPT LANGUAGE="JavaScript"> any JavaScript statements...</SCRIPT>

[0095] Todo código en línea y controladores de eventos de una página se compilan en un único cliente principal compatible, por ejemplo, módulo de Open TV.
H2O y JS20 sostienen la referencia a un fichero de JavaScript externo.
En H2O, un fichero JS externo de fuente es compilado, almacenado y cargado separadamente para mejorar rendimiento.
Esto habilita envío de una única pieza de código, aunque este fichero se usa en muchas páginas.
Se permite incluir el mismo fichero en una página de HTLM única.
J20 precompila los ficheros JS externos antes de compilar el principal código JS.
Un fichero externo se compila en un cliente DLL compatible, por ejemplo, módulo de Open TV.
La Sintaxis de HTLM correspondiente es: <SCRIPT SRC="URI" LANGUAGE="JavaScript"></SCRIPT>, donde "URI" señala a un fichero de fuente JavaScript.

[0096] Un fichero de fuente JavaScript externo puede contener cualquier instrucción JS válida, no obstante, conflictos nombre con los demás módulos JS en funcionamiento son posibles.
Por ejemplo, múltiples instrucciones "vary x;" son problemáticos desde punto de vista de conflicto, pero múltiple "x=value are not" (no lo son).
Conexión de código global: el fichero se compila en un módulo.
El módulo exporta una función que contiene todo el código global.
Esta función se llama en tiempo de ejecución, como si el código externo estuviera "en línea."

[0097] Conexión de variables: el fichero precompilado preferiblemente exporta todas las variables globales usadas por su código, bien externo (por ejemplo, y =5), o interno (por ejemplo, var x = 4).
El compilado código JS principal reserva alguna ranura en sus variables globales para todas aquellas variables exportadas.
El código precompilado también incluye intervalos para su variables globales exportadas, pero aquellos intervalos en realidad mantienen punteros a las variables reales en el código principal.
Las direcciones se inicializan a tiempo de ejecución.

[0098] Conexión de módulo: en el tiempo de ejecución, H2O carga el módulo precompilado, crea un contexto JS20, y luego inicializa el módulo precompilado (ver js2o_dll_init).
Esta inicialización ejecuta la actualización de direcciones de variables del principal código JS al módulo precompilado.
El nombre específico en jjs2o_compile_add_bin se utiliza para referenciar el módulo en el tiempo de ejecución.

[0099] H2O y JS2O soportan refrencia a un fichero precompilado externo.
La Sintaxis de hTLM correspondiente es:
<SCRIPT SRC="URI" LANGUAGE="Open TV"></SCRIPT>,
donde "URI" señala a un cliente compatible, por ejemplo, fichero JS de módulo de código OpenTV.

[0100] Este módulo de código define y soporta requisitos de JS2O específicos.
Conexión es lo mismo en cuanto a un fichero JS externo de fuente.
Un segmento de código JS puede ser específico como el valor de un atributo HTLM (por ejemplo; onClick) para ser ejecutado cuando ocurre un evento específico.
En la terminología ECMA, esto es un ' host code' (código huésped).
La sintaxis es:
OnAttribute=" any JavaScript statements"

[0101] Todos los códigos y controladores inlined de eventos de una página se compilan en un único módulo compatible con cliente principal, por ejemplo, Módulo de Open TV.
Una controlador de evento puede devolver valor de estado True, False u otro.
Para conexión, JS20, en tiempo de compilación, js2o_compild_add_sr devuelve un control a H2O para este controlador de evento.
Este control se usa en el tiempo de ejecución para referenciar el código de controlador de evento.
Código JS puede acceder a objetos compatibles con cliente, por ejemplo, objetos predefinidos de Open TV (incluyendo objetos DOM).
No hay declaración necesitada en las páginas HTLM.

[0102] Para conexión, en tiempo de ejecución, H2O preferiblemente pasa una colección de direcciones de objeto predefinido a JS.
La lista y orden de los objetos predefinidos en esa colección es predefinida (conocida por tanto H2O como JS20).
H2O preferiblemente también pasa una colección de funciones globales.
Aquellas funciones globales pueden implementar métodos para objetos predefinidos (por ejemplo, método setVisibility).

[0103] El siguiente ejemplo de sintaxis puede utilizarse para declarar una clase de objeto nueva para ser usada en el JavaScript.
El código JS puede usar 'nuevo' controlador JS para crear un ejemplo de esta clase, "<OBJECT CLASSID=otv _module_uri DECLARE> </OBJECT>"

[0104] En este ejemplo, el otv_module_uri señala a un módulo de código de Open TV.
Este módulo de código es preferiblemente conforme a la definición de módulo objeto de clase JS.
La capa H2O se asegura preferiblemente que este módulo .otv está cargado antes de iniciar cualquier script JS de esta página.

[0105] Para conexión, en el tiempo de ejecución JS20 pide a H2O la dirección de esto módulo, usando el nombre de clase.
El nombre de clase se incorpora en el módulo de código de Open TV (pasado en el tiempo de compilar a JS2O).
Véase también js2o_compile_add_class.
Un único ID identifica un elemento DOM. por ejemplo, <anchor id = "foo"... >,<
src="xxx.otv" id = "glop">.

[0106] Elementos DOM se acceden en código JS como una variable global regular que usa ese nombre ID.
En ambos casos, el objeto se crea por H2O, no JS20.
Conexión: en tiempo de compilación, H2O registra este elemento con JS20, que devuelve un control.
Este control se usa en el tiempo de ejecución por H2O para decirle a JS2O la dirección de este objeto.

[0107] JS20 proporciona sus propios tipos de datos (int, bool, float, objects, etc), que son los tipos C estándar.
El tipo C genérico para un valor JavaScript es ' js2o_val'.
Incluye todos los otros tipos JS: js2o_int: valor de número entero, js2o_float: valor de flotación, js2o_bool: valor booleano, y js2o_ptr: puntero a objetos, cadenas, etc.

[0108] Para ejecución rápida y eficaz, la presente invención proporciona una única palabra de 32-bits para representar todos tipos de datos JS20 en la pila.
Lo siguiente son los 6 tipos básicos para todas las variables JS20.
Cualquier valor JS20 es preferiblemente compatible con uno de los 6 tipos básicos.
Todos tipos pasados por valor encajan en este único dato de 32 bit, para simplicidad.

[0109] El formato de flotación preferido es: seeeeeeeeffffffffffffffffffffff con: 1-bit s es bit del signo. 0 signica positivo, 1 significa negativo. 8-bit e es campo de exponente.
La polarización de exponente es 127.
El 23- bit f es campo de fracción.
Este significa que la flotación preferida usa un campo de fracción de 22-bit, en vez de 23.
Macros se definen en el apéndice de software.

[0110] Todas las clases de objetos JS comparten estructura común.
Las definiciones de la estructura de objeto comienzan con los mismos campos, y todo el vtable de las clases comienzan con las mismas funciones primarias.
Para acceso rápido a los principios activos JS2O y los métodos, el compilador conoce la estructura de todos aparatos o al menos el principio de la estructura.
Debido al ocode preferido ' vcall' opcode, un un puntero de la tabla virtual es el primer campo de esa estructura.
El vtable mismo contiene la dirección de todas funciones obligatorias para cada aparato, posiblemente seguido de direcciones de funciones específicas a esa clase.
Tiempo de ejecución JS20 también necesita un puntero tipo, y un 'rototipo' en cada estructura de aparato.
Una definición de inicio para objetos JS20 si proporcionada en el apéndice de software.
La vtable señala a una colección de funciones obligatorias para cualquier clase de objeto JS20.
Las funciones obligatorias se proporcionan en el apéndice de software.

[0111] Aquellos principios activos son solo llamados por el subcomponentes de compilador JS en un contexto tipo C.
Algunos parámetros son tipos C (por ejemplo, int índice, char *name), algún son tipos JS20 (por ejemplo, 'this' (este) y valores de devolución).
Nótese que el orden de los parámetros refleja el orden usado por el compilador JS.

[0112] Las siguientes funciones vtable se proporcionan:
La función js2o_val GetProperty (js2o_obj *this, char *name) devuelve el valor de propiedad de la propiedad específica.
Devuelve JS2O_UNDEFINED si la propiedad existe, pero no tiene valor definido.
Devuelve JS2O_NULL si la propiedad no existe.
Esta función no mira por el puntero prototipo, esto lo hará automáticamente el motor de tiempo de ejecución JS2O, si es necesario.
Esta función puede asumir que la propiedad 'prototipo' la maneja directamente el compilador JS2O.
No obstante, las propiedades ValueOf y ToString las controla esta función.
Ver también API ValueOf y ToString por debajo.
Si la propiedad corresponde a un método, el valor de regreso es un tipo js2o de función.

[0113] Una clase de aparato puede decidir solo sostener algunos de los nombres de propiedad bien conocidos, y solo con un nombre literal.
En este caso, las propiedades se acceden con los API GetNumberProperty/SetNumberProperty y el API GetProperty devuelve JS2O_UNDEFINED.

[0114] Name (nombre) es preferiblemente un nombre de propiedad.
Si el objeto es una tipo colección, 'name' puede también ser el índice en la colección.
Es un índice si el nombre representa un válido número de cadena (por ejemplo, "1").
En el caso del objeto sostiene colección y el nombre es un índice, esta función devuelve el número de artículo correspondiente.

[0115] El vacío SetProperty (js2o_obj *this, char *name, js2o_val value) función establece la propiedad específica con valor.
Si la propiedad no existe, el aparato puede bien crearla (con aquel valor) o simplemente no hacer nada.
Esta función devuelve valor de NO.
Esta función puede asumir que la propiedad 'prototipo' se maneja directamente por el compilador JS20, no obstante, las propiedades ValueOf y ToString se manejan por esta función.

[0116] Name es normalmente un nombre de propiedad.
Si el objeto es una tipo colección, 'name' puede también ser el índice en la colección.
Es un índice si el nombre representa un número de cadena válido (por ejemplo, "1").
Por si el objeto sostiene colección y el nombre es un índice, esta función debería ajustar el número de artículo correspondiente, y actualizan su ' lengt' (longitud) propiedad (si apropiado).

[0117] La función vacía *GetMethod (js2o_obj *this, char *name) devuelve la dirección de función de este método.
Si esta propiedad no existe, o no contiene una función, una excepción de tiempo de ejecución es elevada (ver js2o_runtime_error).
El método devuelve un js2o_val valor.
El método se llama con los siguientes argumentos: js2o_obj *this, js2o_int nbarg, js2o_val arg1, ... js2o_val argN: Name es normalmente un nombre de propiedad.
Si el objeto es tipo colección, 'name' puede también ser el índice en la colección.
Es un índice si el nombre representa un número de cadena válido (por ejemplo, "1").
Por si el objeto sostiene colección y el nombre es un índice, esta función devuelve la dirección de función que corresponde con el número de artículo.

[0118] La función js2o_val GetIndexPropertyName (js2o_obj *this, int index) devuelve el nombre de la propiedad/método con aquel índice (0 a N).
Esta función devuelve JS20_NULL si la propiedad/método no existe.
De lo contrario devuelve un cadena de valor JS2O.
Esta función es principalmente usada para la instrucción JavaScript 'for/in' (para/en).
Esta función asume que se llama en una secuencia de índices: 0, 1, 2 ...

[0119] La función js2o_val ValueOf (js2o_obj *this) devuelve el valor de un objeto.
El valor de un objeto es específico de objeto.
El tipo devuelto puede ser un número, booleano, cadena, función o incluso objeto.
Este API es un atajo para GetProperty (this, "ValueOf) entonces una llamada a ese método.

[0120] La función js2o_str ToString (js2o_obj *this) devuelve la representación de cadena del valor de objeto.
Esta función es un atajo para GetProperty (this, 'ToString') luego una llamada a ese método.

[0121] La función vacía * GetCall (js2o_obj *this) devuelve la dirección de una función a ser ejecutada.
Esta se llama normalmente para objetos de función solo.
Llamarla para otros objetos se puede considerar un error de tiempo de ejecución.
Ver, js2o_runtime_error.
Para una función, este API es un atajo para Valueof() + get address of function (conseguir dirección de función).

[0122] La función vacía DeleteProperty (js2o_obj *this, char *name) elimina la propiedad específica.
Si la propiedad no existe o no puede ser eliminada, nada ocurre.

[0123] La función vacía MarkProperties (js2o_obj *this, int markvalue) se usa por el JS20 basurero para marcar todos los valores js2o_valreferenciados por este aparato (excepto el de campo 'prototipo' que es hecho automáticamente por el motor JS20).
En la mayoría de los casos, los valores js2o_val son simplemente el aquellos almacenados en las propiedades de aparato.
El aparato llama a la función js2o_mark_val para marca cada js2o_val.
No hacerlo puede resultar en que el js2o_val (todavía referenciado por el aparato) sea liberado.

[0124] La función js2o_val GetNumberProperty (js2o_obj *this, int property_number) es lo mismo que el GetProperty, excepto que un número de propiedad es específico en vez de un nombre de propiedad.
Este API se usa para objetos predefinidos con propiedades predefinidas.
El número de propiedad viene de una lista de nombres bien conocidos.

[0125] La función vacía SetNumberProperty (js2o_obj *this, int property_number, js2o_yal value) es lo mismo que el SetProperty, excepto que un número de propiedad es específico en vez de un nombre de propiedad.
Este API se usa para objetos predefinidos con propiedades predefinidas.
El número de propiedad viene de una lista de nombres bien conocidos.

[0126] La función vacía *GetNumberMethod (js2o_obj *this, int property_number) devuelve la dirección de función de este método.
Si esta propiedad no existe, o no contiene una función, una excepción de tiempo de ejecución es elevada (ver js2o_runtime_error).

[0127] La función vacía DeleteObject (js2o_obj *this) libera todos recursos internamente asignados por el objeto.
Este es opuesto de la función object_new de la clase de objeto.
Esta función libera el objeto mismo.

El basurero llama a esta función cuando un objeto es descubierto como siendo ya no usado.
Los métodos de aparatos son normalmente accedido a través del GetMethod API.
El aparato devuelve la dirección de función.

[0128] Cada método se ejecuta en un contexto JS20.
Cada método se implementa con las siguientes restricciones: cada parámetro de contribución es de tipo ' js2o_val', no regular C tipo.
Un valor de regreso es obligatorio (puede ser JS2O_NULL).
El valor de regreso también tiene que ser un tipo js2o, y cada método tiene al menos los siguiente primeros dos parámetros: puntero objeto 'this' y el número de parámetros pasados.
Por ejemplo, un método JS2O "foo" con dos parámetros X e Y se pueden declarar en C como: js2o_val foo (js2o_obj *this, js2o_int nbarg, js2o_val x, js2o_val y, ..);

[0129] Cuando se define un objeto externo o una clase de objeto externa, es posible declarar algunas propiedades predefinidas y métodos.

[0130] Un "vtable" método predefinido es definido por nombre y por un índice.
El índice es el número de función en el vtable de la clase.
Debido a funciones requeridas a principios de cualquier vtable, el primer índice disponible es 13.
El compilador optimizará, cuando sea posible, el acceso a los métodos predefinidos.
Cuando la definición de un objeto, por ejemplo, OSD, con una función predefinida, por ejemplo, muestra, el Código JS "OSD.show();" será optimizado (pero no "x.show(); " aunque x iguala OSD.).

[0131] JS2O+H2O define una lista de nombres de propiedad bien conocida.
Cada nombre se une con un número de cadena.
El compilador optimiza acceso a las propiedades bien conocidas llamando a los API del objeto GetNumberProperty y SetNumberProperty.
Si un método no se define como un 'predefined vtable method' (método vtable predefinido), esto puede todavía ser un de los nombres de propiedad bien conocida.
En este caso el compilador optimiza el método de obtención de dirección llamando el GetNumberMethod API.
Cuando se accede a dicho método como una propiedad, el compilador usa el GetNumberProperty y SetNumberProperty API.

[0132] Si un método puede también ser definido como un 'predefined global method' (método global predefinido).
El método es una función global, lo que implementa el método.
Esta función verifica que la clase de objeto es correcto para esto método, y luego ejecuta la funcionalidad real.
Un error es generado si el objeto pertenece a una clase incorrecta.
La lista de 'predefined global methods' es bien conocida compartida por todos componentes H2O.
La dirección de aquellos métodos se pasa a tiempo de ejecución por H2O (colección única de direcciones).

[0133] Hay una lista de nombres de propiedad bien conocida.
Esta lista se conoce por el compilador y por los objetos.
Esta lista incluye la mayoría (si no todos) de los nombres (y método) de la propiedad de los objetos predefinidos (incluyendo objetos DOM).
Esta lista solo se usa para métodos y propiedades accedidos por el compilador, pero no para otras cadenas.

[0134] Para optimizar el acceso a propiedades CSS, un nombre de propiedad bien conocida puede incluir el carácter '.'.
Por ejemplo, "style.color" se puede declarar como un único nombre de propiedad.
Tiempo de ejecución JS2O incluye recogida de inmundicia para objetos dinámicamente creados y cadenas dinámicamente creadas.
Un mecanismo Mark & Sweep se implementa para recogida de inmundicia.
Este es menos costoso en cuanto a velocidad de tiempo de ejecución que Reference Counters.

[0135] Recogida de inmundicia preferiblemente también hace un seguimiento de módulos (definición de clase, módulos de código externo) referenciados por las cadenas y objetos (por ejemplo, vtable, funciones, cadena estática, etc).
Objetos persistentes proporcionarán funciones de restablecimiento para limpieza general referencias externas durante el cambio de páginas (es decir, módulos de descarga).

[0136] H2O proporciona un director de memoria, por ejemplo para almacenamiento en caché de módulos.
JS20 y su basurero trabajan mano a mano con H2O.
JS20 asigna muchos pequeños trozos (objetos; cadenas), mientras H2O asigna unos pocos trozos mayores.

JS2O proporciona una función libre para que H2O sea ejecutado mientras nada ocurre (ninguna contribución de usuario).
Esta función vacía llama la recogida de inmundicia.

[0137] Nombres de funciones: acceso a funciones definidos en el mismo fichero se optimiza en la llamada directa a la dirección de función.
Estos medios en particular, la función no se puede cambiar dinámicamente.
Variables locales se compilan en intervalos directos en la pila.
Variables globales se compilan en las variables directas en la memoria de sección de datos.
Todos objetos predefinidos son parte del alcance global JS.
Objetos predefinidos son referenciados directamente, al igual que sus métodos predefinidos y propiedades.
El objeto de ventana y todos sus propiedades/métodos son directamente parte del alcance global.
Por ejemplo, propiedad "window.document" se puede acceder simplemente por "document".

[0138] El principal código JS y todos controladores de eventos JS (para una página) se compilan en un único módulo de código.
La capa H2O llama a una función de inicialización de esto módulo.
Esta función de inicialización devuelve un puntero de contexto JS2O.

[0139] js2o_cx *js2o_cx_create (void *module, int **DOMObjectHandle, js2o_obj **DOMObjects, js2o_obj **PredefinedObjects, void **GlobalMethods);

[0140] Los parámetros son un puntero al módulo principal e punteros para objetos DOM creados (controles y direcciones), un puntero para una colección de direcciones de objeto predefinido y un puntero para una colección de funciones globales predefinidas.
Los controles de objeto DOM son los controles devueltos por js2o_compile_add_element.
Esta función devuelve NULL si error.

[0141] El parámetro 'module' (módulo) preferiblemente señala a los mismos datos como el devuelto por js2o_compile_generate.
JS20 no hace una copia de estos datos, así que preferiblemente permanece válido hasta que la función destroy (destruir) lo retira.
Nótese también que estos datos son de solo lectura (JS2O no escribe en esos datos).
Nótese que al menos el objeto "window" es definido.

[0142] El contexto JS2O es destruido (cuando ya no se necesita) al llamar js2o_cx_destroy: void js2o_cx_destroy (js2o_cx *context);

[0143] Para ejecutar el principal código JS, las llamadas de capa H2O llama al siguiente js2o_main (js2o_cx *context);

[0144] Esta función devuelve 0 si ningún error, o un número negativo para error de tiempo de ejecución.
Para ejecutar un controlador de eventos, la capa H2O llama al siguiente API para ejecutar el controlador: int js2o_execute (js2o_cx *context, js2o_obj *this, int handle);

[0145] El valor de control se proporciona por el API js2o_compile_add_src en le tiempo de compilar.
La función js2o_execute devuelve los siguientes valores: JS2O_TRUE si el controlador devuelve real, JS2O_FALSE si la controlador devuelve falso, JS2O_NULL si un error de tiempo de ejecución ocurre, cualquier js2o_val si ningún error y el controlador no devuelve un valor.
Nótese que el JS motor de ejecución es preferiblemente no reentrante.
En una forma de realización preferida, las funciones js2o_main y js2o_execute no se puede llamar mientras otro JS controlador se está ejecutando.

[0146] Un fichero JS externo se compila en un módulo de código DLL de Open TV.
La capa H2O llama a una función de inicialización de este módulo.
Esta función de inicialización recibe como contribución el contexto de JS20.
Así, el principal Módulo JS de código es inicializado primero de la siguiente manera: int js2o_dll_init (js2o_cx *context, char *name, void *module).
El parámetro de nombre es el pasado en el tiempo de compilar(js2o_compile_add_bin).
Después de esta llamada, las funciones definidas en ese módulo están disponibles al JS contexto.
El externo módulo JS de código también tiene un API para ejecutar el instrucción global JS de este fichero.

En una forma de realización preferida, el externo módulo JS de código es un módulo de código de Open TV con información específica JS20 almacenada en una cabezcera de módulo de Open TV.
El módulo de clase de objeto define una clase nueva de objetos.
Este módulo se declara en la página de HTLM con la declaración de objeto, y declarado al compilador que utiliza el API js2o_compile_add_class.. Las llamadas del motor JS2O lo siguiente C API, proporcionado por este módulo, para obtener los constructores de función de esta clase de objeto.

[0147] La función js2o_obj *module_class_get_constructor (js2o_cx *context, char *classname): es función exportada #0.
El módulo de clase de objeto, en una forma de realización preferida, es un módulo de código de Open TV con información específica JS20 almacenada en una cabecera de módulo de Open TV.

[0148] Tiempo de ejecución JS accede a la siguiente función proporcionada por la estructura de tiempo de ejecución H2O: js2o_obj * h2o_get_class_constructor (js2o_cx *context, char *classname).
Esta función devuelve preferiblemente el objeto constructor de función para la clase específicoa
El nombre de clase es el mismo valor como el pasado en el CLASSID de la declaración de OBJECT (objeto).
Esta función localiza internamente el módulo de clase y llama el module_class_get_constructor function.
La siguiente instrucción está disponible para que JS cree un objeto de una clase externa.
El nombre es lo mismo como el un especifico en el primero pasado en la clase ID de la declaración de objeto.
Lo siguiente C principios activos están disponibles para desarrollo de clases de objeto y bibliotecas.
Una función vacía js2o_runtime_error (js2o_cx *context) se ejecuta cuando un error de tiempo de ejecución ocurre dentro de un método de aparato o función de vtable, y bibliotecas dentro de js2o, cuando una condición de error ocurre.
La función char *js2o_get_number_property_string (int property Number) devuelve la cadena asociada al número de propiedad bien conocida.

[0149] La presente invención ha sido descrita en televisión interactiva en una forma de realización preferida, no obstante, la presente invención también se puede concretar en una configuración distribuida que comprende un servidor y un dispositivo de cliente.
En otra forma de realización, la presente invención se implementa como un conjunto de instrucciones en un medio legible informático, que comprende ROM, RAM, CD ROM, Flash o cualquier otro medio legible informático, ahora conocido o desconocido que cuando ejecutado causa que un ordenador implemente el método de la presente invención.

[0150] Mientras una forma de realización preferida de la invención ha sido mostrada por la invención anterior, se hace a modo de ejemplo solo y no está destinado a limitar el ámbito de la invención, que se define por las siguientes reivindicaciones.

REIVINDICACIONES

1. Método para compilar un script para ejecución en un dispositivo de cliente en un sistema de televisión interactiva que comprende:
recibir una página de HTLM que contiene al menos un script de un proveedor de servicio a un servidor;
extraer el script de la página de HTLM en un filtro;
compilar el script en un código compatible con cliente para ejecución a un dispositivo de cliente;
transmitir el código compatible con cliente compilado al dispositivo de cliente; y ejecutar el código compatible con cliente compilado en el dispositivo de cliente.

2. Método según la reivindicación 1 que comprende además:
reunir elementos de página de script para un lenguaje de la página de HTLM en al menos un paquete;
y pasar al menos un paquete de elementos de página de script reunido al compilador de script de modo que todos los elementos de script para una página de HTLM sean compilados juntos.

3. Método según la reivindicación 1 que comprende además:
almacenar en caché el código compatible con cliente compilado para la página de HTLM;
valorar un script de página de HTLM entrante en el compilador para determinar si el script de página entrante está ya en la caché; si la página de HTLM entrante no se encuentra en las páginas compiladas almacenadas,
compilar el script para la página de HTLM entrante y almacenar en caché el código compatible con cliente compilado para la página de HTLM entrante; y
si la página compilada se encuentra en la caché, recuperar el código compatible con cliente compilado de la caché y enviar el código compatible con cliente compilado al dispositivo de cliente.

4. Método según la reivindicación 1 que comprende además:
compilar los scripts entrantes para la página de HTLM en un formato intermedio; y
compilar el formato intermedio en el código compatible con cliente.

5. Método según la reivindicación 1 que comprende además:
generar una ramificación del elemento de script en el servidor, y
generar códigos compatibles de cliente de la ramificación del elemento de script.

6. Método según la reivindicación 5 que comprende además:
enviar los códigos compatibles de cliente a un ensamblador binario para ensamblaje de una representación ensamblada.

7. Método según la reivindicación 6 que comprende además:
generar al menos un módulo de código compatible con el cliente de la representación ensamblada; y
pasar el módulo de código compatible con cliente al dispositivo de cliente para su ejecución.

8. Método según la reivindicación 1 que comprende además:
compilar una copia de duplicado de script; y enviar la copia compilada del duplicado de script al cliente para su uso como un objeto externo por múltiples páginas HTLM.

9. Método según la reivindicación 1 que comprende además:
proporcionar una interfaz para definir un objeto predefinido

10. Método según la reivindicación 1 que comprende además:
proporcionar una interfaz de script de modelo de objeto compatible con el cliente para manipular objetos definidos por el cliente.

11. Método según la reivindicación 1, que comprende además:
extraer y reunir scripts para un lenguaje de la página HTLM, donde solo un script que será ejecutado en el dispositivo de cliente es extraído, y pasar los scripts reunidos a un compilador de script para compilar los scripts reunidos en al menos un módulo de código compatible de cliente;
almacenar en caché el código compatible con cliente compilado para páginas HTLM, y verificar un script de página de HTLM entrante para determinar si la página entrante está en la caché de script de código compatible con cliente compilado y si se encuentra en la caché, compilar el script para la página de HTLM.
almacenar en caché el código compatible con cliente compilado y si el código compatible con cliente compilado se encuentra en la caché, recuperar del código compatible con cliente compilado de la caché añade envío del código compatible con cliente compilado al dispositivo de cliente; y

ejecutar el código compatible con cliente compilado en el dispositivo de cliente.

12. Método según la reivindicación 1 que comprende además:
generar un árbol del elemento de script y generar códigos compatibles con el cliente del árbol del elemento de script;
compilar el script entrante para la página HTLM en un formato intermedio y luego compilar el formato intermedio en el código compatible de cliente;
enviar los códigos compatibles de cliente a un ensamblador binario para ensamblaje de una representación ensamblada; y generar un módulo de código compatible con el cliente de la representación ensamblada para ejecución en el dispositivo de cliente.

13. Método según la reivindicación 1 que comprende además:
compilar una copia de un duplicado de script y enviar la copia compilada del duplicado de script al dispositivo de cliente para uso como un objeto externo por múltiples páginas HTLM;
proporcionar una interfaz para objetos predefinidos de definición, donde un objeto predefinido es al menos uno de OSD o canales; y proporcionar una interfaz de script de modelo de objeto compatible con el cliente para manipular objetos definidos por el cliente.

14. Método según la reivindicación 1, que comprende además:
enlazar en el momento de compilar un nombre variable de script a una ubicación de memoria que contendrá el valor de nombre variable para ejecución en el momento de ejecución en el dispositivo de cliente.

15. Método según la reivindicación 1, que comprende además:
enlazar en el momento de compilar un nombre variable de script a un valor para ejecución en el momento de ejecución en el dispositivo de cliente.

16. Método según la reivindicación 1 donde solo una parte del es extraído, compilado y transmitido al dispositivo de cliente.

17. Método según la reivindicación 16 que comprende además:
detectar una clave en el script para determinar si la parte del script serán ejecutada en el dispositivo de cliente.

18. Método según la reivindicación 1 donde el objeto predefinido comprende al menos uno de un OSD y canales.

19. Método según la reivindicación 1 donde la página de HTLM es enviada de una cabecera al servidor.

20. Método según la reivindicación 1 donde el HTLM es parte de una transacción de ecommerce entre el proveedor de servicio y el usuario en el dispositivo de cliente.

21. Medio legible informático que contiene instrucciones para ejecutar el método de cualquiera de las reivindicaciones 1 a 19.

22. Equipo para compilar scripts para ejecución en un dispositivo de cliente en un sistema de televisión interactiva que comprende:
una memoria de servidor para recibir una página de HTLM que contiene al menos un script de un proveedor de servicio a un servidor; un componente de extracción para extraer los scripts de la página de HTLM en un filtro;
un componente de compilador para compilar el script en un código compatible con cliente para ejecución en un dispositivo de cliente; un enlace de comunicación para la transmisión del código compatible con cliente compilado en el dispositivo de cliente; y un dispositivo de cliente para ejecutar el código compatible con cliente compilado.

23. Equipo según la reivindicación 22 que comprende además:
un componente de programa para reunir elementos de página de script para un lenguaje de la página de HTLM en al menos un paquete; y una memoria que reúne elementos de página de script para acceso por el compilador de script de modo que todos los elementos de script para una página de HTLM son compilados juntos.

24. Equipo según la reivindicación 22 donde solo una parte del script que será ejecutado en el dispositivo de cliente es extraída, compilada y transmitida al dispositivo de cliente.

25. Equipo según la reivindicación 22 que comprende además:

una caché para almacenamiento en caché de códigos compatibles de cliente compilados para páginas HTLM; donde el compilador comprende además un componente de programa para control de un script de página de HTLM entrante en el compilador para determinar si el script de página entrante está ya en las páginas compiladas guardadas, donde si la página de HTLM entrante no se encuentra en la caché, compilar el script para la página de HTLM entrante y almacenar el código compatible con cliente compilado para la página de HTLM entrante y si la página compilada se encuentra en la caché, recuperar el código compatible con cliente compilado de la caché y enviar el código compatible con cliente compilado al dispositivo de cliente.

26. Equipo según la reivindicación 22 que comprende además:
un componente de compilador para compilar el script entrante para la página de HTLM en un formato intermedio; y un componente de compilador para compilar el formato intermedio en el código compatible con cliente.

27. Equipo según la reivindicación 22 que comprende además:
memoria para con un árbol de elemento de script en el servidor; y un componente de programa para generar códigos compatibles de cliente de la ramificación del elemento de script.

28. Equipo según la reivindicación 27 que comprende además:
un ensamblador binario para recibir los códigos compatibles de cliente a un ensamblador binario para ensamblaje de una representación ensamblada.

29. Equipo según la reivindicación 28 que comprende además:
al menos un módulo de código compatible con cliente generado de la representación ensamblada para pasar al dispositivo de cliente para su ejecución.

30. Equipo según la reivindicación 22 que comprende además:
una copia compilada de duplicado de script para enviar al cliente para su uso como un objeto externo por múltiples páginas HTLM.

31. Equipo según la reivindicación 22 que comprende además: una interfaz para definición de un predefinido obj ect.

32. Equipo según la reivindicación 31 donde el objeto predefinido comprende al menos uno de un OSD y canales.

33. Equipo según la reivindicación 22 que comprende además:
una interfaz de script de modelo de objeto compatible con el cliente para manipular objetos definidos por el cliente.

34. Equipo según la reivindicación 22 que comprende además:
un componente de programa para extraer y reunir scripts para un lenguaje de la página de HTLM, donde solo un script que será ejecutado en el dispositivo de cliente es extraído, y pasar los scripts reunidos a un compilador de script para compilar los scripts reunidos en al menos un módulo de código compatible con el cliente;
una caché para el almacenamiento de los códigos compatibles compilados para páginas HTLM, y verificar un script de página de HTLM entrante para determinar si la página entrante está en la caché de script de código compatible compilado y si no se encuentra en la caché, compilar el script para la página de HTLM y almacenar el código compatible con cliente compilado y si el código compatible con cliente compilado se encuentra en la caché, recuperar del código compatible con cliente compilado de la caché y enviar el código compatible con cliente compilado al dispositivo de cliente; y un dispositivo de cliente para ejecutar el código compatible con cliente compilado.

35. Equipo según la reivindicación 22 que comprende además:
un árbol de elemento de script para generar unos códigos compatibles con el cliente del árbol del elemento de script;
un componente de formato intermedio para compilar el script entrante para la página de HTLM en el formato intermedio y luego compilar el formato intermedio en el código compatible de cliente; un ensamblador binario para ensamblaje de los códigos compatibles de cliente de una representación ensamblada para generar un módulo de código compatible con el cliente de la representación ensamblada para ejecución en el dispositivo de cliente.

36. Equipo según la reivindicación 22 que comprende además:
compilar una copia de un duplicado de script y enviar la copia compilada del duplicado de script al dispositivo de cliente para uso como un objeto externo por múltiples páginas HTLM;

proporcionar una interfaz para definir objetos predefinidos, donde un objeto predefinido es al menos uno de OSD o canales; y proporcionar una interfaz de script de modelo de objeto compatible con el cliente para manipular objetos definidos por el cliente.

37. Equipo según la reivindicación 22 donde la página HTLM es enviada de una cabecera al servidor.

38. Equipo según la reivindicación 37 donde el HTLM es parte de una transacción de ecommerce entre el proveedor de servicio y el usuario en el dispositivo de cliente.

39. Equipo según la reivindicación 22 que comprende además: un componente de compilador para enlazar en el momento de compilar un nombre variable de script a una ubicación de memoria que contendrá el valor de nombre variable en el momento de ejecución en el dispositivo de cliente.

40. Equipo según la reivindicación 24 que comprende además:
un clave para indicar en el script que una parte del script serán ejecutada en el dispositivo de cliente.

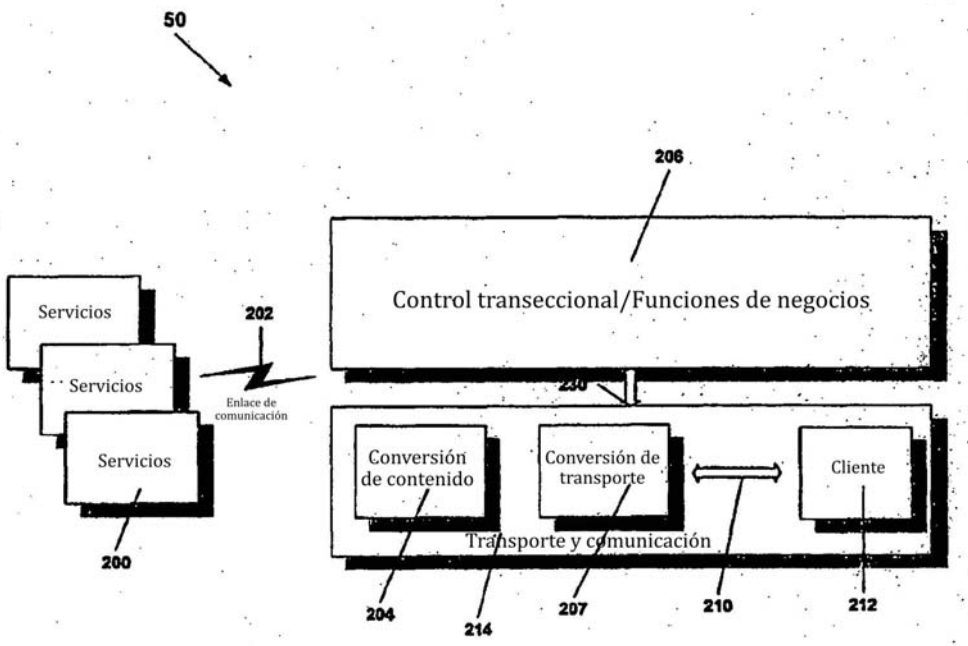
50
206
Control transeccional/Funciones de negocios
Servicios
Servicios
Servicios
202
Enlace de comunicación
200
Conversión de contenido
Conversión de transporte
Cliente
Transporte y comunicación
204
214
207
210
212

Figura 1

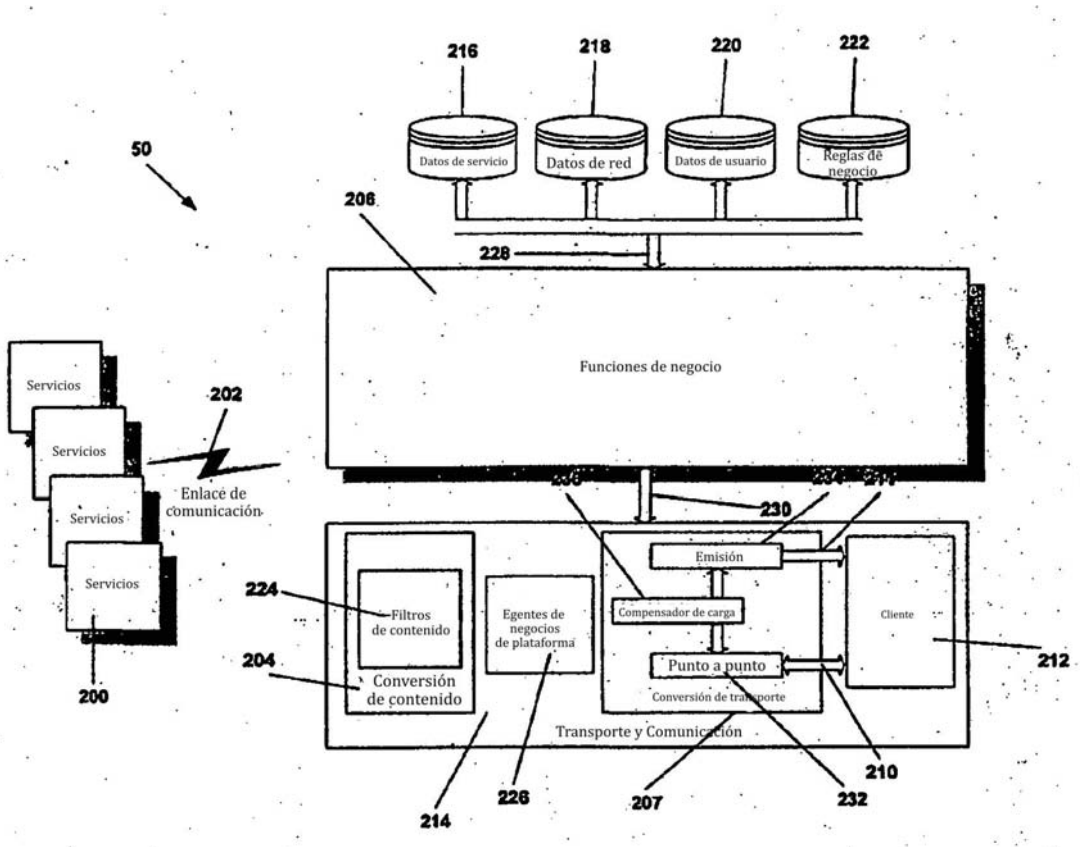
216
218
220
222
50
Datos de servicio
Datos de red
Datos de usuario
Reglas de negocio
208
228
Funciones de negocio
Servicios
Servicios
Servicios
Servicios
202
Enlace de comunicación
230
Emisión
224
Filtros de contenido
Egentes de negocios de plataforma
Compensador de carga
Cliente
212
204
Conversión de contenido
Punto a punto
Conversión de transporte
Transporte y Comunicación
200
214
226
207
232
210

Figura 2

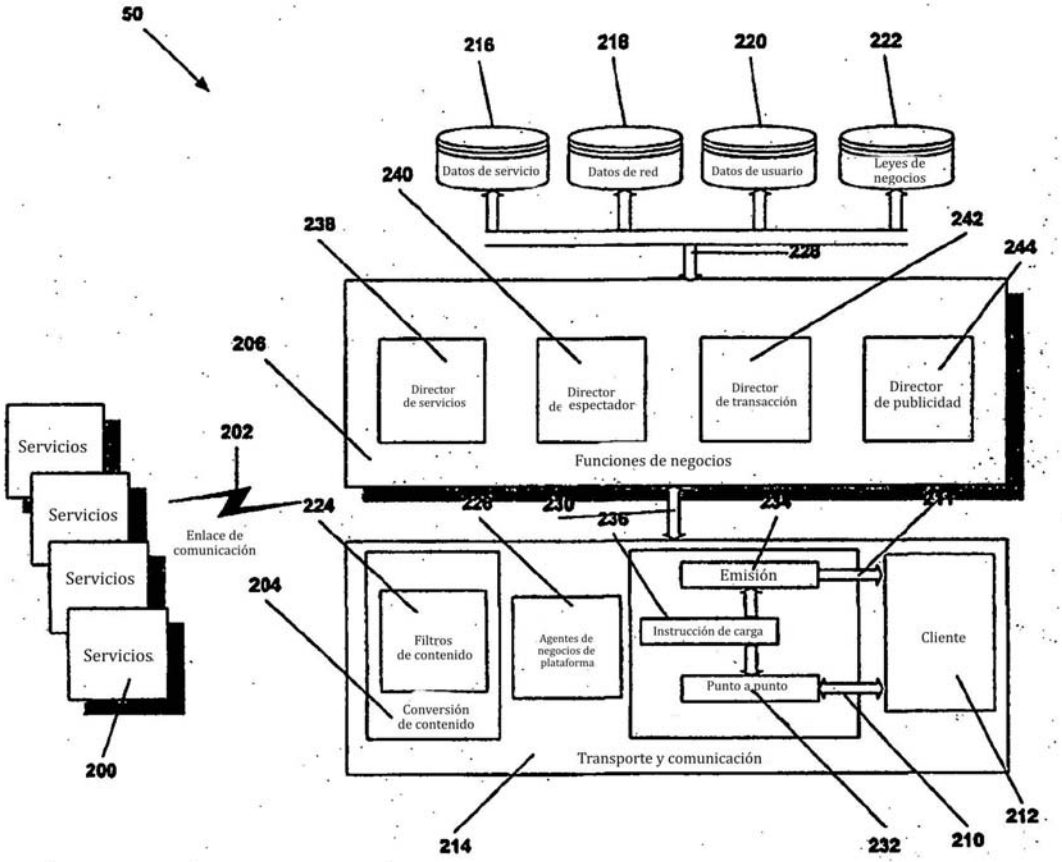
50
216
218
220
222
Datos de servicio
Datos de red
Datos de usuario
Leyes de negocios
240
238
242
244
228
206
Director de servicios
Director de espectador
Director de transacción
Director de publicidad
Funciones de negocios
Servicios
Servicios
Servicios
Servicios
202
Enlace de comunicación
224
226
230
236
234
232
Emisión
Instrucción de carga
Punto a punto
Cliente
204
Filtros de contenido
Conversión de contenido
Agentes de negocios de plataforma
Transporte y comunicación
200
214
232
210
212

Figura 3

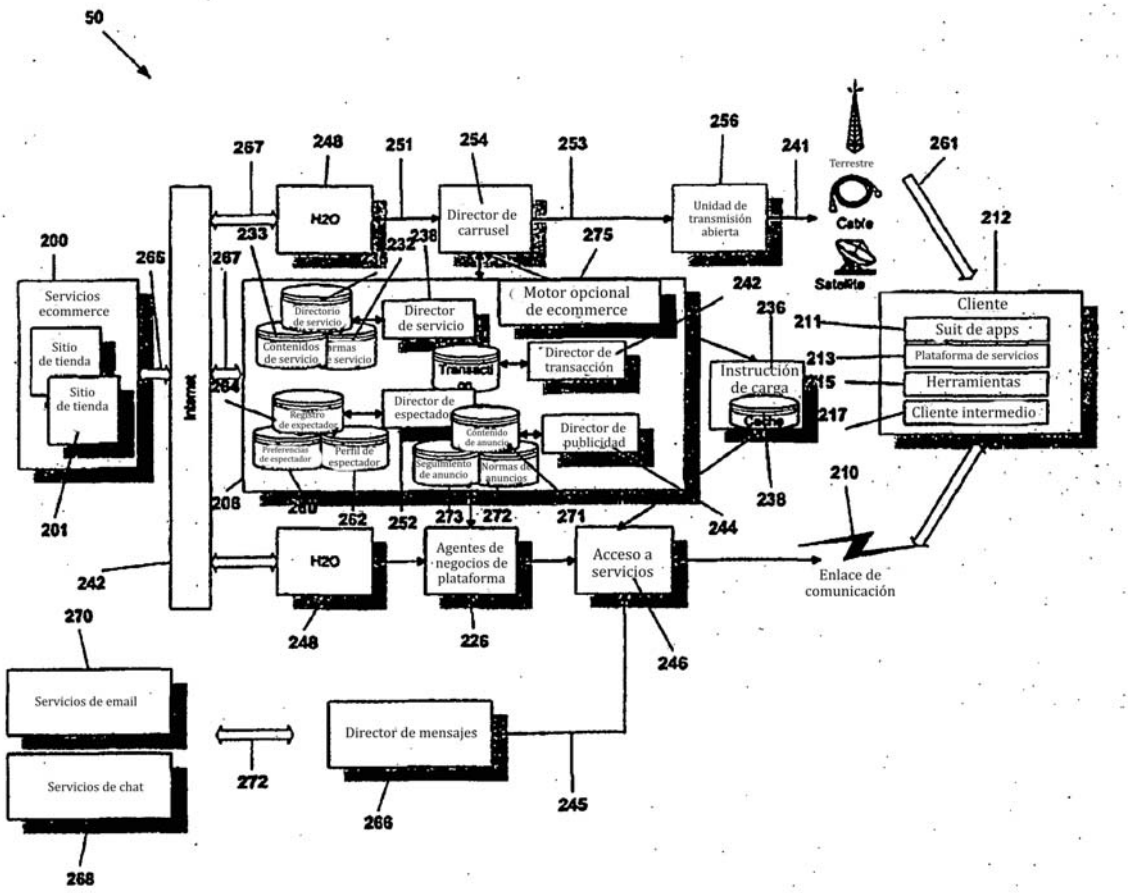
50
267
248
251
254
253
256
241
261
Terrestre
Cable
Satélite
212
200
265
233
232
238
275
242
Servicios ecommerce
Sitio de tienda
Sitio de tienda
Internet
Directorio de servicio
Contenidos de servicio
Director de servicio
Motor opcional de ecommerce
Director de transacción
Transacción
Director de espectador
Registro de espectadores
Preferencias de espectador
Perfil de espectador
Contenido de anuncio
Seguimiento de anuncio
Normas de anuncios
Director de publicidad
Instrucción de carga
Cliente
Suit de apps
Plataforma de servicios
Herramientas
Cliente intermedio
236
211
213
215
217
264
208
280
262
252
273
272
271
244
238
210
201
242
H2O
Agentes de negocios de plataforma
Acceso a servicios
Enlace de comunicación
248
226
246
270
Servicios de email
Servicios de chat
272
Director de mensajes
266
245
268

Figura 4

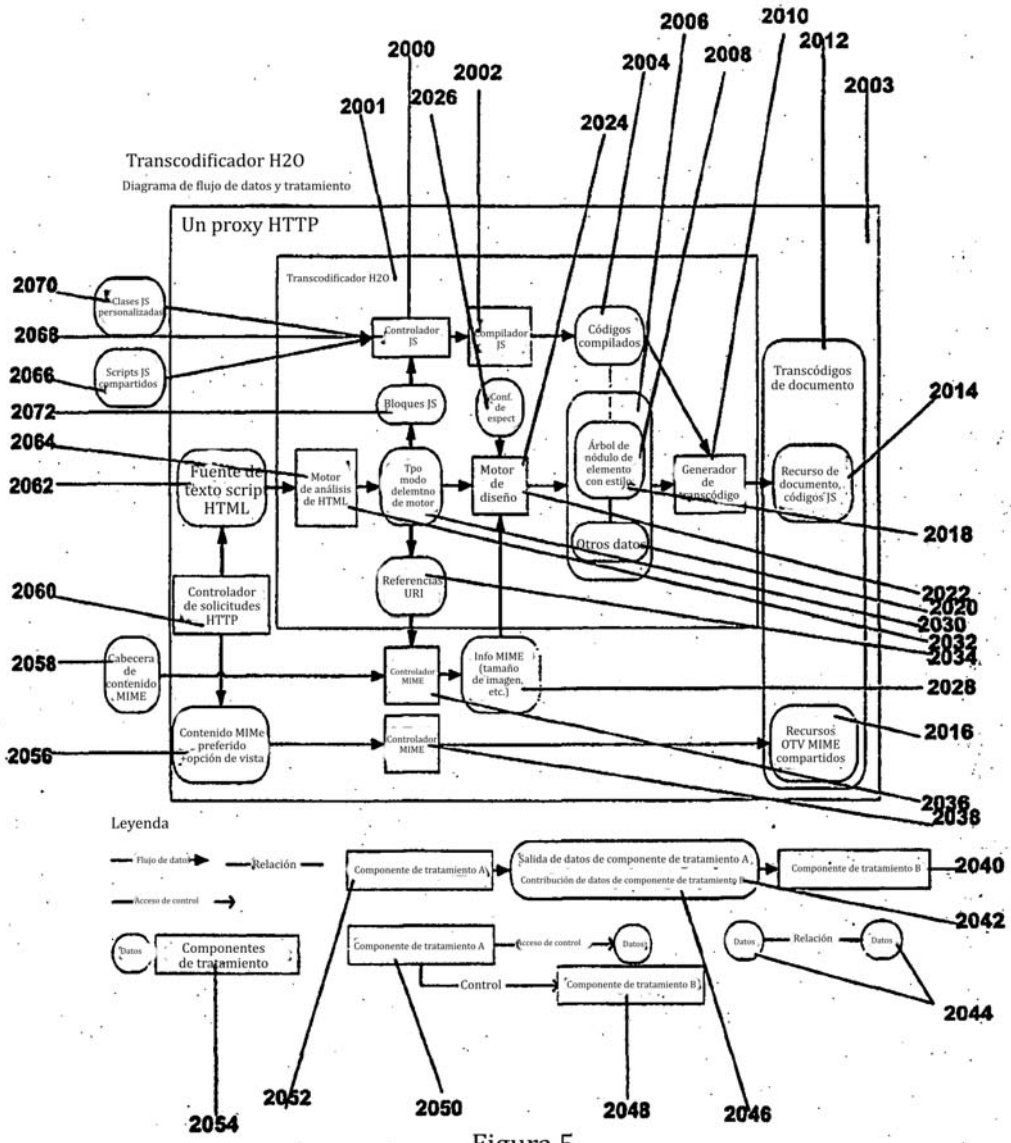
Transcodificador H2O
Diagrama de flujo de datos y tratamiento
Un proxy HTTP
Transcodificador H2O
Clases JS personalizadas
Scripts JS compartidos
Fuente de texto scrip HTML
Controlador de solicitudes HTTP
Cabecera de contenido MIME
Contenido MIME preferido +opción de vista
Controlador JS
Compilador JS
Bloques JS
Conf. de espect
Motor de análisis de HTML
Tpo modo delemtno de motor
Motor de diseño
Referencias URI
Controlador MIME
Info MIME (tamaño de imagen, etc.)
Controlador MIME
Códigos compilados
Árbol de nódulo de elemento con estilo
Otros datos
Generador de transcódigo
Transcódigos de documento
Recurso de documento, códigos JS
Recursos OTV MIME compartidos
Leyenda
Flujo de datos
Relación
Acceso de control
Componentes de tratamiento
Componente de tratamiento A
Componente de tratamiento A
Acceso de control
Control
Componente de tratamiento B
Salida de datos de componente de tratamiento A
Componente de tratamiento B
Datos
Relación
Datos
2000
2001
2002
2003
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
2026
2028
2030
2032
2034
2036
2038
2040
2042
2044
2046
2048
2050
2052
2054
2056
2058
2060
2062
2064
2066
2068
2070
2072

Figura 5

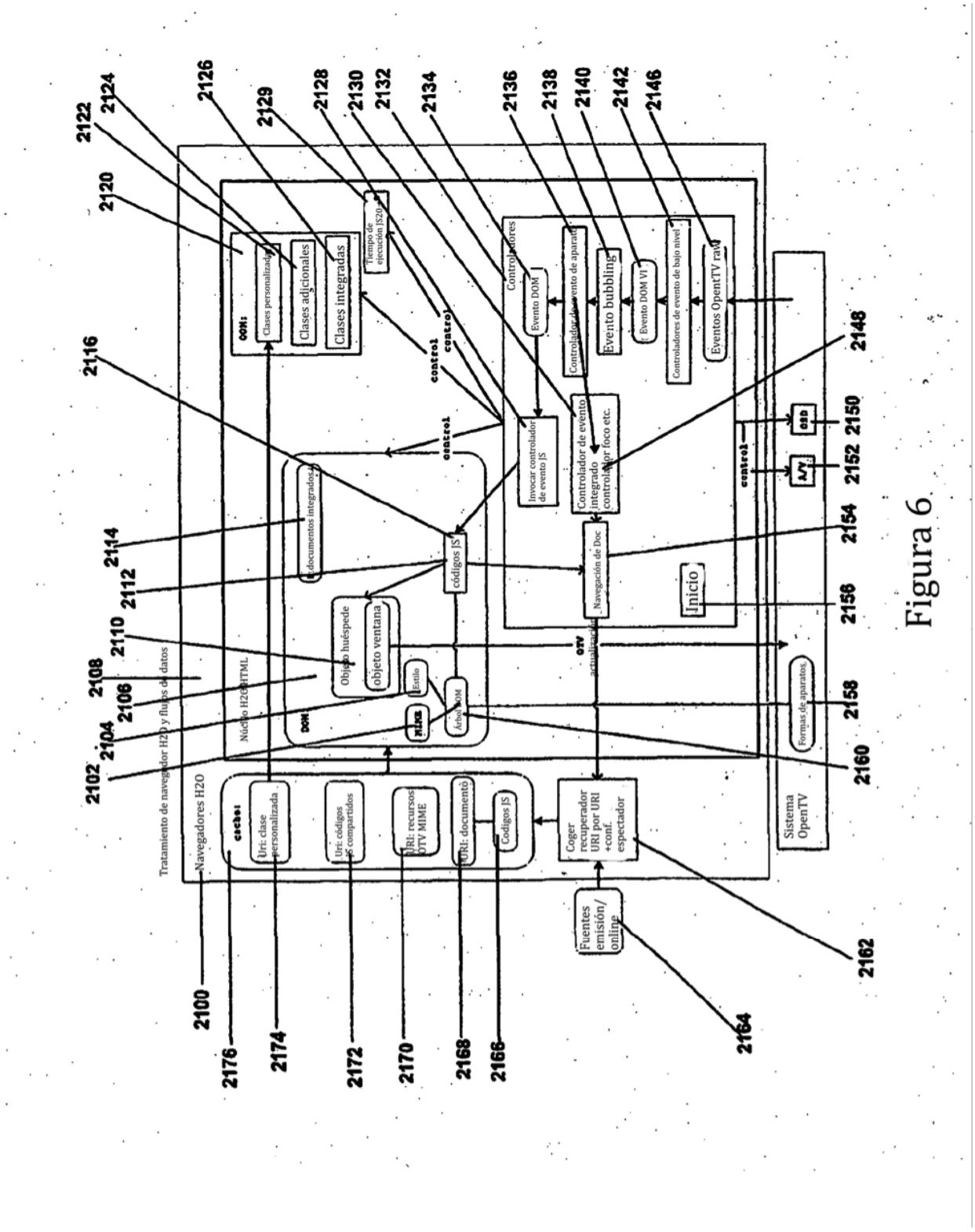
Tratamiento de navegador H2O y flujos de datos
Navegadores H2O
Núcleo H2OHTML
DOM:
Clases personalizadas
Clases adicionales
Clases integradas
Tiempo de ejecución JS2O
Controladores
Evento DOM
Controlador de evento de aparato
Evento bubbling
Evento DOM VI
Controladores de evento de bajo nivel
Eventos OpenTV raw
control
Invocar controlador de evento JS
Controlador de evento integrado controlador foco etc.
Objeto huéspede
objeto ventana
Estilo
Árbol DOM
documentos integrados
códigos JS
Navegación de Doc
Inicio
OTV actualización
Formas de aparatos
Sistema OpenTV
Coger recuperador URI por URI +conf. espectador
Fuentes emisión/ online
caché:
Uri: clase personalizada
Uri: códigos JS compartidos
URI: recursos OTV MIME
URI: documentó
Codigos JS
2100
2102
2104
2106
2108
2110
2112
2114
2116
2120
2122
2124
2126
2128
2129
2130
2132
2134
2136
2138
2140
2142
2146
2148
2150
2152
2154
2156
2158
2160
2162
2164
2166
2168
2170
2172
2174
2176

Figura 6

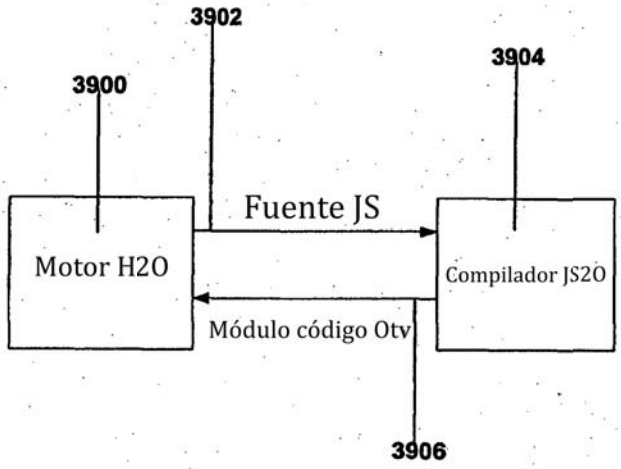
3902
3900
3904
Fuente JS
Motor H2O
Compilador JS2O
Módulo código Otv
3906

Figura 7

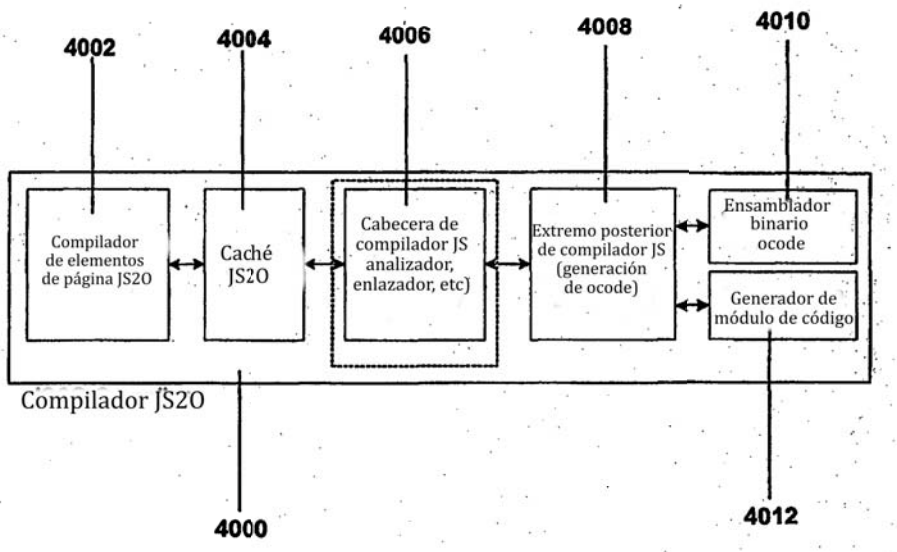
4002
4004
4006
4008
4010
Compilador de elementos de página JS2O
Caché JS2O
Cabecera de compilador JS analizador, enlazador, etc)
Extremo posterior de compilador JS (generación de ocode)
Ensamblador binario ocode
Generador de módulo de código
Compilador JS2O
4000
4012

Figura 8

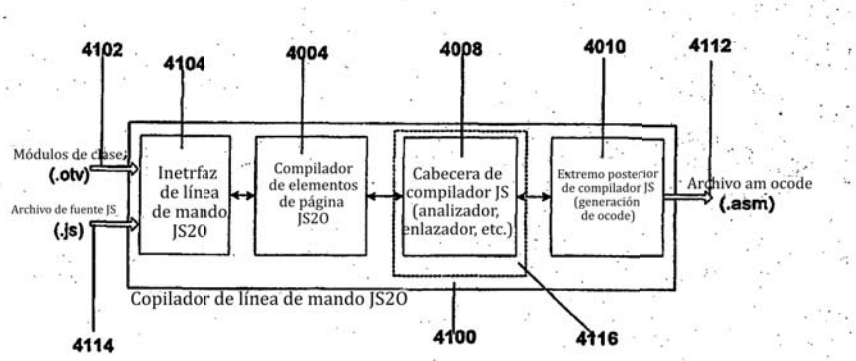
4102
4104
4004
4008
4010
4112
Módulos de clase
(.otv)
Archivo de fuente JS
(.js)
Inetrfaz de línea de mando JS20
Compilador de elementos de página JS20
Cabecera de compilador JS (analizador, enlazador, etc.)
Extremo posterior de compilador JS (generación de ocode)
Archivo am ocode
(.asm)
Copilador de línea de mando JS20
4114
4100
4116

Figura 9

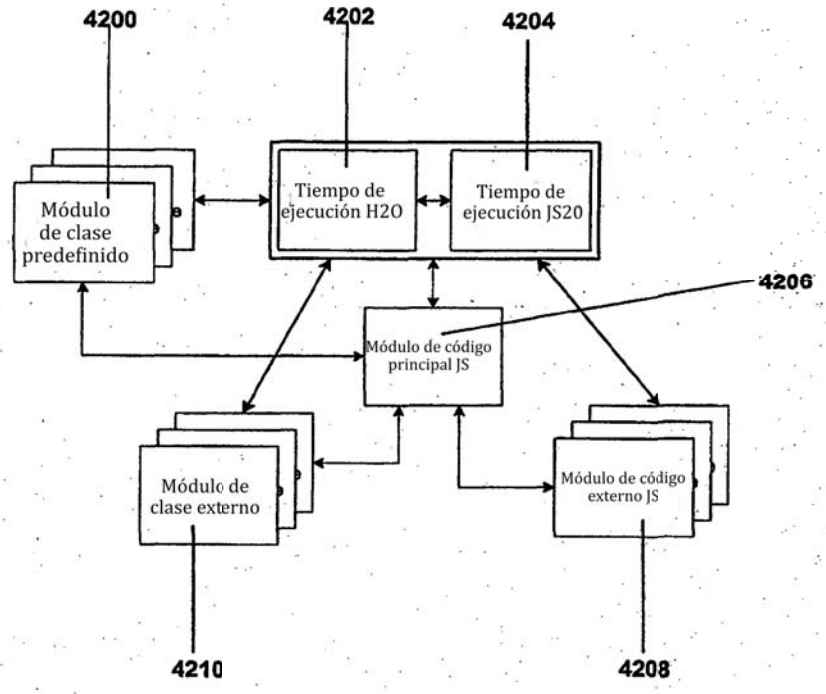
4200
4202
4204
Módulo de clase predefinido
Tiempo de ejecución H2O
Tiempo de ejecución JS20
4206
Módulo de código principal JS
Módulo de clase externo
Módulo de código externo JS
4210
4208

Figura 10

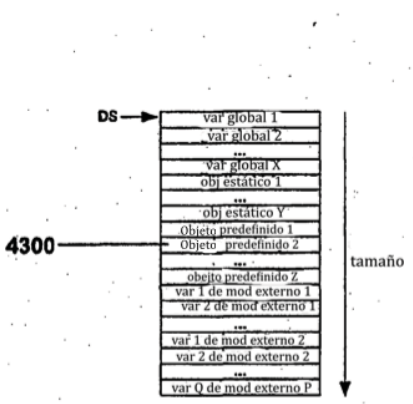

Figura 11

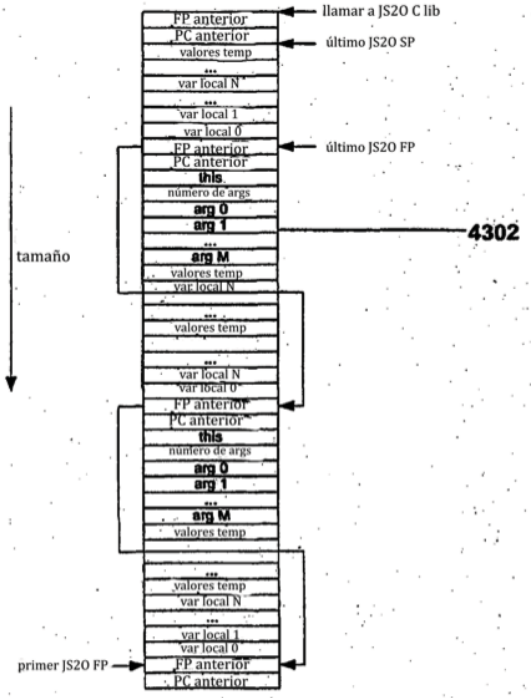

Figura 12