(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 July 2019 (25.07.2019)

WIPO | PCT

(10) International Publication Number
WO 2019/143808 A1

(51) International Patent Classification:
*G06F 16/383* (2019.01)

(21) International Application Number:
PCT/US2019/013998

(22) International Filing Date:
17 January 2019 (17.01.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/619,092 18 January 2018 (18.01.2018) US

(71) Applicant: **BEVARA TECHNOLOGIES, LLC** [US/US]; 253 Mt. Auburn St., Watertown, MA 02472 (US).

(72) Inventors: **GORIN, Jerome**; 253 Mt. Auburn St., Watertown, MA 02472 (US). **BYSTROM, Maja**; 253 Mt Auburn St., Watertown, MA 02472 (US).

(74) Agent: **ARNO, Thomas**; Veros Legal Solution, LLP, 2305 Historic Decatur Road, Suite 100, San Diego, CA 92106 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,

(54) Title: BROWSER NAVIGATION FOR FACILITATING DATA ACCESS

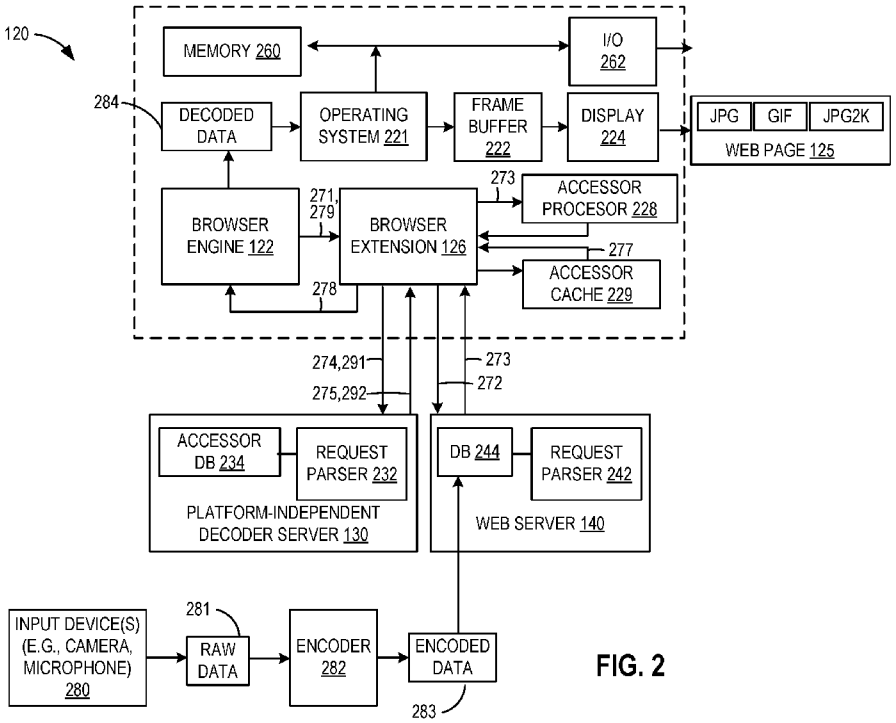



FIG. 2

(57) Abstract: An apparatus for decoding and rendering or otherwise processing, manipulating, or outputting a variety of digital data is provided. The apparatus may include a browser engine configured to receive first encoded digital data. The apparatus may include a browser extension configured to intercept the first encoded digital data transmitted in response to a first request from the browser engine. The browser extension may be configured to intercept a first decoder written in a platform-independent syntax for decoding at least a portion of the first encoded digital data. The browser extension may be configured to point the browser engine to one or more local storage locations of the first encoded digital data and the first decoder.

[Continued on next page]

WO 2019/143808 A1

EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

**Published:**

— *with international search report (Art. 21(3))*

# BROWSER NAVIGATION FOR FACILITATING DATA ACCESS

## RELATED APPLICATIONS

**[0001]** This application claims priority to U.S. Provisional Application No. 62/619,092, filed on January 18, 2018. The disclosures of all the related applications set forth in this section are hereby incorporated by reference in their entireties.

## BACKGROUND

**[0002]** Digital multimedia and general digital data capabilities can be incorporated into a wide range of devices, including digital televisions or displays, digital direct broadcast systems, wireless communication devices such as radio telephone handsets, wireless broadcast systems, personal handheld or wearable devices, laptop or desktop computers, digital cameras, digital recording devices, video gaming devices, video game consoles, data servers, and the like. Digital devices implement image and video encoding and communication techniques or formats such as JPEG, GIF, RAW, TIFF, PBM, MPEG-2, MPEG-4, H.264/MPEG-4, Part 10, Advanced Video Coding (AVC), OBJ, 3DS, and DIMACS to store, transmit and receive digital images and video efficiently. Digital devices implement audio encoding techniques or formats such as, AAC, MP3, and WAV to store, transmit, and receive digital audio efficiently. Digital devices implement additional data and graphics encoding techniques or formats such as IGES, 3DT, PS, MNG, ODF, HDF5, NetCDF, DWG, DXF, and SVG. Digital devices implement document, spreadsheet, and presentation formats such as PowerPoint, PDF, Microsoft Word, Microsoft Excel, and the like. Digital devices may further implement proprietary data storage formats for storage of scientific or other data.

**[0003]** Digital data is commonly encoded in a selected format (e.g. the formats set forth above) prior to transmission or storage. The encoding typically comprises operations such as compression and/or organization into a selected format. Digital data may be embedded in or otherwise referenced by other digital data. For instance, an image, video, data, or animation encoded in one format may be part of an electronic news article, electronic slideshow, web page, or technical paper that is encoded in a different format. In any case, encoded digital data must be accessed by an electronic device to be manipulated, processed, displayed, output in a manner useful for human interaction, or otherwise used by the electronic device or an operator thereof for a desired function. Accessors resident on electronic devices such as mobile devices, DVD players, Blu-Ray players, TV sets, tablets, laptops, computers, or set top boxes comprise software code configured to “undo” the encoding process, typically using metadata provided by the encoding to extract, decompress, and/or reformat the encoded data so that it is in a format useable by the device and/or software on the device such as an OS, a browser, a word processor, any other application program, etc. For a PC, such accessors can include DivX, Flash, or Adobe Acrobat, for example. However, a particular

electronic device may not have software that supports decoding one or more formats for digital data it receives. For example, the format used for a received dataset may be a legacy format no longer supported or may be a new format that the receiving device does not yet support. This presents challenges to the content generator or archivist who wishes to ensure that the digital data are always accessible.

**[0004]** Since different access techniques support different formats, the traditional solutions are to either: encode the digital data in many different formats to support many different decoders; or to select a single format in which to encode a particular type of data. As examples of the latter approach, all images may be converted to a JPEG-2000 format, or all text documents may be converted to rich text format or PDF format. In both cases, decoding and re-encoding of digital data can lead to loss in quality and content. Transcoding older formats into new formats may require licensing of the new formats.

Furthermore, the former case requires additional storage for the copies of the data in the various formats, while the latter case relies on a single accessor always being supported on all platforms. Using a web browser running on a mobile device as an example, in the case in which the web browser does not support a particular format, such as JPEG-2000, a data user might see a blank spot in the web page where the JPEG-2000 image was to be displayed. Current approaches may require a user to manually download plug-ins for a browser and/or additional packages for a software application in order to access data encoded in each type of unsupported data format.

**[0005]** It should be noted that this Background is not intended to be an aid in determining the scope of the claimed subject matter nor be viewed as limiting the claimed subject matter to implementations that solve any or all of the disadvantages or problems presented above. The discussion of any technology, documents, or references in this Background section should not be interpreted as an admission that the material described is prior art to any of the subject matter claimed herein.

## SUMMARY

**[0006]** In one implementation, an apparatus for decoding data comprises a browser engine configured to issue requests for network resources hosted on servers using network resource identifiers and to receive network resources in response to the requests. The apparatus also comprises and a browser extension configured to intercept at least a first request issued from the browser engine for first encoded data associated with a first network resource identifier, receive the first encoded data from a first server, generate and issue a request for at least a first accessor associated with a second, different network resource identifier based at least in part on the intercepting and/or receiving, and receive the first accessor from a second server. The first accessor

comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data requested by the browser engine.

**[0007]** In another implementation, a system for decoding data comprises browser code configured for execution on general-purpose processing circuitry and memory circuitry storing a plurality of accessors, wherein at least some of the plurality of accessors comprise platform-independent syntax implementing an algorithm for decoding encoded data. In this implementation, the browser code is configured to access the memory circuitry automatically without user intervention and retrieve one or more of the plurality of accessors in response to retrieving encoded data.

**[0008]** In another implementation, a method of decoding digital data for manipulation by and/or output to a user of browser software is provided. The method comprises receiving a user request to obtain a first network resource comprising first encoded digital data, wherein the first network resource is associated with a first network resource identifier; using the first network resource identifier to generate and issue a request to obtain the first encoded data over a network; receiving the first encoded data over the network; without further user intervention, using a second different network resource identifier to generate and issue a request to obtain a first accessor over the network, wherein the first accessor comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data; and receiving the first accessor over the network.

**[0009]** In another implementation, a method of decoding digital data for manipulation by and/or output to a user of browser software comprises receiving a user request to obtain a first network resource comprising first encoded digital data with a browser engine, wherein the first network resource is associated with a first network resource identifier. The method further comprises using the first network resource identifier to generate and issue a request to obtain the first encoded data over a network with the browser engine. The method further comprises intercepting the request for the first encoded data, determining an encoding method applied to the first encoded data, and using a second different network resource identifier to generate and issue a request to obtain a first accessor over the network with a browser extension, wherein the first accessor comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data.

**[0010]** It is understood that various configurations of the subject technology will become apparent to those skilled in the art from the disclosure, wherein various configurations of the subject technology are shown and described by way of illustration. As will be realized, the subject technology is capable of other and different configurations and its several details are capable of modification in various other respects, all without departing from the scope of the subject technology.

Accordingly, the summary, drawings and detailed description are to be regarded as illustrative in nature and not as restrictive.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0011]** Various embodiments are discussed in detail in conjunction with the Figures described below, with an emphasis on highlighting the advantageous features. These embodiments are for illustrative purposes only and any scale that may be illustrated therein does not limit the scope of the technology disclosed. These drawings include the following figures, in which like numerals indicate like parts.

**[0012]** FIG. 1 shows an example of a network comprising one or more web browsers, a wide area network (WAN), one or more web servers, and a platform-independent decoder server, according to some embodiments;

**[0013]** FIG. 2 is a flow chart illustrating several structural aspects of a decoding apparatus capable of performing any of the browser navigation and/or flexible decoding methods disclosed herein, according to some embodiments;

**[0014]** FIG. 3A illustrates a flowchart of a method of flexibly decoding and rendering a variety of multi-media data, in accordance with some embodiments;

**[0015]** FIG. 3B illustrates another flowchart of a method of flexibly decoding and rendering a variety of multi-media data, in accordance with some embodiments;

**[0016]** FIG. 3C illustrates yet another flowchart of a method of flexibly decoding and rendering a variety of multi-media data, in accordance with some embodiments;

**[0017]** FIG. 4 shows an example of an ISO-BMFF format container (e.g., a BVR container), according to some embodiments;

**[0018]** FIG. 5 shows an example of an ISO-BMFF container (e.g., a BVR container) that includes one or more accessors for data contained within the ISO-BMFF container, according to some embodiments;

**[0019]** FIG. 6 shows another example of an ISO-BMFF container (e.g., a BVR container) format, according to some embodiments;

**[0020]** FIG. 7 shows another example of an ISO-BMFF container (e.g., a BVR container) format, according to some embodiments.

**[0021]** FIG. 8 shows another example of an ISO-BMFF container (e.g., a BVR container) format, according to some embodiments;

**[0022]** FIG. 9 shows another example of an ISO-BMFF container (e.g., a BVR container) format, according to some embodiments;

**[0023]** FIG. 10 shows another example of an ISO-BMFF container (e.g., a BVR container) format, according to some embodiments.

**[0024]** FIG. 11 shows an example of a ZIP format container (e.g., a BVR container), according to some embodiments;

## DETAILED DESCRIPTION

**[0025]** The following description and examples illustrate some exemplary implementations, embodiments, and arrangements of the disclosed invention in detail. Those of skill in the art will recognize that there are numerous variations and modifications of this invention that are encompassed by its scope. Accordingly, the description of a certain example embodiment should not be deemed to limit the scope of the present invention.

**[0026]** Implementations of the technology described herein are directed generally to extracting media from containers and providing media decoding capabilities for media files downloaded to and/or stored on computing devices. To facilitate an understanding of the various embodiments described herein, certain terms are defined below.

### Definitions

**[0027]** *Accessor:* Software that can make encoded digital data available for display, interaction, processing, output, or any other desired functional use on an electronic device receiving and/or storing the data. Generally, a particular implementation of an accessor is configured to decode data that is encoded according to a particular defined compression algorithm/organizational format, or a family of compression algorithms/organizational formats. An accessor may also be referred to as a "decoder" herein. In some cases, in addition to basic decoding functionality such as decompression, an accessor may also function to extract one or more encoded data files from a container for decoding. An accessor as set forth herein can also include, in addition to one or more extraction and/or decoding functions, data manipulation or processing functionality that can be applied to digital data decoded by the accessor.

**[0028]** *Container:* A data file assembled in accordance with a metafile format defining how one or more data files and metadata associated with the one or more data files are organized together as the container data file. Containers can be nested, wherein a first container is packaged as a data file in a second container.

**[0029]** *BVR Container:* A container including encoded data to be decoded and one or more accessors configured to decode encoded data, in some cases for subsequent rendering, processing, presentation, or other manipulation by one or more processing, output or display device(s). As shown further below, a single BVR container may contain one or more data files or data sets, which may be encoded in one or more different formats, and may also include one or more platform-independent accessors configured to decode data that has been encoded in one or more different formats. Examples of some BVR containers that have been developed by Bevara

Technologies, LLC in Watertown MA are provided in FIGS. 3-11 and the corresponding description below. In the example BVR containers of FIGS. 4-10, the BVR containers use the ISO-BMFF format. In the example of FIG. 11, the BVR container uses the ZIP format.

**[0030]** *Browser and Web Page:* A browser is a computer program that provides functionality to a computer for executing syntax contained in web pages. The computer may be connected to a computer network, and the network may be, and usually will be, the Internet. As used herein, browsers and web pages together provide functionality to a computer connected to a network (e.g. the Internet) at least sufficient to request, retrieve, and render at least some network resources including web pages themselves, and to execute at least some links contained within or referred to in retrieved web pages to retrieve other web pages specified with the links. Web pages may include references such as uniform resource locators (URLs) and/or universal resource identifiers (URIs) to other network resources that contain images or other data that is retrieved by the browser from the network or from a cache memory when executing the web page, and may also include programs, libraries, style sheets, scripts, and the like which are executed in the browser environment when executing the web page. Executing some of these items may require prior separate loading of third party software onto the computer for the browser to utilize, and some may be executable natively by the browser itself. As browsers become more sophisticated, they tend to incorporate more functionality natively that may have been introduced originally as third-party code. Any of these items that are accessed, used, and/or retrieved during browser execution of web page syntax are considered to be included as a component of the "web page" as that term is used herein. Examples of browsers include, but are not limited to, Internet Explorer and Edge distributed by Microsoft, Firefox distributed by Mozilla, and Chrome distributed by Google. Example web page syntax that can be executed by browser engines include the various versions of HyperText Markup Language (HTML) promulgated by the World Wide Web Consortium (W3C).

**[0031]** *Browser Engines and Browser Extensions:* Browser engines and browser extensions generally run together as the "browser" on a given computing device. There is no strict functional demarcation defining what part of a browser is engine, and what part is extension. As used herein, the term browser engine is used to refer to the software code of a browser that provides basic navigation and syntax execution for a browser as described in the definition of Browser and Web Page set forth above. Software code that provides functionality to a browser beyond that provided by such a browser engine may be referred to herein as a browser extension. Browser extensions may perform functions such as displaying and executing toolbar functions on top of retrieved web pages or blocking browser engine retrieval of advertisements that would otherwise be downloaded in conjunction with web page execution. A browser extension is typically executed within browser allocated memory space anytime the browser engine is running on a computing device.

Accordingly, a browser extension may be periodically, intermittently, or continuously monitoring operations of the browser engine and, in some cases as will be described in more detail below, intercepting one or more communications from and/or to the browser engine. Browsers such as Internet Explorer and Chrome provide internal functionality allowing them to interact with browser extension software code distributed by third parties so long as the third-party code complies with the interface for browser extensions provided with the browser. However, as used herein, the terms browser engine and browser extension are not distinguished by how they are developed or distributed. Rather, the browser engine is that part of browser code that provides basic navigation and syntax execution capabilities like http format communication, html page interpretation, script and code execution such as Java and WebAssembly, whereas the browser extension is that part of browser code that provides additional functionality beyond those basic functions.

**[0032]** *Browser Plug-In:* A browser plug-in is a computer program invoked by the browser and run in browser allocated memory specifically in response to the browser engine reading code that invokes the plug-in. Accordingly, a browser engine may operate for extended periods of time without invoking and/or loading a plug-in and a plug-in never intercepts communications not addressed to the plug-in. Moreover, plug-ins must be explicitly loaded and/or enabled by a user of a browser engine in response to the browser engine reading code requiring functionality of the plug-in. For example, the Adobe Flash plug-in provides functionality to a browser engine for decoding and rendering Flash video files in response to the browser engine finding a Flash video MIME type in a web page. Plug-Ins are becoming deprecated technology as their functionality is increasingly incorporated into the native code of browsers themselves.

**[0033]** *Server:* Software executing on processing hardware coupled to a computer network having network resources accessible thereto that is configured at least in part to respond to client access requests to use or retrieve the network resources accessible to the server. Two or more different servers may be executing in parallel on a single hardware component such as a microprocessor or computer. A particular piece of computer hardware that is executing server software may also be referred to as a server.

**[0034]** *Client:* Software executing on processing hardware coupled to a computer network configured at least in part to issue requests to servers to use or retrieve network resources accessible to servers. Two or more different clients may be executing in parallel on a single hardware component such as a microprocessor or computer. Both clients and servers may be executing in parallel on a single hardware component such as a microprocessor or computer. A client executing on a processor may issue a request to a server executing on the same processor, which may respond to that client request. A particular piece of computer hardware that is executing

client software may also be referred to as a client. A particular piece of computer hardware may be a client and a server at the same time.

**[0035]** *Internet:* The globally interconnected system of computers and computer networks that evolved from ARPANET and NSFNET over the late 1980s and early 1990s that may utilize TCP/IP network communication protocols.

**[0036]** *Network Resource Identifier:* A definition of a network resource (e.g. by storage location and filename) that is used by client computers to specify a network resource in access requests issued to the network by the client computers. A network resource identifier may also be referred to as a location of a network resource such as an image or a web page. Currently, when the network is the Internet, Network resource identifiers are known as URLs that are formatted in accordance with RFC 3986 of the Internet Engineering Task Force (IETF). For the purposes of this disclosure, any format for specifying a network resource in client access requests issued to a network is within the definition of the term Network Resource Identifier. A network resource identifier, including URLs as currently defined on the Internet, may further include data in addition to data identifying the network resource that a server hosting the network resource associated with the network resource identifier may use for other purposes beyond identifying the requested network resource.

**[0037]** *Web Site:* A collection of network resources including at least some web pages that share a common network resource identifier portion, such as a set of web pages with URLs sharing a common domain name but different pathnames.

**[0038]** *Web Server:* A server that includes functionality for responding to requests issued by browsers to a network, including, for example, requests to receive network resources such as web pages. Currently, browsers and web servers format their requests and responses thereto in accordance with the HyperText Transfer Protocol (HTTP) promulgated by the IETF and W3C. In some embodiments, a web server may also be a content server.

**[0039]** *World Wide Web:* The collection of web pages stored by and accessible to computers running browsers connected to the Internet that include references to each other with links.

**[0040]** *Link:* Syntax that instructs a browser executing the syntax to access a resource such as a network resource that is defined directly or indirectly by the syntax. The link syntax and/or internal browser engine functionality may also define conditions under which the access request is made by the browser engine, for example through cursor position and/or other interaction with an I/O device such as a keyboard or mouse. Some link syntax may cause the browser engine to access the specified network resource automatically while processing the syntax without user prompt or interaction. Links include HTML hyperlinks. A link may be directly coded with, for example, HTML tags and an explicit URL, or may be in the form of a script or other called function defined in a browser, in a browser extension, and/or in a webpage.

**[0041]** *Network Resource:* A web page, file, document, program, service, or other form of data or instructions which is stored on a network node and which is accessible for retrieval and/or other use by other network nodes.

**[0042]** *Redirection Response:* A response that may be provided by a server when processing an access request of a client for a network resource, wherein the response includes a network resource identifier of a different network resource that the client should access for the desired information or action. In the HTTP protocol, a redirection response may also include a 303 status code, and the client receiving the redirection response may then send a GET or other request for the network resource identified by the URL provided in the response.

**[0043]** *Navigate:* Controlling a browser to use a series of links to access a series of network resources.

**[0044]** *Platform-independent:* Software is platform-independent if it is not written specifically for execution by a particular operating/file system and/or a particular physical processing architecture. Platform-independent software generally executes inside another piece of software such as a browser or virtual machine that is not itself platform independent. Examples of platform-independent languages include but are not limited to Java and WebAssembly and, accordingly, platform-independent algorithms as described herein may comprise algorithms written in such platform-independent languages.

**[0045]** *WebAssembly:* A platform-independent programming language developed by W3C which can be executed within some browsers. It is similar to Java in that browsers can interpret WebAssembly code for execution on the underlying OS and hardware but WebAssembly is considered more efficient in its utilization of those underlying resources. Modules of WebAssembly code can run inside JavaScript. WebAssembly modules can be written in a human readable text format and compiled into WebAssembly syntax that is executable by browser software. Tools are also available that can compile programs written in other programming languages such as C into WebAssembly syntax files executable in browsers.

**[0046]** **FIG. 1** shows an example of a network comprising user terminals 110, 120 comprising respective web browser engines 112, 122, a wide area network (WAN) 170, one or more web servers 140, 150, 160, and a platform-independent accessor server 130, according to some embodiments. **FIG. 2** is a block diagram including several structural aspects of a decoding apparatus capable of performing any of the browser navigation and/or flexible decoding methods disclosed herein, such as user terminal 120 of **FIG. 1**, according to some embodiments. Functionality of several implementations of the present disclosure will now be described in connection with **FIGs. 1** and **2** below.

**[0047]** As shown in **FIG. 2**, one or more input device(s) 280 may be configured to generate raw data 281. As a non-limiting example, input device 280 may comprise a camera configured to generate one or more raw image or video files and a microphone configured to generate one or more raw audio files. However, the present disclosure is not so limited and input device 280 may be configured to generate any type of raw data, and the raw data 281 can be of any type and generated in any manner. The raw data 281 is fed into an encoder 282 configured to compress, synchronize, manipulate into one or more container formats, modify for error resilience, and/or otherwise encode raw data 281 into encoded data 283. In some embodiments, encoder 282 may be intrinsic to input device 280. In some other embodiments, encoder 282 may be separate from input device 280. Encoded data 283 may be stored in one or more content or web servers, e.g., servers 140, 150, 160 of **FIG. 1**, which may be communicably coupled or couplable with one or both of user terminals 110, 120 via WAN 170, which may be the Internet. Although in the specific implementation of FIG. 1, the servers and user terminals are connected over a WAN, the invention is not so limited, and one or more servers could be connected in other ways such as over a LAN or even be resident as software on a single physical device. Examples of such encoded data 283 may comprise image file formats such as JPEG, GIF, TIF, JPEG-2000, video file formats such as H.264 (MP4), Theora, F4V (Flash), VP8, VP9, HEVC or AV1, document or web page file formats such as PDF, HTML, or any other encoded or unencoded file types. In addition, one or more encoded files may be stored in one or more of servers 140, 150, 160 in a BVR container (see **FIGs. 4-11**), which in addition to including the one or more encoded files may further include one or more accessors comprising platform-independent syntax that implement one or more algorithms to decode the encoded files in the container. In one advantageous embodiment, the platform-independent syntax for the accessors is written in WebAssembly language. Moreover, although particular file types are illustrated as being stored in servers 140, 150, 160, the present disclosure is not so limited and any of servers 140, 150, 160 may have stored thereon any file(s), file type(s) or format(s).

**[0048]** As shown in **FIG. 1**, conventional user terminal 110 may be in communication with at least some of servers 140, 150, 160 via WAN 170 and may comprise conventional browser engine 112 in communication with an input/output device 114 configured to display at least a portion of a web page 115.

To retrieve a network resource, such as web page 115, browser engine 112 may send a request for a network resource (e.g. an HTTP Request) to web server 140 via WAN 170. In response to the request, web server 140 generates a response (e.g. an HTTP Response) which is transmitted back to browser engine 112 via WAN 170. The response may include the network resource, for example web page 115. Browser engine 112 renders web page 115 including any content for which it has appropriate decoders. This may include a JPEG and a GIF image, for example. For an image for

which browser engine 112 doesn't have a decoder, such as a JPEG-2000 image for example, browser engine 112 is incapable of decoding and rendering such an image and may instead render a blank rectangle as a placeholder for the unrendered and/or undecoded image. Accordingly, conventional rendering of such an image would further require a user of user terminal 110 to explicitly download and instantiate an additional software program specifically designed to operate on/with browser engine 112 and decode such JPEG-2000 images, a requirement that would preclude seamless, transparent decoding and rendering of content to a user of user terminal 110.

**[0049]** U.S. Patent No. 8,995,534, U.S. Pat. App. Pub. No. 2015/0334413, and U.S. Pat. App. Pub. No. 2015/0331870, all of which are incorporated by reference herein in their entireties, describe systems and methods for generating accessor functionality. Instead of converting digital data to one or more new formats, a platform-independent accessor that can decode the encoded data may be packaged with the encoded data. This ensures that there is no conversion loss and minimal additional storage or transmission requirements. However, the digital data, associated metadata, and accessor may still require significant amounts of storage space, may require certain rights, or may have evolving features. Accordingly, a content generator or content provider may not wish to package the full-featured platform-independent accessor with the digital data or may wish to provide both packaged and unpackaged versions but still want data receivers to be able to display or playback all or a portion of data received.

**[0050]** To address this concern, implementations and embodiments described herein may intercept network resource requests or other messages from browser engines and look for data requests that can't be handled natively by the browser engine. For example, a browser engine might request an image or data file that it doesn't have the software to decode. Alternatively, the browser engine might request one or more BVR containers and may or may not have native support for accessing data files packaged in such containers. Implementations and embodiments described herein may produce seamless handling of file formats, using platform-independent accessors, without a user having to manually authorize, install or instantiate an accessor for every single format and also without browser engines having to build native support for every single format. For example, utilizing implementations described herein, a user of user terminal 120 (**FIG. 1**) may be enabled to access encoded data for which browser engine 122 does not have native decoding and/or interpreting ability, without being prompted to download and/or install additional decoding software, thus providing a seamless browsing experience both to the user and to browser engine 122.

**[0051]** Described herein are several implementations and sub-options, described in connection with at least **FIGs. 1** and **2** and user terminal 120, based on whether browser engine 122 requests a BVR container or data comprising another filetype that is not natively supported by browser engine

122, and based on whether browser engine 122 can support BVR containers natively. As will be described in more detail below, the present disclosure may include, describe, or utilize software configured to function in association with browser engines such as, but not limited to Chrome, Safari, Firefox, Opera, Internet Explorer, and Edge.

**[0052]** **FIG. 2** is a block diagram of a digital data processing apparatus (e.g., user terminal 120 of **FIG. 1**) capable of performing any one or more of the browser engine navigation and/or flexible decoding methods disclosed herein, according to some embodiments. **FIG. 2** illustrates user terminal 120 comprising browser engine 122, a browser extension 126, an accessor processor 228, a accessor cache 229, an operating system 221, a frame buffer 222 and a display 224. Display 224 may correspond to at least a portion of I/O device 124 of user terminal 120 illustrated in **FIG. 1**. The operating system 221 may be further coupled to memory 260 and other I/O devices 262. Browser engine 122 may be in communication with operating system 221 and may be configured to ultimately provide decoded data 284 to operating system 221, which may provide decoded data 284 to frame buffer 222, which may provide decoded data 284 to display 224, which may be configured to render a decoded version of data 284 to, for example, a user of user terminal 120. Rendering may comprise generating and/or formatting for single- or multi-dimensional audio, visual, and/or tactile interfaces (e.g., I/O device 124). As noted above, the decoded data may not be all or even any audio or visual data, and accordingly the decoded data will not necessarily be displayed. In some cases, the decoded data 284 can be stored in a memory 260 for further manipulation or processing. Browser extension 126 and/or browser engine 122 may be configured to communicate with one or more servers (e.g., first content or web server 140, second content or web server 150, third content or web server 160 and/or decoder (accessor) server 130), via WAN 170, as will be described in more detail below.

**[0053]** In some implementations, for example first and second implementations described below, browser engine 122 is configured to natively support accessors provided in BVR containers. For example, for BVR containers that include accessors written in WebAssembly, browser engine 122 may be configured to interpret WebAssembly syntax to instantiate and execute the accessor software.

**[0054]** In some other implementations, for example third and fourth implementations described below, browser engine 122 is not configured to natively support accessors provided in BVR containers. For example, for BVR containers that include accessors written in WebAssembly, browser engine 122 may not be configured to interpret WebAssembly syntax, and therefore may not be able itself to instantiate and execute the accessor software. In these embodiments, an accessor processor 228 may be provided coupled to or part of browser extension 126 to instantiate

one or more accessors, by executing WebAssembly syntax for example to decode received encoded data.

**[0055]** In some embodiments, browser extension 126 may also be known as, or comprise, an intercept engine and decision processor. In some embodiments, an accessor cache 229 may be provided coupled to or part of browser extension 126 to store one or more accessors locally with respect to user terminal 120. These and other embodiments and implementations will be described in more detail below.

**[0056]** Functionality of user terminal 120 will now be described in connection with a first implementation, in which browser engine 122 is configured to natively execute WebAssembly programs or another platform-independent programming language which can be used to write scripts/programs for in-browser execution.

**[0057]** Browser engine 122 generates and transmits a request 271 (e.g., an HTTP request) for first encoded data. The first encoded data may comprise at least one file type that browser engine 122 does not natively support decoding of (e.g., Flash, XLS, or JPEG-2000 files). In some embodiments, request 271 is addressed to first content server 140.

**[0058]** Browser extension 126 is in communication with browser engine 122 and may intercept request 271 from browser engine 122. In response, browser extension 126 may reissue request 271, as request 272 (e.g., an HTTP request), to first content server 140 via WAN 170.

**[0059]** First content server 140 may comprise a request parser 242, which may comprise a general-purpose or special-purpose processor configured to perform one or more tasks, actions or steps as described below regarding server 140 (using HTTP protocols for example when the WAN is the Internet). Request parser 242 is configured to receive and parse request 272. First content server 140 may further comprise a database 244 storing the first encoded data associated with a network address. In response to receipt of request 272 from browser extension 126, first content server 140 may transmit the first encoded data in a generated response 273 (e.g., HTTP response) via WAN 170. Browser extension 126 may intercept the first encoded data transmitted from first content server 140.

**[0060]** In some embodiments, browser extension 126 may be configured to determine whether the first encoded data is embedded in a BVR container together with at least a first accessor written in a language supported by the browser engine 122 (e.g. WebAssembly), and if not, whether the first encoded data comprises at least one file type with an encoding format that is unsupported by browser engine 122. This determination can sometimes be made in advance of receiving the encoded first data. For example, an HTML document may include embedded MIME types for files to be retrieved to complete the page rendering process. In other cases, the data being requested will be of an unknown type until the data is received.

**[0061]** In response to a determination that the first encoded data is not embedded in a BVR container with the first accessor and that the first encoded data comprises at least one file type for which decoding is unsupported by browser engine 122, browser extension 126 may be configured to acquire a first accessor comprising a platform-independent program for decoding at least a portion of the first encoded data from platform-independent accessor server 130. Acquiring the first accessor from accessor server 130 may comprise generating and transmitting a request 274 to accessor server 130 and intercepting all or a portion of the first accessor in a response 275 from accessor server 130. For example, accessor server 130 may comprise a request parser 232, which may comprise a general-purpose or special-purpose processor configured to perform one or more tasks, actions or steps as described below regarding accessor server 130. Request parser 232 is configured to receive and parse request 274. Accessor server 130 may further comprise an accessor database 234 storing the first accessor and, in some embodiments, one or more additional accessors written in a platform-independent language that when executed (in a browser for example) will decode encoded data. In some embodiments, the first accessor may be embedded within a BVR container transmitted in response 275. In some embodiments, upon acquisition of the first accessor, browser extension 126 may store the first accessor in local accessor cache 229.

**[0062]** In some embodiments where the first accessor has already been previously stored in accessor cache 229, in response to the above determination(s), browser extension 126 may alternatively be configured to acquire the first accessor from local accessor cache 229 instead of accessor server 130.

**[0063]** The ability to acquire the first decoder from remote decoder server 130 via WAN 170 or from local accessor cache 229 when stored therein may allow for balancing between a required size of accessor cache 229 and an amount of data transmitted via WAN 170. For example, if the first accessor is already stored in accessor cache 229, acquiring the first accessor from accessor cache 229 rather than from remote accessor server 130 may reduce an amount of data transmitted via WAN 170. On the other hand, local accessor cache 229 may have a set amount of memory and may be configured to store and retain decoders based on some algorithm, for example first-in first-out (where, in order to make room to store new accessors as required, the oldest or first cached accessors are deleted first) or a most-used basis (where accessors having been least used in a prior period of time are deleted first, etc.) In such embodiments, acquiring the first accessor from remote accessor server 130 may allow for maintenance of a smaller amount of memory for accessor cache 229, since accessors can be more readily deleted from local accessor cache 229 and reacquired remotely on-demand.

**[0064]** Browser extension 126 may then point browser engine 122 to one or more local storage locations of the first encoded data and the first accessor by messaging the browser engine 122 over

the browser extension API of the browser engine 122 for example, thereby facilitating browser engine 122 to subsequently render or otherwise manipulate or process a decoded version of the first encoded data. For example, browser extension 126 may generate a URL or URI addressed to the local storage location(s) of the first encoded multi-media data and the first decoder and transmit a notification 278 of the URL or URI to browser engine 122. Browser engine 122 then transmits, and browser extension 126 intercepts, a second request 279 (e.g., an HTTP request) for the first encoded multi-media data and the first decoder stored at the one or more local locations. Browser extension 126 then grants browser engine 122 access to the first encoded multi-media data and the first decoder at the one or more local storage locations. Browser engine 122 then decodes and/or interprets the first encoded multi-media data utilizing the first decoder comprising the platform-independent algorithm. Browser engine 122 may be configured to further take advantage of various plug-ins and/or protocols such as HTML5, which may allow browser engine 122 to support at least some decoders and/or decoder functionalities natively. Browser engine 122 then renders a decoded version of the first encoded multi-media data utilizing operating system 221, frame buffer 222 and display 224 of I/O device 124, for example, as web page 125 comprising a JPEG image, a GIF image, and a JPEG-2000 image.

**[0065]** In this first implementation, browser engine 122 is configured to support WebAssembly or another similar platform-independent programming language that the accessor is coded in, so browser engine 122 is able to execute the accessor directly decode first encoded data utilizing the platform-independent first accessor, even though browser engine 122 does not natively support the filetype(s) that the first accessor is configured to decode. Moreover, because browser extension 126 acts as intermediary for browser engine 122, the functionality may be transparent to both browser engine 122 and to the user of user terminal 120, neither having to perform any function to manually acquire or run the first accessor. Instead, browser extension 126 transparently intercepts, initiates, and routes necessary communications to provide browser engine 122 access to both the retrieved first encoded data and the first accessor at a local storage location.

**[0066]** Functionality of user terminal 120 will now be described in connection with a second implementation, in which browser engine 122 is configured to natively support WebAssembly or another similar platform-independent programming language, and where the first encoded multi-media data and the first accessor are both intercepted by browser extension 126 embedded in a BVR container.

**[0067]** Browser engine 122 generates and transmits request 271 (e.g., an HTTP request) for first encoded data. In some embodiments, request 271 is addressed to first content server 140. Browser extension 126 is in communication with browser engine 122 and may intercept request

271. In response, browser extension 126 may reissue request 271, as request 272 (e.g., an HTTP request), to first content server 140 via WAN 170.

**[0068]** In response to receipt of request 272 from browser extension 126, reissued in response to request 271 from browser engine 122, first content server 140 may transmit a BVR container, having embedded therein the first encoded data and the first accessor, in response 273 (e.g., HTTP response) via WAN 170. Browser extension 126 may intercept the BVR container, having embedded therein the first encoded data and the first accessor, from first content server 140.

**[0069]** In some embodiments, browser extension 126 may be configured to determine whether response 273 comprises a BVR container having embedded therein the first encoded data and the first accessor. In response to a determination that response 273 comprises a BVR container, browser extension 126 may unpack one or more encoded data files and accessors from the container to be stored as separate files, and point the browser engine 122 to one or more local storage locations of the first encoded data and the first accessor, thereby facilitating browser engine 122 to subsequently render, process, or otherwise manipulate a decoded version of the first encoded data. For example, browser extension 126 may message the browser engine 122 over its browser extension API with information about the storage locations of the encoded data and the accessor(s). Browser engine 122 then decodes the first encoded data utilizing the first platform-independent accessor. Browser engine 122 then renders, processes, output, or otherwise manipulates a decoded version of the first encoded multi-media data. Using web page rendering as an example, the user terminal 120 may display data from the BVR container utilizing operating system 221, frame buffer 222, and display 224 of I/O device 124, for example, as some or all of a web page 125 comprising a JPEG image, a GIF image, and a JPEG-2000 image.

**[0070]** In this second implementation, as in the first implementation, because browser engine 122 is configured to support WebAssembly or another similar platform-independent programming language, browser engine 122 is able to directly decode first encoded data utilizing the platform-independent first accessor, even though browser engine 122 does not natively support the filetype(s) for which the first accessor is configured to decode. Moreover, in this second implementation, because browser extension 126 acts as intermediary for browser engine 122 and determines that response 273 includes a BVR container having embedded therein the first encoded data and the first accessor for decoding at least a portion of the first encoded data, no separate acquisition of the first accessor is required or performed by the user of the terminal 120 or the browser engine 122. Instead, browser extension 126 transparently intercepts, initiates, and routes necessary communications and then, based on a determination that response 273 already includes both the first encoded data and the first accessor, provides browser engine 122 access to both at a local storage location.

**[0071]** Functionality of user terminal 120 will now be described in connection with a third implementation, in which browser engine 122 is not configured to natively support WebAssembly or another similar platform-independent programming language in which platform independent accessors that are stored in BVR containers or in accessor server 130 are written. In the third implementation, similar to the second implementation, the first encoded data and the first accessor are both intercepted by browser extension 126 embedded in a BVR container.

**[0072]** In this implementation, the browser engine 122 generates and transmits request 271 (e.g., an HTTP request) for first encoded data. In some embodiments, request 271 is addressed to first content server 140. Browser extension 126 is in communication with browser engine 122 and may intercept request 271. In response, browser extension 126 may reissue request 271, as request 272 (e.g., an HTTP request), to first content server 140 via WAN 170.

**[0073]** In response to receipt of request 272 from browser extension 126, reissued in response to request 271 from browser engine 122, first content server 140 may transmit a BVR container, having embedded therein the first encoded data and the first accessor, in response 273 (e.g., HTTP response) via WAN 170. Browser extension 126 may intercept the BVR container, having embedded therein the first encoded data and the first accessor, from first content server 140.

**[0074]** In some embodiments, browser extension 126 may be configured to determine whether response 273 comprises a BVR container having embedded therein the first encoded data and the first accessor. In response to a determination that response 273 comprises a BVR container, browser extension 126 may unpack the data and the accessor from the container and route at least a portion of response 273, e.g., the first encoded data and the first accessor, to accessor processor 228. Accessor processor 228 is configured to instantiate the first accessor and decode at least a portion of the first encoded data utilizing the first accessor written in a platform-independent syntax. Accessor processor 228 may thus comprise a WebAssembly execution engine that may be coupled to or be a part of the browser extension 126. Accessor processor 228 may then deliver to browser extension 126 a decoded version of the first encoded data, as shown by the arrow extending from accessor processor 228 to browser extension 126. The browser extension 126 may then store the decoded data in local browser memory space, and may discard the accessor, or in some implementations as described above, store the accessor in the accessor cache 229 for later retrieval and/or use without requiring re-retrieval from the accessor server 130.

**[0075]** Browser extension 126 may then point browser engine 122 to one or more local storage locations of the decoded version of the first encoded data, thereby facilitating the browser engine 122 to subsequently render or otherwise process or manipulate a decoded version of the first encoded data. In some implementations, browser extension 126 may include web server like functionality to point the browser engine for decoded data retrieval. In this case, the browser

extension 126 may generate a redirect response that is sent to the browser engine 122 in response to the original data request 271 from the browser engine 122. This redirect response may include a URL or URI generated by the browser extension 126 for use by the browser engine in generating a subsequent request that is also intercepted by the browser extension 126. Thus, browser engine 122 may then transmit, and browser extension 126 may intercept, second request 279 (e.g., an HTTP request) for the decoded version of the first encoded data. Browser extension 126 then delivers the decoded data to browser engine 122 as if the browser engine retrieved it over the WAN. Browser engine 122 then renders, processes, or otherwise manipulates the decoded version of the first encoded data.

**[0076]** In this third implementation, if acting as an intermediary for browser engine 122, browser extension 126 determines that response 273 comprises a BVR container having embedded therein the first encoded data and the first accessor for decoding at least a portion of the first encoded data. Thus, as in the second implementation, no separate acquisition of the first accessor is required or performed. However, unlike the first and second implementations, browser engine 122 is not configured to support WebAssembly or another similar platform-independent programming language for interpreting the accessor syntax and so is not able to directly decode first encoded data utilizing the platform-independent syntax of the first accessor. Accordingly, browser extension 126 transparently routes the first encoded data and the first accessor to decoder processor 228, which instantiates the first accessor, decodes the first encoded data utilizing the first accessor, and forwards the decoded version of the first encoded data to browser extension 126. Browser extension 126 may then point browser engine 122 to the decoded version of the first encoded data at a local storage location, for subsequent rendering, processing, manipulation, etc..

**[0077]** Functionality of user terminal 120 will now be described in connection with a fourth implementation, in which browser engine 122 is, like the third implementation, not configured to natively support WebAssembly or another similar platform-independent programming language which the accessors are coded in.

**[0078]** Browser engine 122 generates and transmits request 271 (e.g., an HTTP request) for first encoded data. The first encoded data may comprise at least one file type that browser engine 122 does not natively support (e.g., Flash, XLS, or JPEG-2000 files). In some embodiments, request 271 is addressed to first content server 140.

**[0079]** Browser extension 126 is in communication with browser engine 122 and may intercept request 271. In response, browser extension 126 may reissue request 271, as request 272 (e.g., an HTTP request), to first content server 140 via WAN 170.

**[0080]** In response to receipt of request 272 from browser extension 126, reissued in response to request 271 from browser engine 122, first content server 140 may transmit the first encoded

data in response 273 (e.g., HTTP response) via WAN 170. Browser extension 126 may intercept the first encoded data transmitted from first content server 140.

**[0081]** In some embodiments, browser extension 126 may be configured to determine whether the first encoded data is embedded in a BVR container together with a first accessor comprising platform-independent syntax for decoding at least a portion of the first encoded data and, if not, whether the first encoded data comprises at least one file type that is unsupported by browser engine 122, e.g., a file type which browser engine 122 is unable to natively decode. Examples of such BVR containers are described in connection with **FIGs. 4-11** below.

**[0082]** In response to a determination that the first encoded data is not embedded in a BVR container with the first accessor and that the first encoded data comprises at least one file type that is unsupported by browser engine 122, browser extension 126 may be configured to acquire the first accessor comprising a platform-independent syntax for decoding at least a portion of the first encoded data from remote accessor server 130. Acquiring the first accessor from accessor server 130 may comprise generating and transmitting request 274 to accessor server 130 and intercepting all or a portion of the first accessor in response 275 from accessor server 130. In some embodiments, the first accessor may be transmitted in response 275 embedded within a BVR container.

**[0083]** In some embodiments where the first accessor has already been previously stored in accessor cache 229, in response to the above determination(s), browser extension 126 may alternatively be configured to acquire the first accessor from local accessor cache 229.

**[0084]** As previously described in connection with the first implementation above, the ability to alternatively acquire the first accessor from remote accessor server 130 via WAN 170 or from local accessor cache 229 when stored therein may allow for balancing between a required size of accessor cache 229 and an amount of data transmitted via WAN 170.

**[0085]** Browser extension 126 may route at least a portion of response 273, e.g., the first encoded data, and at least a portion of response 275, e.g., the first accessor), to accessor processor 228. Accessor processor 228 is configured to instantiate the first accessor and decode at least the unsupported filetype of the first encoded data utilizing the first accessor comprising the platform-independent syntax. Accessor processor 228 then outputs a decoded version of the first encoded data. The browser extension 126 may then point the browser engine 122 to the decoded data in any manner described for the other embodiments above.

**[0086]** In this fourth implementation, if acting as an intermediary for browser engine 122, browser extension 126 determines that the first encoded data of response 273 does not comprise a BVR container but does comprise at least one filetype that browser engine 122 does not natively support. Thus, as in the first implementation, acting as an intermediary for browser engine 122,

browser extension 126 transparently requests and intercepts the first accessor comprising the platform-independent syntax for decoding at least the unsupported filetype of first encoded data. However, unlike the first and second implementations, browser engine 122 is not configured to support WebAssembly or another similar platform-independent programming language and so is not able to directly decode at least the unsupported file type of first encoded data utilizing the platform-independent algorithm of the first accessor. Accordingly, browser extension 126 transparently routes the first encoded data and the first accessor to accessor processor 228, which instantiates the first accessor, decodes the first encoded data utilizing the first accessor, and forwards the decoded version of the first encoded data to browser extension 126. Browser extension 126 then transparently routes browser engine 122 to the decoded version of the first encoded data at a local storage location, for subsequent rendering, processing, or other manipulation.

**[0087]** Potential additional and/or alternative features of the above-described implementations will now be described.

**[0088]** In some embodiments, a BVR container intercepted from first content server 140 in response 273 (see e.g., second and third implementations described above) may be a partial-feature BVR container. For example, such a BVR container may comprise the first encoded data and the first accessor. However, the first accessor may include a platform-independent program for decoding only a portion of the first encoded data, while a second accessor, including different/additional platform-independent code for decoding additional features of the first encoded data, that is not included in the BVR container would need to be acquired before full decoding of the first encoded data may be accomplished. Accordingly, browser extension 126 may be configured to determine whether the BVR container is such a partial-feature BVR container (e.g., whether the first encoded data comprises at least one feature unsupported by the first accessor embedded in the BVR container) and, if so, browser extension 126 may be configured to acquire a second accessor comprising second platform-independent syntax for the at least one additional feature not supported by the first accessor in the BVR container. Acquiring the second accessor from accessor server 130 may comprise generating and transmitting a request 291 to accessor server 130 and intercepting all or a portion of the second accessor in a response 292 from accessor server 130.

**[0089]** In some embodiments, browser extension 126 is configured to analyze at least one of browser permissions and user preferences to generate one or both of request 274 for the first accessor and request 291 for the second accessor as previously described. In some embodiments, browser extension 126 is configured to perform at least one of an integrity check, data rights management, software rights management, decryption, and virus detection on at least one received response, for example, response 273 (either comprising a BVR container having embedded therein

the first encoded data and the first accessor or comprising the first encoded data), response 275 (comprising the first accessor), or response 292 (comprising the second accessor).

**[0090]** The systems and methods described herein provide much more flexibility and convenience with respect to these issues than conventional systems. Much of this flexibility is provided by the fact that the different platform-independent accessors available from the database will advantageously differ not just by what file type they can decode but also by quality metrics, data manipulation capabilities, and any number of other parameters. For example, the Chrome browser includes a built-in pdf file accessor. However, this pdf accessor has limited functionality. If the user obtains a pdf file while browsing with Chrome and wants to manipulate that document in more extensive ways than the Chrome pdf file accessor allows, the user must download the pdf file to a local operating system folder and re-open it with a local Adobe Acrobat program that is not platform independent and that had to be separately acquired by the user. This local application program has data manipulation features that are commensurate in scope with the license to that program purchased by the user when it was acquired. With the browser configuration and platform independent accessor server 130 described herein, multiple pdf accessors of varying capability can be stored or assembled for delivery by the platform independent accessor server 130. Furthermore, user rights and preferences with respect to data and its use can be delivered to the browser or browser extension in many ways, including via log-in credentials, system configuration monitoring by the browser, or obtained by the browser or browser extension from the content server when the data is retrieved. This information can be used by the browser to retrieve an accessor from the accessor server 130 that is not only appropriate for the file type retrieved from the content server but also for the license permissions applicable to the user and any particular user preferences, enterprise preferences, security preferences, etc. that may be input to or otherwise known to the browser.

**[0091]** In some embodiments, browser extension 126 may be configured to redirect request 272 to a server different from a server to which request 271 is originally addressed by browser engine 122. For example, request 271 for first encoded data may alternatively be addressed to, and initially request the first encoded data from, second server 150. However, browser extension 126 may be aware that first server 140 is currently storing the first encoded data or a BVR container having the first encoded data therein or may be aware of particular geographical or permission-based limitations associated with first server 140 or second server 150. In such embodiments, browser extension 126 may intercept first request 271 addressed to second server 150 from browser engine 122, and reformat first request 271, as request 272 addressed to first content server 140.

**[0092]** In some embodiments, browser extension 126 may be configured to generate and/or replace at least one embedded control within a received BVR container. An effect of such generation

and/or replacement may comprise a change in one or more of branding, a look-and-feel, link(s) to other site(s), ads, metadata or other host content embedded within the BVR container.

**[0093]** Several process flowcharts, corresponding to the above-described implementations, are described in more detail in connection with **FIGs. 3A-3C** below. Although particular steps, actions or blocks are described below. The present disclosure is not so limited and one or more additional or alternative steps, actions or blocks may also be utilized, one or more steps, actions or blocks may be omitted, or a different ordering may be utilized without departing from the scope of the disclosure.

**[0094]** **FIG. 3A** illustrates a flowchart 300 of a method of flexibly decoding and rendering a variety of digital data, in accordance with some embodiments. Flowchart 300 may correspond to operations of browser extension 126 for at least the first and second implementations described above, wherein browser engine 122 is configured to support BVR containers natively (e.g., browser engine 122 is configured to support WebAssembly or another similar platform-independent programming language).

**[0095]** Flowchart 300 begins at the START block and proceeds to block 302, which includes intercepting a first request for first encoded data from a browser engine. For example, as previously described in connection with **FIGs. 1** and **2**, browser engine 122 generates and transmits request 271 for first encoded data and browser extension 126 may intercept request 271 from browser engine 122.

**[0096]** Flowchart 300 proceeds to either block 304 or block 306, where browser extension 126 either reissues or reroutes the first request to a first content server. For example, as previously described in connection with **FIGs. 1** and **2**, if request 271 was originally addressed to first content server 140, browser extension 126 may reissue request 271, as request 272, to first content server 140 via WAN 170. On the other hand, if request 271 was originally addressed to a particular server (e.g., second content server 150) and browser extension 126 determines the request should be sent to a different server (e.g., first content server 140), browser extension 126 may reroute request 271, as request 272, to first content server 140 via WAN 170.

**[0097]** Flowchart 300 proceeds to block 308, which includes intercepting first encoded data transmitted in response to the first request from the browser engine. For example, as previously described in connection with **FIGs. 1** and **2**, first content server 140 may transmit the first encoded data in response 273 via WAN 170 and browser extension 126 may intercept the first encoded data transmitted from first content server 140. In embodiments where browser extension 126 is not configured to intercept requests from browser engine 122, flowchart 300 may advance from the START block directly to block 308.

**[0098]** Flowchart 300 proceeds to block 310, which includes determining whether first encoded data is embedded in a BVR container with a first accessor. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may be configured to determine whether the first encoded data is embedded in a BVR container together with a first accessor comprising platform-independent syntax for decoding at least a portion of the first encoded data. In response to a determination that first encoded data is embedded in a BVR container, flowchart 300 may proceed to block 312.

**[0099]** In response to a determination that first encoded data is not embedded in a BVR container, flowchart 300 may alternatively proceed to block 314, which includes determining whether the first encoded data comprises at least one file type unsupported by the browser engine. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may determine whether first encoded data comprises at least one file type that is unsupported by browser engine 122, e.g., a file type which browser engine 122 is unable to natively decode. In response to a determination that first encoded data does not comprise at least one file type unsupported by the browser engine, flowchart 300 may advance to block 322.

**[0100]** In response to a determination that first encoded data comprises at least one file type unsupported by the browser engine, flowchart 300 may advance to block 316, which includes generating and transmitting a request for a first accessor comprising platform-independent syntax for decoding at least a portion of the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may generate and transmit either request 274 to accessor server 130 or request 276 to local accessor cache 229.

**[0101]** Flowchart 300 advances to block 318, which includes intercepting the first accessor. For example, as previously described in connection with **FIGs 1** and **2**, browser extension 126 may intercept all or a portion of the first accessor in response 275 from accessor server 130 or in response 277 from local accessor cache 229. The operations of blocks 316 and 318 may, together, be considered acquiring the first accessor comprising platform-independent syntax for decoding at least a portion of the first encoded data.

**[0102]** Flowchart 300 advances to block 312, which includes determining whether the first encoded data comprises at least one feature unsupported by the first accessor. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may determine whether the first encoded datacomprises at least one feature unsupported by the first accessor. In some embodiments, this determination may be carried out in either condition where the first encoded data is embedded in a BVR container or where the first encoded data is not embedded in a BVR container.

**[0103]** In response to a determination that the first encoded data does not comprise at least one feature unsupported by the first accessor, flowchart 300 may advance to block 320, which includes pointing the browser engine to one or more local storage locations of the first encoded data and the first accessor. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may reroute browser engine 122 to one or more local storage locations of the first encoded data and the first accessor, thereby facilitating browser engine 122 to subsequently render or otherwise process or manipulate a decoded version of the first encoded data at block 322.

**[0104]** In response to a determination that the first encoded data comprises at least one feature unsupported by the first accessor, flowchart 300 may advance to block 324, which includes generating and transmitting a request for a second accessor. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may generate and transmit request 291 to accessor server 130 or a request similar to request 276 to local accessor cache 229.

**[0105]** Flowchart 300 proceeds to block 326, which includes intercepting the second accessor. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may intercept all or a portion of the second accessor in response 292 from accessor server 130 or in a response similar to response 277 from local accessor cache 229. The operations of blocks 324 and 326 may, together, be considered acquiring a second accessor implementing a second platform-independent syntax for the at least one additional feature not supported by the first accessor.

**[0106]** Flowchart 300 proceeds to block 328, which includes pointing the browser engine to one or more local storage locations of the first encoded data and the first and second accessors. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may point browser engine 122 to one or more local storage locations of the first encoded data and the first and second accessors, thereby facilitating browser engine 122 to subsequently render, process, or manipulate a decoded version of the first encoded data at block 322.

**[0107]** At block 322, the browser engine decodes and renders, processes, manipulates, or outputs the decoded version of the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser engine 122 may as one example, render a decoded version of the first encoded data utilizing, for example, one or more of operating system 221, frame buffer 222, and display 224 of I/O device 124, for example, as web page 125 comprising a JPEG image, a GIF image, and a properly decoded, interpreted and rendered JPEG-2000 image. Flowchart 300 may then advance to END block.

**[0108]** **FIG. 3B** illustrates another flowchart 350 of a method of flexibly decoding a variety of digitaldata, in accordance with some embodiments. Flowchart 350 may correspond to operations of browser extension 126 for at least the third and fourth implementations described above, wherein browser engine 122 is not configured to support BVR containers natively (e.g., browser engine 122

is not configured to support WebAssembly or another similar platform-independent programming language) and such decoding and/or interpreting of the first encoded data utilizing at least the first accessor is seamlessly and transparently handled by accessor processor 228 rather than by browser engine 122.

**[0109]** As can be seen by comparison of **FIGs. 3A** and **3B**, flowchart 350 includes some of the same blocks as flowchart 300. For example, blocks 302, 304, 306, 308, 310, 312, 314, 316, 318, 324 and 326 are substantially the same between flowchart 300 and flowchart 350. Accordingly, their descriptions will not be repeated here.

**[0110]** From block 314, in response to a determination by the browser extension that the first encoded data does not comprise at least one file type unsupported by the browser engine (and previously that the first encoded data was not embedded in a BVR container with the first accessor), flowchart 350 advances to block 340, which includes the browser engine decoding the first encoded data. For example, if the first encoded data does not include any file types or features unsupported by browser engine 122, browser engine 122 may itself decode the data. Flowchart 350 would then advance to block 366.

**[0111]** Jumping back to block 312, in response to a determination that the first encoded data does not comprise at least one feature unsupported by the first accessor, flowchart 350 advances to block 330, which includes rerouting the first encoded data and the first accessor to an accessor processor configured to instantiate the first accessor and decode the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may reroute at least a portion of response 273, e.g., the first encoded data and the first accessor in a BVR container, or of responses 273 and 275, e.g., the first encoded data and the first accessor respectively, to accessor processor 228. Accessor processor 228 instantiates the first and second accessors and decodes at least a portion of the first encoded data utilizing the first platform-independent syntax of the first decoder and the second platform-independent syntax of the second decoder. Flowchart 350 advances to block 338.

**[0112]** Jumping back to block 326 of flowchart 350, where a second accessor is needed or desired for decoding at least a portion of the first data, upon intercepting the second accessor as previously described for flowchart 300, flowchart 350 advances to block 338, which includes routing the first encoded data and the first and second accessors to an accessor processor configure to instantiate the first and second accessors and decode the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may route at least a portion of response 273, e.g., the first encoded data and the first accessor in a BVR container, or of responses 273 and 275, e.g., the first encoded data and the first accessor respectively, and at least a portion of response 292, e.g., the second accessor, to accessor processor 228. Accessor

processor 228 instantiates the first and second accessors and decodes at least a portion of the first encoded data utilizing the first platform-independent syntax of the first accessor and the second platform-independent syntax of the second decoder. Flowchart 350 advances to block 338.

**[0113]** Block 338 includes intercepting the decoded version of the first encoded data from the accessor processor. For example, as previously described in connection with **FIGs. 1** and **2**, accessor processor 228 transmits, and browser extension 126 receives, a decoded version of the first encoded data, as shown by the arrow extending from accessor processor 228 to browser extension 126 in **FIG. 2**.

**[0114]** Flowchart 350 advances to block 334, which includes pointing the browser engine to one or more local storage locations of the decoded version of the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 points browser engine 122 to one or more local storage locations of the decoded version of the first encoded data, thereby facilitating the browser engine to subsequently render or otherwise process or manipulate a decoded version of the first encoded data.

**[0115]** **FIG. 3C** illustrates another flowchart 380 of a method of flexibly decoding and rendering a variety of digital data, in accordance with some embodiments. Flowchart 380 may correspond to operations of platform-independent accessor server 130 for at least some of the first through fourth implementations described above.

**[0116]** Flowchart 380 begins at the START block and proceeds to block 382, which includes parsing a first request for the first accessor from a browser extension, the first request being a reissued version of a prior request intercepted from a browser engine for the first encoded data. For example, as previously described in connection with **FIGs. 1** and **2**, browser decoder server 130 may comprise a request parser 232, which may comprise a general-purpose or special-purpose processor configured to receive and parse request 274.

**[0117]** Flowchart 380 proceeds to block 384, which includes generating a first response configured to route the first accessor to the browser extension. For example, as previously described in connection with **FIGs. 1** and **2**, the request parser processor of request processor 232 may generate response 275 including the first accessor stored in accessor database 234.

**[0118]** Flowchart 380 proceeds to block 386, which includes routing the first response comprising the first accessor to the browser extension. For example, as previously described in connection with **FIGs. 1** and **2**, accessor server 130 may transmit, and browser extension 126 may intercept, response 275 including the first accessor. In some embodiments, the first accessor may be transmitted in response 275 embedded within a BVR container.

**[0119]** Flowchart 380 proceeds to block 388, which includes parsing a second request for a second accessor from the browser extension, the second request transmitted in response to a

determination by the browser extension that the first encoded data comprises at least one feature unsupported by the first accessor and the browser engine. For example, as previously described in connection with **FIGs. 1** and **2**, browser extension 126 may determine whether the first encoded data comprises at least one feature unsupported by the first accessor. In some embodiments, this determination may be carried out in either condition where the first encoded data is embedded in a BVR container or where the first encoded data is not embedded in a BVR container. Browser extension 126 may generate and transmit request 291 to accessor server 130. Accordingly, request parser 232 may parse request 291 for the second accessor from browser extension 126.

**[0120]** Flowchart 380 proceeds to block 390, which includes generating a second response configured to route the second accessor to the browser extension. For example, as previously described in connection with **FIGs. 1** and **2**, request parser 232 may generate response 292, which includes the second accessor therein.

**[0121]** Flowchart 380 proceeds to block 392, which includes routing the second response comprising the second accessor to the browser extension. For example, as previously described in connection with **FIGs. 1** and **2**, accessor server 130 may transmit, and browser extension 126 may intercept, response 292 including the second accessor. In some embodiments, the second accessor may be transmitted in response 292 embedded within a BVR container.

**[0122]** Although shown as a single flowchart, accessor server need not carry out blocks 382, 384 and 386 in order to carry out blocks 388, 390 and 392, and vice versa. For example, in implementations where response 273 from first content server 140 comprises a BVR file having embedded therein both the first encoded data and the first accessor, platform independent accessor server 130 need not transmit first accessor in response 275. Similarly, where first accessor is alternatively received from local accessor cache 229 in response 277, platform independent accessor server 130 need not transmit the first accessor in response 275. Likewise, even where blocks 382, 384 and 386 are carried out, a second accessor may not be needed to decode first encoded data, in which case, blocks 388, 390 and 392 need not be carried out. Similarly, where second accessor is needed or desired and is alternatively received from local accessor cache 229 in a response similar to response 277, platform-independent accessor server 130 need not transmit the second accessor in response 292.

**[0123]** Several examples of BVR containers (e..g, ISO-BMFF containers), as referred to in this specification, are described in more detail in connection with **FIGs. 4-11** below.

**[0124]** **FIG. 4** is a high-level illustration of a proposed ISO-BMFF container 401. The general concepts illustrated by **FIG. 4** may be selectively applied in any of the example container formats shown below in **FIGS. 5-10**. The boxes in **FIG. 4** show that additional non-traditional track “trak” boxes may be added to the container 401 to support digital data types such as, but not limited to,

text, rich text documents, presentations, and scientific data. This extends the ISO-BMFF format by providing for the embedding of multiple types of digital data in a container. That is, a container may consist of one or more of the illustrated track boxes.

**[0125]** The container 401 includes a movie ("moov") box 403 and a media data ("mdat") box 405. Generally, the ISO-BMFF specification defines a moov box as a container box whose sub-boxes define metadata for a presentation. The mdat box 405 is defined by the ISO-BMFF specification as a box that can hold actual media data for a presentation. The illustrated aspect of **FIG. 4** includes a moov box 403 that includes five trak boxes 407*a-e*. The number of trak boxes within the moov box may vary in other embodiments. Each track box includes an mdia box 408*a-e*. The ISO-BMFF standard traditionally defines a trak box to include a sequence of related samples for media data. Each mdia box 408*a-e* is defined by the ISO-BMFF specification to contain media information for a track. Some of the methods and systems disclosed herein propose to add additional data to each of the track boxes 407*a-e*. In some aspects, the proposed additional data may be included in only a subset of trak boxes included in the moov box 403.

**[0126]** The new data included in each of the trak boxes 407*a-e* includes checksum fields 410*a-e*, algorithm format fields 412*a-e*, and algorithm data indicator fields 414*a-e*. The checksum fields 410*a-e* may store hashes or checksums for algorithm defining data stored in the respective algorithm fields 414*a-e*. In some aspects, the hash or checksum fields 410*a-e* may indicate checksum values for both the algorithm data indicated by the respective algorithm field and also for content/presentation data for the associated trak, where the content and accessor algorithm are stored in the mdat box 405.

**[0127]** The algorithm format fields 412*a-e* may indicate a minimum version of an algorithm execution engine, such as a java virtual machine for Java based algorithms, common language runtime for C# algorithms, or other run-time interpreter, that is necessary to execute the algorithm indicated in the algorithm fields 414*a-e*. In some aspects, the algorithm format fields 412*a-e* may also indicate other dependencies associated with algorithms stored in the algorithm fields 414*a-e*. For example, input or output formats (such as RGB, YUV, etc), input/output functions of the algorithms stored in algorithm fields 414*a-e*, and/or data indicating possible optimizations and/or alternative functionality of the algorithms indicated in algorithm fields 414*a-e* may also be specified in the algorithm format fields 412*a-e*. For example, an accessor may define several methods for functions. Some of those functions may include platform specific functionality for decoding. In some aspects, the algorithm format fields 412*a-e* may store details of the data format being accessed.

**[0128]** In some aspects, the algorithm fields 414*a-e* indicate data implementing an algorithm. In some other aspects, the algorithm fields 414*a-e* identify a portion of the mdat box 405 that stores data implementing the algorithm. For example, the algorithm fields 414*a-e* may indicate a starting

location of data implementing an algorithm based on an offset from the beginning of the container 401. A length of respective algorithm implementing data stored in the mdat box 405 may also be indicated by the algorithm fields 414*a-e.*

**[0129]** Note that in some aspects, existing ISO-BMFF boxes, such as the xml box or the udat box, may contain or point to header, provider-supplied metadata, and accessor information. This option may override other required information in the container. This may not promote reusability of information between track boxes. Each track box may contain header information such as one or more checksums, e.g., MD5 or SHA, versioning information, and the index to the intermediate-format accessor algorithms.

**[0130]** **FIG. 5** shows an example implementation of an ISO-BMFF container. The container 501 includes one or more accessors for data contained within the container. As illustrated in the high level overview provided by **FIG. 4**, the ISO-BMFF container 501 includes a movie (moov) box 503 and a media data (mdat) box 505. The moov box 503 includes at least one trak box 507. The trak box 507 includes a mdia box 508. The mdia box includes a new box, provisionally called a Bevara Resource Indication box or "bvri" box 509. The bvri box 509 may be assigned any unique box identifier not already assigned to a known box type. The bvri box 509 includes an info entry that identifies an iteminfoentry 527 within an iinf box 523. The bvri box 509 also identifies (via the info entry) an iloc box 525 within a meta box 521 (discussed below).

**[0131]** The mdia box 508 also includes a media information container or minf box 511. The minf box includes information identifying a location of content 540 in the mdat box 505. For example, the minf box 511 may indicate an offset from the beginning of the container 501 where the content 540 begins, and a length of the content in the mdat box 505. The content 540 may include any type of content. For example, the content 540 may comprise audio and/or video data. In some aspects, the content 540 may include other types of content not traditionally included in an ISO-BMFF container, such as word processing data encoding a document (such as .doc, .docx, .pdf, .rtf files). In some aspects, the content 540 may encapsulate or include other container formats. For example, content 540 may define an ogg container, avi container, f4v or FLV (Flash® video), DivX® media format, any of the Microsoft® Office® document formats, etc.

**[0132]** The container 501 also includes a meta box 521. The meta box 521 includes the iinf box 523 and the iloc box 525. The iinf box 523 includes the item info entry 527. The item info entry 527 includes information about an accessor 535 for the content 540. For example, the item info entry 527 may include a hash of the content 540, such as a checksum, or other information about content 540.

**[0133]** In some aspects, the content 540 may include uncontained data. In these aspects, the accessor 535 may implement a decoder/interpreter for the uncontained data. For example, if content

540 contains document data, accessor 535 may implement a word processor for the document data. If the content 540 is video data, the accessor 535 may implement a video decoder for the video data.

**[0134]** As discussed above, in some other aspects, the content 540 may include data formatted in one or more container formats. In these aspects, the accessor 535 may implement an algorithm that can both extract data from the container format, and then also access or decode the data encapsulated or "contained" by that container. For example, if the content 540 contains ogg container data that encapsulates avi data, the accessor 535 may implement both an ogg extractor and an avi decoding algorithm.

**[0135]** The udat box 560 illustrates that some aspects of an enhanced container may include metadata 561 associated with content identified by the minf box 511 in the media box 508 (or in the case of content 640 below, content identified indirectly by box 610*a*, via an index and the actual location information in box 644*b*). In some aspects, the metadata 561 may be supplied by a user. The metadata 561 may be in the form of text, XML, or other typed data. Alternatively, or in addition, as shown in **FIGS. 9** and **10** below, metadata may itself have associated accessors.

**[0136]** The disclosed methods and systems may encode typed metadata (and XML which is really formatted text) by either including it in a udat bvri box, such as udat bvri box 561, or by pointing to it when the metadata is contained in the mdat box 505. When content or meta-data has an associated accessor, an additional bvriInfoEntry may be provided in the udat box as illustrated in **FIG. 10** below

**[0137]** The format disclosed in **FIG. 5** embeds a platform independent accessor in a container and also embeds the content to be accessed. The format may optionally embed metadata related to the content. If the content is multimedia content such as video, a traditional video player able to interpret MP4 containers may then parse the extended ISO-BMFF container shown in **FIG. 5**, and may ignore boxes 509, 527, 525, 535, and 561. This traditional video player will use its own built-in video decoder to decode the content in box 540. Thus, the format shown in **FIG. 5** is backwards compatible to existing players when the content in box 540 conforms with the ISO-BMFF specification at the time of filing the present application (for example, audio content, image content, video content).

**[0138]** **FIG. 6** shows another example of a proposed ISO-BMFF container format 601. Similar to **FIGS. 4** and **5**, the ISO-BMFF container 601 includes a movie (moov) box 603 and a media data (mdat) box 605. The moov box 603 includes at least one trak box 607. The trak box 607 includes an mdia box 608. The mdia box includes a new "bvri" box 609, similar to bvri box 509 discussed above. The bvri box 609 may be assigned any unique box identifier not already assigned to a known box type.

**[0139]** The bvri box 609 includes two bvri info entries 610*a* and 610*b*. In some aspects, the two bvri info entries 610*a-b* may be an array of bvriInfoEntry boxes or an array of info inside one bvriInfoEntry. The first bvri info entry 610*a* identifies an item info entry 634*a*. The item info entry 634*a* is located within an iinf box 632. The iinf box 632 is located within a meta box 621. The item info entry 634*a* includes information relating to accessor data 635. For example, the item info entry 634*a* may include checksum information for the accessor data 635. In some aspects, the item info entry 634*a* may include version information for the accessor data 635. For example, the version information may indicate a minimum version of an accessor execution engine that can be used to invoke an algorithm implemented by the accessor data 635. Accessor data 635 implements one or more algorithms for accessing content 640.

**[0140]** The first bvri info entry 610*a* also identifies a iloc box 644*a* within the meta box 621. The iloc box 644*a* includes information identifying accessor data 635 within the mdat box 605. For example, the iloc box 644*a* may indicate a starting offset for accessor data 635 relative to the beginning of the container 601. This starting offset may fall within the mdat box 605. The iloc box 644*a* may also indicate a length of the accessor data 635 within the mdat box 605.

**[0141]** The second bvri info entry 610*b* identifies an item info entry 634*b* stored within the iinf box 632. The item info entry 634*b* includes information relating to content 640. The second bvri info entry 610*b* also identifies a second iloc box 644*b*. The second iloc box 644*b* identifies content 640 within mdat box 605. Similar to content 540 discussed with respect to **FIG. 5**, content 640 may contain media data, for example, video, audio, data encoding a document, data encoding a container, or other form of digital data.

**[0142]** Accessor 635 relates to or is associated with content 640 because it is referenced by the same trak 607 as content 640. A decoder of the container 601 may invoke accessor 635 on content 640 based on the accessor 635 and the content 640 being indicated by the same trak 607. In the disclosed embodiments, a container may be structured to relate one accessor with multiple content(s) by referencing the one accessor from multiple tracks (trak boxes).

**[0143]** The trak box 607 also includes a udat box 660, and a brvi entry 661. In the illustrated embodiment, the bvri entry 661 indicates metadata relating to the content stored in content box 640 and associated with the trak box 607. In the example of udat box 660, a type of the metadata 661 may be text, xml, image, or any type.

**[0144]** Similar to the container 501 of **FIG. 5**, the container format 601 disclosed in **FIG. 6** also embeds a platform-independent accessor in a container with the content to be accessed. The container may also optionally embed metadata pertaining to content 640. Whereas **FIG. 5** utilizes the minf box 511 to identify content 540, note that the embodiment of **FIG. 6** utilized the iloc box 644*b* to identify content 640; see boxes 660 and 661 corresponding to boxes 560 and 561 of **FIG.**

**5**. Utilization of the minf box provides for fragmenting of content into samples or access units using stbl boxes as described by the ISO-BMFF specification. Each sample may have associated metadata (timing and content information).

**[0145]** **FIG. 7** shows another example of a proposed ISO-BMFF container format 701. Similar to **FIGS. 4-6**, the ISO-BMFF container 701 includes a movie (moov) box 703 and a media data (mdat) box 705. The moov box 703 includes at least one trak box 707. While the trak box 707 is shown as including document type content, it should be understood that the format of the container provided in **FIG. 7** can support any content data type. The trak box 707 includes a mdia box 708. The mdia box includes a new "bvri" box 709, similar to bvri boxes 509, and/or 609 discussed above. The bvri box 709 may be assigned any unique box identifier not already assigned to a known box type. The bvri box 709 may include a first bvri info entry 710*a* that identifies an item info entry 734*a* within an iinf box 732 and an iloc box 744*a*, discussed below. The bvri box 709 may also include a second bvri info entry 710*b* that identifies a second item info entry 734*b* within the iinf box 732 and a second iloc box 744*b*.

**[0146]** The mdia box 708 also includes a media information container or minf box 711. The minf box 711 includes information identifying a location of content 740 in the mdat box 705. For example, the minf box 711 may indicate a starting position of the content 740 based on an offset from the beginning of the container 701. The minf box 711 may also indicate a length of the content 740 in the mdat box 705. The content 740 identified by the minf box 711 may be content corresponding to the track identified by the trak box 707. To retain compatibility with decoders that can access mp4 video, the embodiment of **FIG. 7** utilizes standard video container structures. For example, in some aspects, content 740 may be video content, and is identified via the minf box 711. In some other aspects, content 740 may be document content, audio content, a container, image content, or virtually any other digital typed data.

**[0147]** The ISO-BMFF container 701 also includes a meta box 721. The meta box 721 includes the iinf box 732 and the iloc boxes 744*a* and 744*b*. The iinf box 732 includes an item info entry 734*a*. The item info entry 734*a* includes information relating to accessor 735*a* (optionally including version and/or hash/checksum information as discussed above). The iinf box 732 also includes a second item info entry 734*b* that includes information relating to accessor 735*b*.

**[0148]** The meta box 721 also includes an iloc box 744*a*. The iloc box 744*a* identifies data in the mdat box 705 implementing an accessor 735*a* for content 740. For example, the iloc box 744*a* may identify accessor 735*a* via an offset from the beginning of the container 701 and length within the mdat box 705. The meta box 721 also includes an iloc box 744*b*. The iloc box 744*b* identifies data in the mdat box 705 defining an accessor 735*b* for content 740. The embodiment of **FIG. 7**

shows that more than one accessor may be defined for the same content (content 740) when using various aspects of the proposed ISO-BMFF container formats.

**[0149]** The format disclosed in **FIG. 7** permits more than one platform-independent accessor to be packaged with their associated content. The accessors 735*a-b* are related to the content 740 through the bvri box 709 within the trak 707. This structure provides a concise and consistent packaging of the content 740 and the accessors 735*a-b*. To process the container format illustrated in **FIG. 7**, a decoding device (for example, browser engine 122, browser extension 126 or decoder processor 228) determines by reading the trak 707 and identifying at least two bvrInfoEntries 710*a-b* and associated content 740, that more than one accessor is packaged with the trak 707. In some aspects, the decoder may provide a user with an option of which one of the accessors should be executed on the associated content/data. In other aspects, both accessors 735*a-b* may be invoked automatically on the content 740. The container may also optionally contain embedded metadata pertaining to content 740; see boxes 760 and 761 corresponding to boxes 560 and 561 of **FIG. 5**.

**[0150]** **FIG. 8** shows another example of a proposed ISO-BMFF container format 801. Similar to **FIGS. 4-7**, the ISO-BMFF container 801 includes a movie (moov) box 803 and a media data (mdat) box 805. The moov box 803 includes at least one trak box 807. The trak box 807 includes a mdia box 808. The mdia box includes a new "bvri" box 809, similar to bvri boxes 509, 609, and/or 709 discussed above. The bvri box 809 may be assigned any unique box identifier not already assigned to a known box type. The bvri box 809 may include a bvri info entry 810 that identifies an item info entry 834*a* within iinf box 832 and an iloc box 844*a*, discussed below.

**[0151]** The mdia box 808 also includes a media information container or minf box 811*a*. The minf box 811*a* includes information that identifies content 840*a*. For example, the minf box 811*a* may indicate a starting position of content 840*a* based on an offset from the beginning of the container 801. The minf box 811*a* may also indicate a length of the content 840*a.*

**[0152]** The moov box 803 also includes a second trak box 857. The trak box 857 includes a second mdia box 858. The second mdia box 858 includes a new "bvri" box 859. The bvri box 859 may also be assigned any unique box identifier not already assigned to a known box type. The bvri box 859 has the same box identifier as bvri box 809. The bvri box 859 may include a bvri info entry 860 that identifies an item info entry 834*b* within the iinf box 832. The bvri info entry 860 within the bvri box 859 may also identify an iloc box 844*b*. The second bvri box 859 also includes a second minf box 811*b.*

**[0153]** The ISO-BMFF container 801 also includes a meta box 821. The meta box 821 includes the iinf box 832. The iinf box 832 includes the item info entry 834*a* and a second item info entry 834*b*. The item info entry 834*a* may include information relating to accessor 835*a* (such as hash

and/or version information as discussed previously). The second item info entry 834*b* includes information relating to accessor 835*b.*

**[0154]** The meta box 821 also includes the iloc box 844*a*. The iloc box 844*a* identifies data in the mdat box 805 defining an accessor 835*a* for content 840*a*. For example, the iloc box 844*a* may identify an offset from the beginning of the container 801 and length within the mdat box 805. The meta box 821 also includes an iloc box 844*b*. The iloc box 844*b* identifies data in the mdat box 805 implementing an accessor 835*b* for content 840*b*. The embodiment of **FIG. 8** shows that a single ISO-BMFF container may define multiple tracks, with each track identifying its own content and at least one accessor for that content. The accessor(s) 835*a* and 835*b* are specific to each track 807 and 857 respectively in **FIG. 8**. Note that while **FIG. 8** does not show metadata associated with either content 840*a* or 840*b*, some aspects of the container 801 may also associate metadata with one or both of contents 840*a-b*. For example, the metadata structures shown in **FIG. 9** and/or **FIG. 10** may also be used with the container 801 in some aspects.

**[0155]** The format disclosed in **FIG. 8** permits the packaging of more than one content (e.g. 840*a-b*) while also packaging platform-independent accessors 835*a-b* for each content 840*a-b*. Each accessor is related to its associated content through a bvri box within the trak. This provides a concise and consistent packaging of the multiple contents and their associated accessors.

**[0156]** To process the ISO-BMFF format illustrated in **FIG. 8**, an ISO-BMFF decoder 262 may read the moov box 803 and recognize that more than one trak, e.g. 807 and 857, are present. This indicates that more than one content is available and that more than one accessor is packaged. The decoding device, depending on configuration, may then either invoke both accessors so that they operate on their respective content. In some aspects, the decoding device may provide a user with an option to select which one of the accessors to invoke and use on its associated content/data.

**[0157]** In some aspects, the format of **FIG. 8** may encode embedded video (embedded content) in a powerpoint presentation (wrapper content) or embedded data graphics (embedded content) in a scientific document (wrapper content). One aspect of a decoder of the format of **FIG. 8** may enable a user to view/playback the presentation or document (wrapper content) via a corresponding second accessor until the embedding location is reached. When the playback reaches the location of the embedded content, the decoder may invoke a corresponding first accessor which operates on the embedded content. In some aspects, a pointer from the first content back to the secondary content trak in the ISO-BMFF container is provided (not shown). In some aspects, a pointer from the first track to the second trak may be provided, which provides an indication of where the embedded content and accessor for the embedded content is located.

**[0158]** **FIG. 9** shows another example of a proposed ISO-BMFF container format 901. Similar to **FIGS. 4-8**, the ISO-BMFF container 901 includes a movie (moov) box 903 and a media data

(mdat) box 905. The moov box 903 includes at least one trak box 907. The trak box 907 includes a mdia box 908. The mdia box 908 includes a new "bvri" box 909, similar to bvri boxes 509, 609, 709, and/or 809 discussed above. The bvri box 909 may be assigned any unique box identifier not already assigned to a known box type. The bvri box 909 may include a bvri info entry 910 that identifies an item info entry 934*a* within iinf box 932 and an iloc box 944*a*, discussed below.

**[0159]** The mdia box 908 also includes a media information container or minf box 911. The minf box includes information that identifies content 940*a*. In some aspects, content 940*a* may define a document, such as a word processing document (.doc, .txt, .rtf, etc), but may alternatively include any other type of content.

**[0160]** The trak box 907 also includes a user data or udat box 960. The udat box 960 includes a metadata bvri box 959 including two bvri info entries 912*a* and 912*b*. The bvri info entry 912*a* identifies an item info entry 934*b* within iinf box 932 and an iloc box 944*b*. The bvri info entry 912*b* identifies an item info entry 934*c* within iinf box 932 and an iloc box 944*c.*

**[0161]** The ISO-BMFF container 901 also includes a meta box 921. The meta box 921 includes the iinf box 932. The iinf box 932 includes an item info entry 934*a*, a second item info entry 934*b*, and a third item info entry 934*c*. The item info entry 934*a* includes information relating to accessor 935*a*. The iinf box 932 also includes a second item info entry 934*b* that includes information relating to accessor 935*b*. The iinf box 932 also includes a third item info entry 934*c* that includes information relating to metadata content 940*b.*

**[0162]** The meta box 921 also includes three iloc boxes 944*a-c*. The iloc box 944*a* identifies data in the mdat box 905 defining an accessor 935*a* for content 940*a*. For example, the iloc box 944*a* may identify an offset from the beginning of the container 901 identifying a starting position of accessor 935*a*, and a length within the mdat box 905 of accessor 935*a*. The meta box 921 also includes a iloc box 944*b*. The iloc box 944*b* identifies data in the mdat box 905 defining an accessor 935*b* for content 940*b*. The iloc box 944*c* identifies data in the mdat box 905 defining metadata content 940*b*, which may be audio metadata in some aspects, but may alternatively include any other type of content.

**[0163]** The embodiment of **FIG. 9** shows that complex metadata (identified by the udat box 960) may include multiple formats of metadata for one content. Metadata may not be limited to text or xml data. For example, audio or graphics data may be used to annotate content. For example, content 940*b* may annotate content 940*a* in the example of **FIG. 9**. Because the metadata identified in the udat box 960 is within the trak box 907, the metadata is associated with content associated with track box 907 (i.e. content 940*a* via minf box 911).

**[0164]** The format disclosed in **FIG. 9** provides for complex forms of metadata packaged and associated with particular content. This provides some degree of "future-proofing" of metadata, in a

similar manner that the main content is future proofed, i.e., an expanded ISO-BMFF container contains information to interpret/decode both the packaged content and the packaged metadata

**[0165]** When processing the proposed ISO-BMFF format illustrated in **FIG. 9**, a decoder 262 may reads the moov box 903 and recognize that the trak 907 "contains" (contains or points to) both content and metadata. By examining the udat metadata bvri box 959, the decoder can identify that some of the metadata is complex, that is, it also requires an accessor above and beyond the accessor required by the content 940*a*. The decoder, depending on configuration, may then either invokes both accessors such that they operate on their respective content simultaneously, or may provide a user with an option of displaying/playing the metadata in addition or, or instead of, the main content.

**[0166]** **FIG. 10** shows another example of a proposed ISO-BMFF container format 1001. Similar to **FIGS. 4-9**, the ISO-BMFF container 1001 includes a movie (moov) box 1003 and a media data (mdat) box 1005. The moov box 1003 includes at least one trak box 1007*a*. The trak box 1007*a* includes an mdia box 1008*a*. The mdia box 1008*a* includes a new "bvri" box 1009*a*, similar to bvri boxes 509, 609, 709, 809, and/or 909 discussed above. The bvri box 1009*a* may be assigned any unique box identifier not already assigned to a known box type. The bvri box 1009*a* may include a bvri info entry that identifies an item info entry 1034*a* within iinf box 1032 and an iloc box 1044*a*, discussed below.

**[0167]** The mdia box 1008*a* also includes a media information container or minf box 1011*a*. The minf box 1011*a* includes information that identifies content 1040*a*. In some aspects, content 1040*a* may define a document, such as a word processing document. (.doc, .txt, .rtf, etc), or may define any other type of content, for example, spreadsheets, presentations, figures, video, image, and/or audio data.

**[0168]** The trak box 1007*a* also includes a user data or udat box 1060. The udat box 1060 includes a bvri box 1061 including a bvri info entry 1012. The bvri info entry 1012 identifies a second track box 1007*b*.

**[0169]** The second track box 1007*b* includes a second mdia box 1008*b*. The second mdia box 1008*b* includes a second bvri box 1009*b*. This second bvri box 1009*b* includes one or more item info entries 1010*b* that identify a second item info entry 1034*b* within iinf box 1032 and a second iloc box 1044*b*. The second mdia box 1008*b* also includes a second minf box 1011*b* which includes information identifying metadata content 1040*b*.

**[0170]** The ISO-BMFF container 1001 also includes a meta box 1021. The meta box 1021 includes the iinf box 1032. The iinf box 1032 includes an item info entry 1034*a*, and a second item info entry 1134*b*. The item info entry 1034*a* includes information relating to accessor 1035*a* (for

example, versioning or hash information as discussed previously). The iinf box 1032 also includes a second item info entry 1034*b* that includes information relating to accessor 1035*b.*

**[0171]** The meta box 1021 also includes two iloc boxes 1044*a-b*. The iloc box 1044*a* identifies data in the mdat box 1005 defining an accessor 1035*a* for content 1040*a*. For example, the iloc box 1044*a* may identify an offset from a beginning of the container 1001 and length within the mdat box 1005 that includes data implementing an algorithm for accessing the content 1040*a*. The meta box 1021 also includes the iloc box 1044*b*. The iloc box 1044*b* identifies data in the mdat box 1005 defining an accessor 1035*b* for content 1040*b*. For example, the iloc box 1044*b* may identify a starting position for the accessor 1035*b* based on offset from a beginning of the container 1001. The iloc box 1044*b* may also indicate a length of accessor 1035*b.*

**[0172]** The embodiment of **FIG. 10** shows that complex metadata may identify multiple formats of metadata for one content 1040*a*. Metadata may not be limited to text or xml data. In this example, metadata is shown to include an audio annotation (provided by accessor 1035*b* operating on content 1040*b*) in addition to or in place of text/XML metadata (provided by udat box 1060).

**[0173]** The advantages of the format disclosed in **FIG. 10** are that complex forms of metadata can be packaged with the associated content. This provides some degree of future proofing of metadata in the same way that the main content is future proofed, i.e., that the expanded ISO-BMFF container holds all the information required to interpret/decode both the packaged content and the packaged metadata.

**[0174]** To process an ISO-BMFF format illustrated in **FIG. 10**, a decoder, such as browser engine 122, browser extension 126 or decoder processor 228) may read the moov box 1003 and determine that the trak 1007*a* "contains" (contains or points to) both content and metadata. By examining the udat metadata bvri box, the decoder may determine that some of the metadata is complex, that is, it also requires an accessor above and beyond the accessor required by the content 1040*a*. As opposed to the embodiment in **FIG. 9**, the trak structure 1007*a* identifies another trak as specific to metadata. This structure additionally provides for recursive metadata, e.g., the metadata can have its own metadata.

**[0175]** Furthermore, neither the embodiments of **FIG. 9** or **FIG. 10** are limited to a single non-text metadata. For instance, annotating audio and an author image may be included in the metadata. A decoding device, depending on configuration, may invoke both accessors such that they operate on their respective content, or can provide a user with the option of displaying/playing the metadata in addition or, or instead of, the main content.

**[0176]** Note that although distinct examples of container formats are provided in **FIGS. 5-10**, it should be understood that various features of each of **FIGS. 5-10** may be combined with other features disclosed in other of **FIGS. 5-10** where appropriate as one of skill in the art may judge. For

example, the bvriInfoEntry 610*b* iinf/iloc indicator of content may be used in place of the minf indicators within boxes 708, 808, 858, 908, 1008*a*, and 1008*b*. As another example, text or XML metadata may be packaged with a text or XML accessor, respectively, in the manner indicated for audio and other non-text content in **FIG. 9**. Similarly, a bvriInfoEntry may provide information on the link between several tracks (for instance a document could be stored on several tracks). In addition, a bvriInfoEntry could be used to point to extra information in the mdat for one particular track.

**[0177]** **FIG. 11** illustrates another example BVR container, but this time in the ZIP format. The ZIP format may be simpler than the ISO-BMFF format container described above, but all the same data and metadata described above can also be packaged in a ZIP format. The end of a ZIP file includes a Central Directory portion that includes a metadata block for each file in the ZIP file. These metadata blocks may include name, file start offset, and other information about each file. These blocks are denoted 1125a through 1125d in **FIG. 11**, corresponding to files 1120a, 1120b, 1130a, and 1130b. When used as a BVR container, the data files may comprise one or more encoded data files and one or more accessors similar to the ISO-BMFF format BVR containers. Each file 1120, 1130 is introduced by a file header 1105a, 1105b, 1105c, and 1105d. These headers may include file name and size. The ZIP format also allows the use of freely definable metadata fields in these headers, which may be used to store BVR container metadata such as set forth above with reference to the ISO-BMFF format containers.

## General Interpretive Principles for the Present Disclosure

**[0178]** Various aspects of the novel systems, apparatuses, and methods are described more fully hereinafter with reference to the accompanying drawings. The teachings disclosure may, however, be embodied in many different forms and should not be construed as limited to any specific structure or function presented throughout this disclosure. Rather, these aspects are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the disclosure to those skilled in the art. Based on the teachings herein one skilled in the art should appreciate that the scope of the disclosure is intended to cover any aspect of the novel systems, apparatuses, and methods disclosed herein, whether implemented independently of or combined with any other aspect of the disclosure. For example, a system or an apparatus may be implemented, or a method may be practiced using any one or more of the aspects set forth herein. In addition, the scope of the disclosure is intended to cover such a system, apparatus or method which is practiced using other structure, functionality, or structure and functionality in addition to or other than the various aspects of the disclosure set forth herein. It should be understood that any aspect disclosed herein may be set forth in one or more elements of a claim. Although some benefits and advantages of

the preferred aspects are mentioned, the scope of the disclosure is not intended to be limited to particular benefits, uses, or objectives. The detailed description and drawings are merely illustrative of the disclosure rather than limiting, the scope of the disclosure being defined by the appended claims and equivalents thereof.

**[0179]** With respect to the use of plural vs. singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

**[0180]** When describing an absolute value of a characteristic or property of a thing or act described herein, the terms "substantial," "substantially," "essentially," "approximately," and/or other terms or phrases of degree may be used without the specific recitation of a numerical range. When applied to a characteristic or property of a thing or act described herein, these terms refer to a range of the characteristic or property that is consistent with providing a desired function associated with that characteristic or property.

**[0181]** In those cases where a single numerical value is given for a characteristic or property, it is intended to be interpreted as at least covering deviations of that value within one significant digit of the numerical value given.

**[0182]** If a numerical value or range of numerical values is provided to define a characteristic or property of a thing or act described herein, whether or not the value or range is qualified with a term of degree, a specific method of measuring the characteristic or property may be defined herein as well. In the event no specific method of measuring the characteristic or property is defined herein, and there are different generally accepted methods of measurement for the characteristic or property, then the measurement method should be interpreted as the method of measurement that would most likely be adopted by one of ordinary skill in the art given the description and context of the characteristic or property. In the further event there is more than one method of measurement that is equally likely to be adopted by one of ordinary skill in the art to measure the characteristic or property, the value or range of values should be interpreted as being met regardless of which method of measurement is chosen.

**[0183]** It will be understood by those within the art that terms used herein, and especially in the appended claims (e.g., bodies of the appended claims) are intended as "open" terms unless specifically indicated otherwise (e.g., the term "including" should be interpreted as "including but not limited to," the term "having" should be interpreted as "having at least," the term "includes" should be interpreted as "includes but is not limited to," etc.).

**[0184]** It will be further understood by those within the art that if a specific number of an introduced claim recitation is intended, such an intent will be explicitly recited in the claim, and in

the absence of such recitation no such intent is present. For example, as an aid to understanding, the following appended claims may contain usage of the introductory phrases “at least one” and “one or more” to introduce claim recitations. However, the use of such phrases should not be construed to imply that the introduction of a claim recitation by the indefinite articles “a” or “an” limits any particular claim containing such introduced claim recitation to embodiments containing only one such recitation, even when the same claim includes the introductory phrases “one or more” or “at least one” and indefinite articles such as “a” or “an” (e.g., “a” and/or “an” should typically be interpreted to mean “at least one” or “one or more”); the same holds true for the use of definite articles used to introduce claim recitations. In addition, even if a specific number of an introduced claim recitation is explicitly recited, those skilled in the art will recognize that such recitation should typically be interpreted to mean at least the recited number (e.g., the bare recitation of “two recitations,” without other modifiers, typically means at least two recitations, or two or more recitations).

**[0185]** In those instances where a convention analogous to “at least one of A, B, and C” is used, such a construction would include systems that have A alone, B alone, C alone, A and B together without C, A and C together without B, B and C together without A, as well as A, B, and C together. It will be further understood by those within the art that virtually any disjunctive word and/or phrase presenting two or more alternative terms, whether in the description, claims, or drawings, should be understood to contemplate the possibilities of including one of the terms, either of the terms, or both terms. For example, the phrase “A or B” will be understood to include A without B, B without A, as well as A and B together.”

**[0186]** Various modifications to the implementations described in this disclosure can be readily apparent to those skilled in the art, and generic principles defined herein can be applied to other implementations without departing from the spirit or scope of this disclosure. Thus, the disclosure is not intended to be limited to the implementations shown herein but is to be accorded the widest scope consistent with the claims, the principles and the novel features disclosed herein. The word “exemplary” is used exclusively herein to mean “serving as an example, instance, or illustration.” Any implementation described herein as “exemplary” is not necessarily to be construed as preferred or advantageous over other implementations.

**[0187]** Certain features that are described in this specification in the context of separate implementations also can be implemented in combination in a single implementation. Conversely, various features that are described in the context of a single implementation also can be implemented in multiple implementations separately or in any suitable sub-combination. Moreover, although features can be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised

from the combination, and the claimed combination can be directed to a sub-combination or variation of a sub-combination.

**[0188]** The methods disclosed herein comprise one or more steps or actions for achieving the described method. The method steps and/or actions may be interchanged with one another without departing from the scope of the claims. In other words, unless a specific order of steps or actions is specified, the order and/or use of specific steps and/or actions may be modified without departing from the scope of the claims.

WHAT IS CLAIMED IS:

1. An apparatus for decoding data, the apparatus comprising:

a browser engine configured to issue requests for network resources hosted on servers using network resource identifiers and to receive network resources in response to the requests; and

a browser extension configured to:

intercept at least a first request issued from the browser engine for first encoded data associated with a first network resource identifier;

receive the first encoded data from a first server;

generate and issue a request for at least a first accessor associated with a second, different network resource identifier based at least in part on the intercepting and/or receiving;

receive the first accessor from a second server, wherein the first accessor comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data requested by the browser engine.

2. The apparatus of claim 1, wherein the browser extension is configured store the first encoded data and the at least first accessor in a portion of local browser memory space and to point the browser engine to the portion of local browser memory space containing the first encoded data and the first accessor.

3. The apparatus of any preceding claim, wherein the browser extension is configured to determine whether the first encoded data is embedded in a meta-file container that also contains an accessor comprising platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data requested by the browser engine.

4. The apparatus of any preceding claim, further comprising an accessor processor, wherein the accessor processor is configured to execute the first accessor to decode the first encoded data and store the decoded data in a portion of local browser memory space.

5. The apparatus of any preceding claim, wherein the pointing comprises sending a redirect response from the browser extension to the browser engine in response to intercepting the browser engine request for the first encoded data.

6. The apparatus of any preceding claim, wherein the browser extension is further configured to obtain a second accessor comprising platform-independent syntax implementing an algorithm for performing at least one additional decoding feature not supported by the first accessor from the second, different network.

7. The apparatus of any preceding claim, wherein the browser extension is further configured to reissue the first request, the first encoded data being received from the first server by the browser extension in response to the reissued first request.

8. The apparatus of any preceding claim, wherein the browser extension is further configured to generate a request for the first accessor based at least in part on at least one of browser permissions and user preferences.

9. The apparatus of any preceding claim, wherein the browser extension is further configured to perform at least one of a data rights management check, software rights management check, decryption, and virus detection on at least one of the first encoded data and the first accessor.

10. The apparatus of any preceding claim, further comprising an accessor cache in a portion of browser memory space.

11. A system for decoding data, the system comprising:

browser code configured for execution on general-purpose processing circuitry;

memory circuitry storing a plurality of accessors, wherein at least some of the plurality of accessors comprise platform-independent syntax implementing an algorithm for decoding encoded data;

wherein the browser code is configured to access the memory circuitry automatically without user intervention and retrieve one or more of the plurality of accessors in response to retrieving encoded data.

12. The system of claim 11, wherein the browser code is configured to access at least a portion of the memory circuitry over a WAN or LAN.

13. The system of claim 11 or 12, wherein at least a portion of the memory circuitry forms an accessor cache local to the general-purpose processor.

14. The system of any one of claims 11 to 13, wherein the browser code includes an accessor processor configured to decode retrieved encoded data using one or more of the accessors stored in the memory.

15. The system of any of claims 11 to 14, wherein the plurality of accessors includes accessors that differ by what file formats they decode.

16. The system of claim 15, wherein the plurality of accessors includes accessors that decode the same file formats but differ by their data manipulation capabilities.

17. A method of decoding digital data for manipulation by and/or output to a user of browser software, the method comprising:

receiving a user request to obtain a first network resource comprising first encoded digital data, wherein the first network resource is associated with a first network resource identifier;

using the first network resource identifier to generate and issue a request to obtain the first encoded data over a network;

receiving the first encoded data over the network;

without further user intervention, using a second different network resource identifier to generate and issue a request to obtain a first accessor over the network, wherein the first accessor comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data; and

receiving the first accessor over the network.

18. The method of claim 17, comprising storing the first encoded data and the first accessor in memory space allocated to the browser.

19. The method of claim 17 or 18, comprising decoding the first encoded data with the first accessor.

20. A method of decoding digital data for manipulation by and/or output to a user of browser software, the method comprising:

with a browser engine:

receiving a user request to obtain a first network resource comprising first encoded digital data, wherein the first network resource is associated with a first network resource identifier;

using the first network resource identifier to generate and issue a request to obtain the first encoded data over a network;

with a browser extension:

intercepting the request for the first encoded data;

determining an encoding method applied to the first encoded data;

using a second different network resource identifier to generate and issue a request to obtain a first accessor over the network, wherein the first accessor comprises platform-independent syntax implementing an algorithm configured to decode at least a portion of the first encoded data.

21. The method of claim 20, comprising decoding the first encoded data with the first accessor by an accessor processor that is part of or is coupled to the browser extension.

22. The method of claim 21, comprising storing the decoded first encoded data in a local memory location allocated to the browser software.

23. The method of claim 22, comprising pointing the browser engine to the local memory location where the decoded first encoded data is stored.

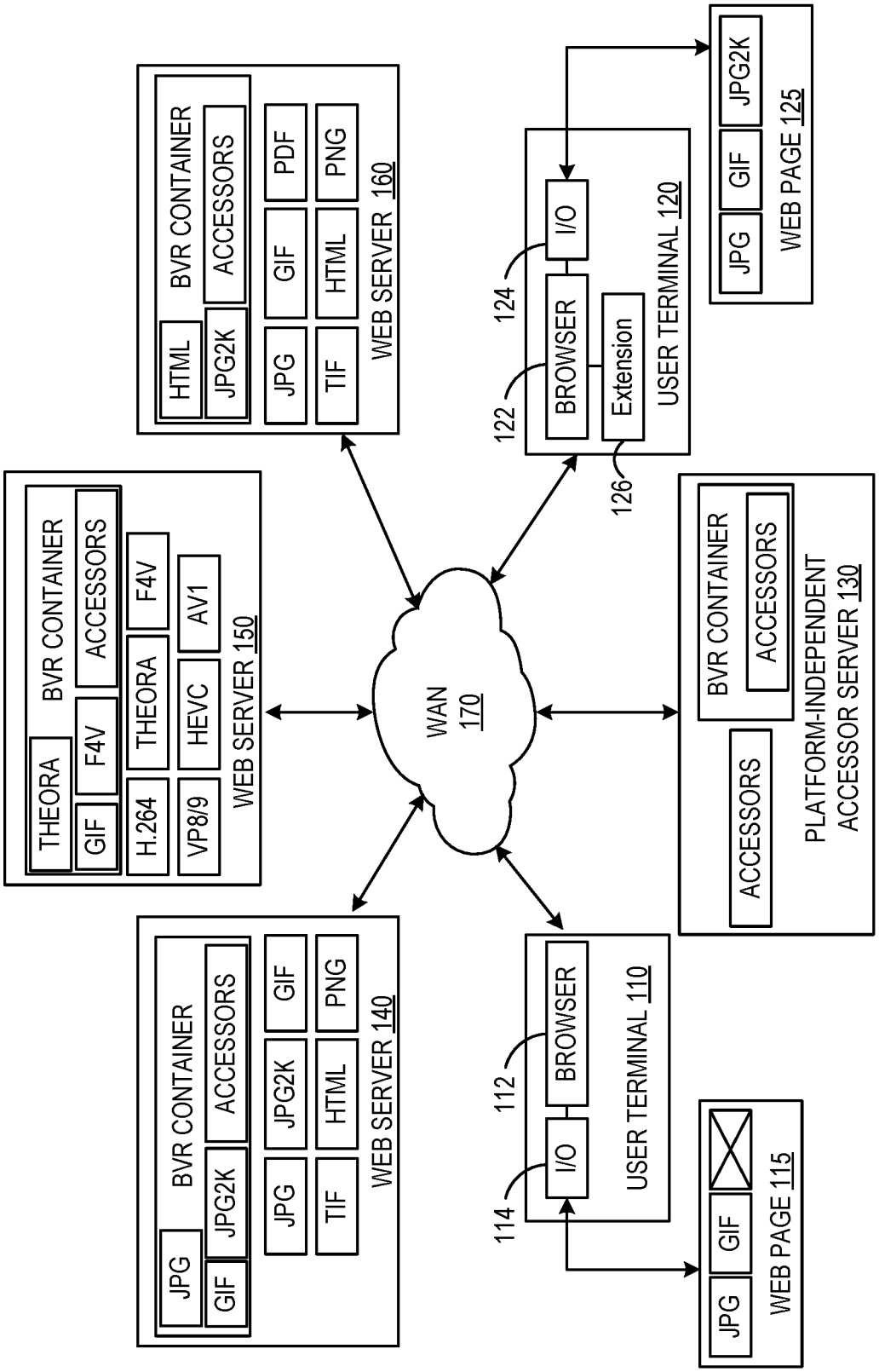

WEB SERVER 160
BVR CONTAINER
HTML
JPG2K
ACCESSORS
JPG
TIF
GIF
HTML
PDF
PNG
WEB SERVER 150
BVR CONTAINER
THEORA
GIF
F4V
ACCESSORS
H.264
VP8/9
THEORA
HEVC
F4V
AV1
WEB SERVER 140
BVR CONTAINER
JPG
GIF
JPG2K
ACCESSORS
JPG
TIF
JPG2K
GIF
HTML
PNG
WAN
170
USER TERMINAL 120
124
I/O
122
BROWSER
Extension
126
WEB PAGE 125
JPG
GIF
JPG2K
USER TERMINAL 110
112
BROWSER
114
I/O
WEB PAGE 115
JPG
GIF
PLATFORM-INDEPENDENT
ACCESSOR SERVER 130
BVR CONTAINER
ACCESSORS
ACCESSORS

FIG. 1

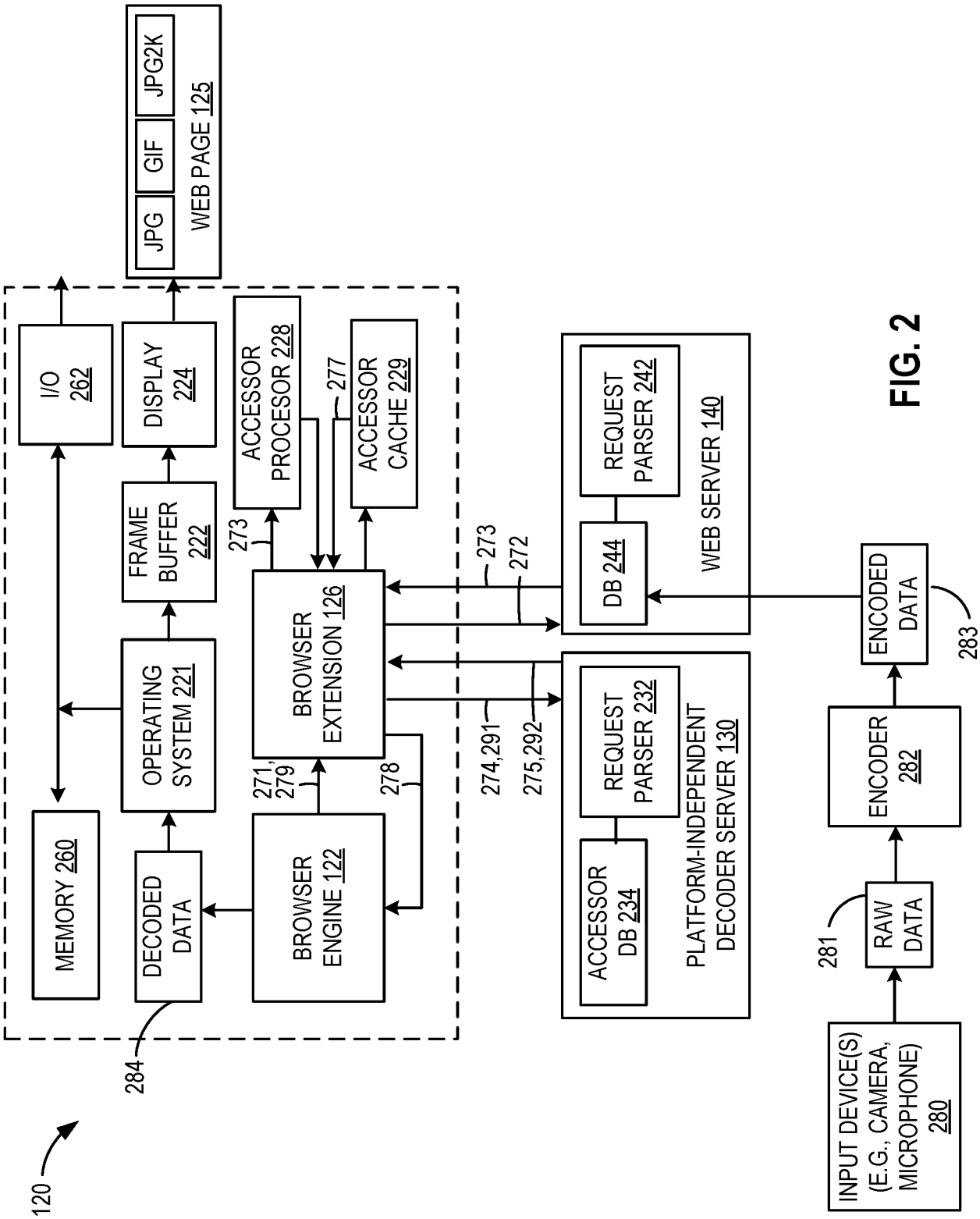

WEB PAGE 125
JPG
GIF
JPG2K
I/O 262
DISPLAY 224
FRAME BUFFER 222
ACCESSOR PROCESOR 228
ACCESSOR CACHE 229
277
273
OPERATING SYSTEM 221
BROWSER EXTENSION 126
271, 279
278
MEMORY 260
DECODED DATA
BROWSER ENGINE 122
284
120
274,291
275,292
273
272
WEB SERVER 140
REQUEST PARSER 242
DB 244
PLATFORM-INDEPENDENT DECODER SERVER 130
REQUEST PARSER 232
ACCESSOR DB 234
ENCODED DATA
283
ENCODER 282
281
RAW DATA
INPUT DEVICE(S) (E.G., CAMERA, MICROPHONE) 280

FIG. 2

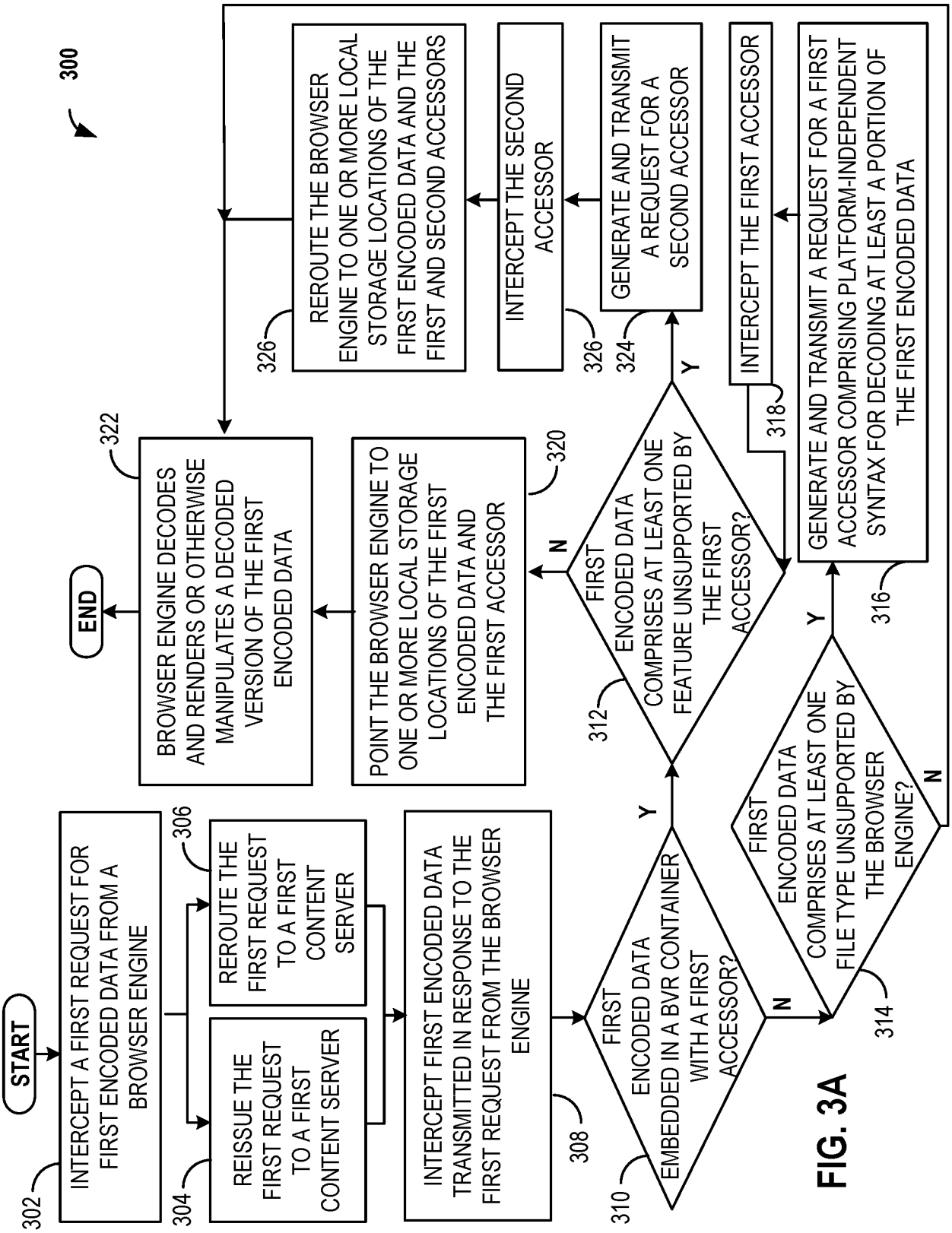

300
START
302
INTERCEPT A FIRST REQUEST FOR FIRST ENCODED DATA FROM A BROWSER ENGINE
304
REISSUE THE FIRST REQUEST TO A FIRST CONTENT SERVER
306
REROUTE THE FIRST REQUEST TO A FIRST CONTENT SERVER
INTERCEPT FIRST ENCODED DATA TRANSMITTED IN RESPONSE TO THE FIRST REQUEST FROM THE BROWSER ENGINE
308
FIRST ENCODED DATA EMBEDDED IN A BVR CONTAINER WITH A FIRST ACCESSOR?
310
Y
N
FIRST ENCODED DATA COMPRISES AT LEAST ONE FEATURE UNSUPPORTED BY THE FIRST ACCESSOR?
312
Y
N
FIRST ENCODED DATA COMPRISES AT LEAST ONE FILE TYPE UNSUPPORTED BY THE BROWSER ENGINE?
314
Y
N
GENERATE AND TRANSMIT A REQUEST FOR A FIRST ACCESSOR COMPRISING PLATFORM-INDEPENDENT SYNTAX FOR DECODING AT LEAST A PORTION OF THE FIRST ENCODED DATA
316
INTERCEPT THE FIRST ACCESSOR
318
GENERATE AND TRANSMIT A REQUEST FOR A SECOND ACCESSOR
324
INTERCEPT THE SECOND ACCESSOR
326
REROUTE THE BROWSER ENGINE TO ONE OR MORE LOCAL STORAGE LOCATIONS OF THE FIRST ENCODED DATA AND THE FIRST AND SECOND ACCESSORS
326
POINT THE BROWSER ENGINE TO ONE OR MORE LOCAL STORAGE LOCATIONS OF THE FIRST ENCODED DATA AND THE FIRST ACCESSOR
320
BROWSER ENGINE DECODES AND RENDERS OR OTHERWISE MANIPULATES A DECODED VERSION OF THE FIRST ENCODED DATA
322
END

FIG. 3A

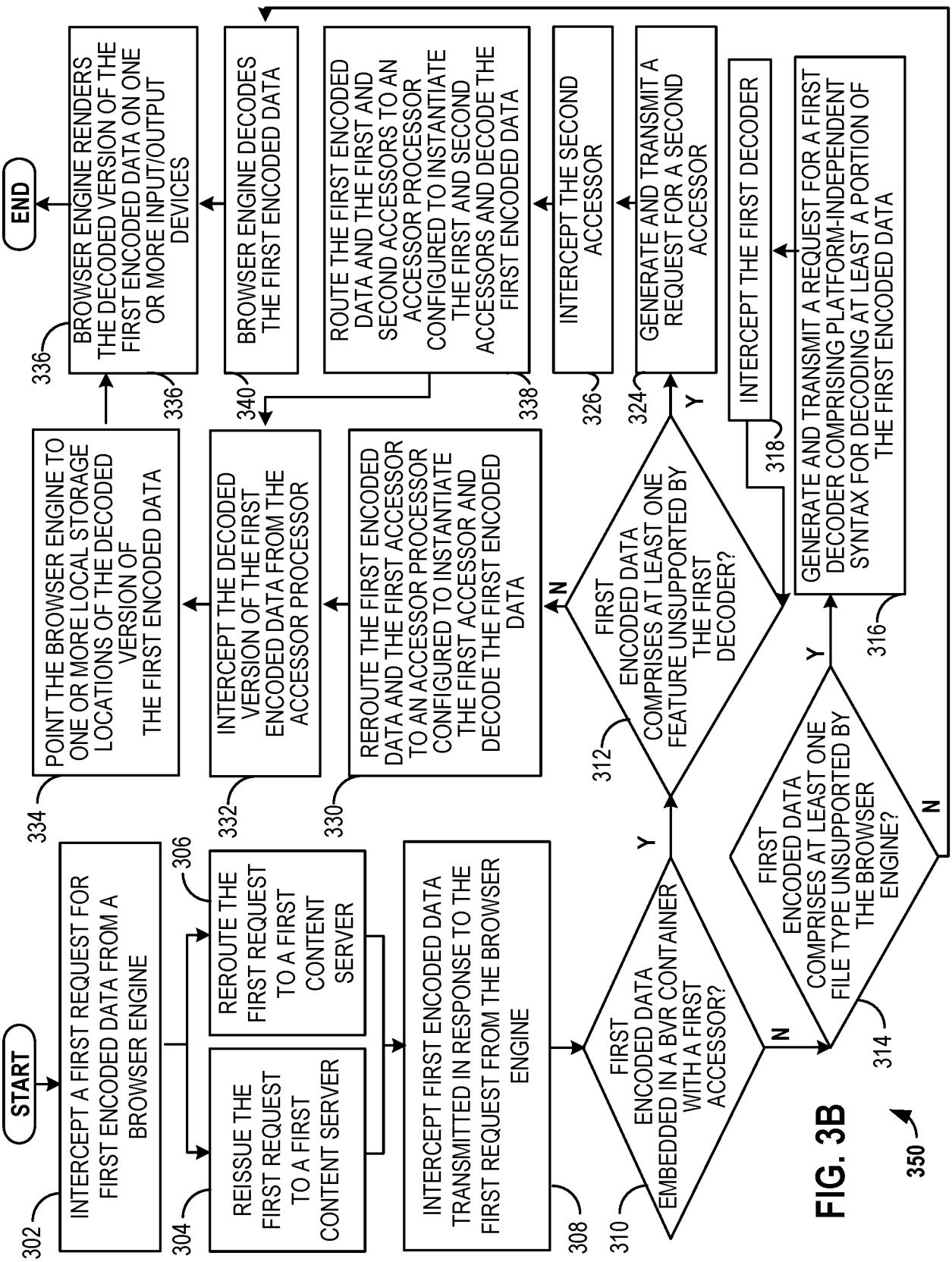

START
INTERCEPT A FIRST REQUEST FOR FIRST ENCODED DATA FROM A BROWSER ENGINE
302
REISSUE THE FIRST REQUEST TO A FIRST CONTENT SERVER
304
REROUTE THE FIRST REQUEST TO A FIRST CONTENT SERVER
306
INTERCEPT FIRST ENCODED DATA TRANSMITTED IN RESPONSE TO THE FIRST REQUEST FROM THE BROWSER ENGINE
308
FIRST ENCODED DATA EMBEDDED IN A BVR CONTAINER WITH A FIRST ACCESSOR?
310
Y
N
FIRST ENCODED DATA COMPRISES AT LEAST ONE FEATURE UNSUPPORTED BY THE FIRST DECODER?
312
Y
N
FIRST ENCODED DATA COMPRISES AT LEAST ONE FILE TYPE UNSUPPORTED BY THE BROWSER ENGINE?
314
Y
N
GENERATE AND TRANSMIT A REQUEST FOR A FIRST DECODER COMPRISING PLATFORM-INDEPENDENT SYNTAX FOR DECODING AT LEAST A PORTION OF THE FIRST ENCODED DATA
316
INTERCEPT THE FIRST DECODER
318
GENERATE AND TRANSMIT A REQUEST FOR A SECOND ACCESSOR
324
INTERCEPT THE SECOND ACCESSOR
326
ROUTE THE FIRST ENCODED DATA AND THE FIRST AND SECOND ACCESSORS TO AN ACCESSOR PROCESSOR CONFIGURED TO INSTANTIATE THE FIRST AND SECOND ACCESSORS AND DECODE THE FIRST ENCODED DATA
338
BROWSER ENGINE DECODES THE FIRST ENCODED DATA
340
REROUTE THE FIRST ENCODED DATA AND THE FIRST ACCESSOR TO AN ACCESSOR PROCESSOR CONFIGURED TO INSTANTIATE THE FIRST ACCESSOR AND DECODE THE FIRST ENCODED DATA
330
INTERCEPT THE DECODED VERSION OF THE FIRST ENCODED DATA FROM THE ACCESSOR PROCESSOR
332
POINT THE BROWSER ENGINE TO ONE OR MORE LOCAL STORAGE LOCATIONS OF THE DECODED VERSION OF THE FIRST ENCODED DATA
334
BROWSER ENGINE RENDERS THE DECODED VERSION OF THE FIRST ENCODED DATA ON ONE OR MORE INPUT/OUTPUT DEVICES
336
END
350

FIG. 3B

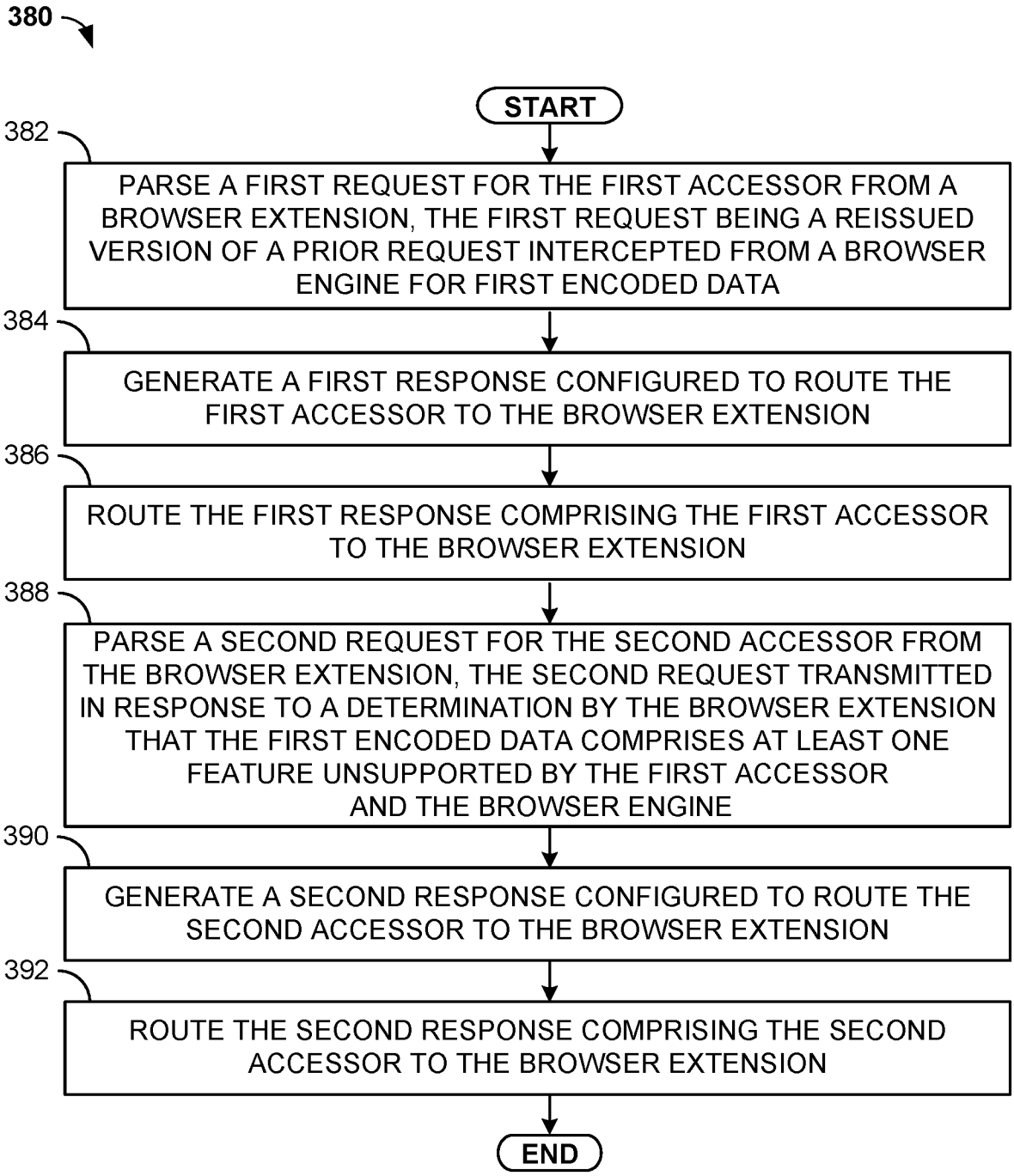

380
START
382
PARSE A FIRST REQUEST FOR THE FIRST ACCESSOR FROM A BROWSER EXTENSION, THE FIRST REQUEST BEING A REISSUED VERSION OF A PRIOR REQUEST INTERCEPTED FROM A BROWSER ENGINE FOR FIRST ENCODED DATA
384
GENERATE A FIRST RESPONSE CONFIGURED TO ROUTE THE FIRST ACCESSOR TO THE BROWSER EXTENSION
386
ROUTE THE FIRST RESPONSE COMPRISING THE FIRST ACCESSOR TO THE BROWSER EXTENSION
388
PARSE A SECOND REQUEST FOR THE SECOND ACCESSOR FROM THE BROWSER EXTENSION, THE SECOND REQUEST TRANSMITTED IN RESPONSE TO A DETERMINATION BY THE BROWSER EXTENSION THAT THE FIRST ENCODED DATA COMPRISES AT LEAST ONE FEATURE UNSUPPORTED BY THE FIRST ACCESSOR AND THE BROWSER ENGINE
390
GENERATE A SECOND RESPONSE CONFIGURED TO ROUTE THE SECOND ACCESSOR TO THE BROWSER EXTENSION
392
ROUTE THE SECOND RESPONSE COMPRISING THE SECOND ACCESSOR TO THE BROWSER EXTENSION
END

FIG. 3C

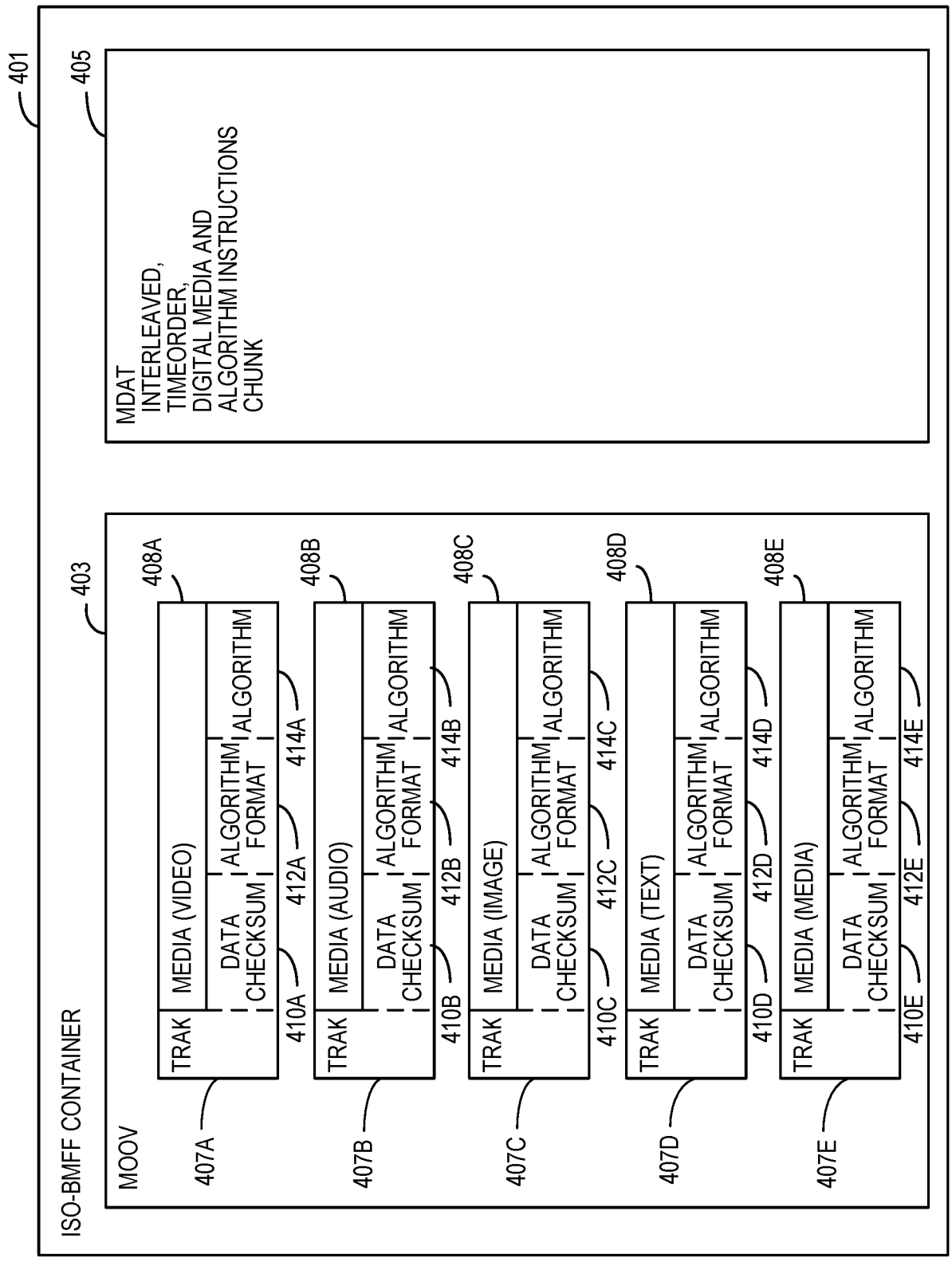

401
ISO-BMFF CONTAINER
403
MOOV
TRAK
MEDIA (VIDEO)
DATA
CHECKSUM
ALGORITHM
FORMAT
ALGORITHM
408A
410A
412A
414A
407A
TRAK
MEDIA (AUDIO)
DATA
CHECKSUM
ALGORITHM
FORMAT
ALGORITHM
408B
410B
412B
414B
407B
TRAK
MEDIA (IMAGE)
DATA
CHECKSUM
ALGORITHM
FORMAT
ALGORITHM
408C
410C
412C
414C
407C
TRAK
MEDIA (TEXT)
DATA
CHECKSUM
ALGORITHM
FORMAT
ALGORITHM
408D
410D
412D
414D
407D
TRAK
MEDIA (MEDIA)
DATA
CHECKSUM
ALGORITHM
FORMAT
ALGORITHM
408E
410E
412E
414E
407E
405
MDAT
INTERLEAVED,
TIMEORDER,
DIGITAL MEDIA AND
ALGORITHM INSTRUCTIONS
CHUNK

FIG. 4

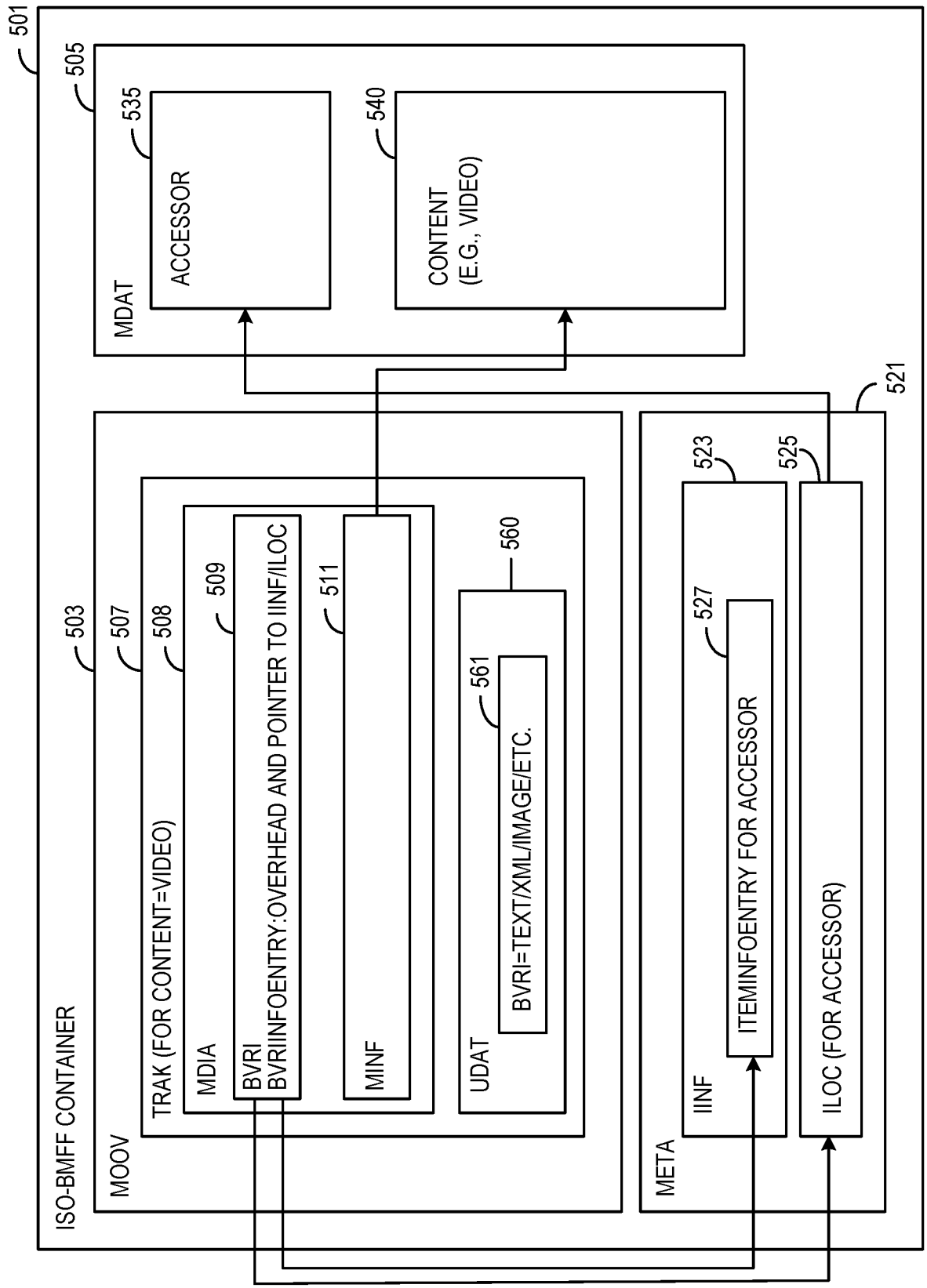

501
505
535
ACCESSOR
MDAT
540
CONTENT
(E.G., VIDEO)
521
503
507
508
509
511
560
561
BVRI=TEXT/XML/IMAGE/ETC.
BVRI
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC
MINF
UDAT
TRAK (FOR CONTENT=VIDEO)
MDIA
MOOV
ISO-BMFF CONTAINER
523
525
527
ITEMINFOENTRY FOR ACCESSOR
ILOC (FOR ACCESSOR)
IINF
META

FIG. 5

8/13

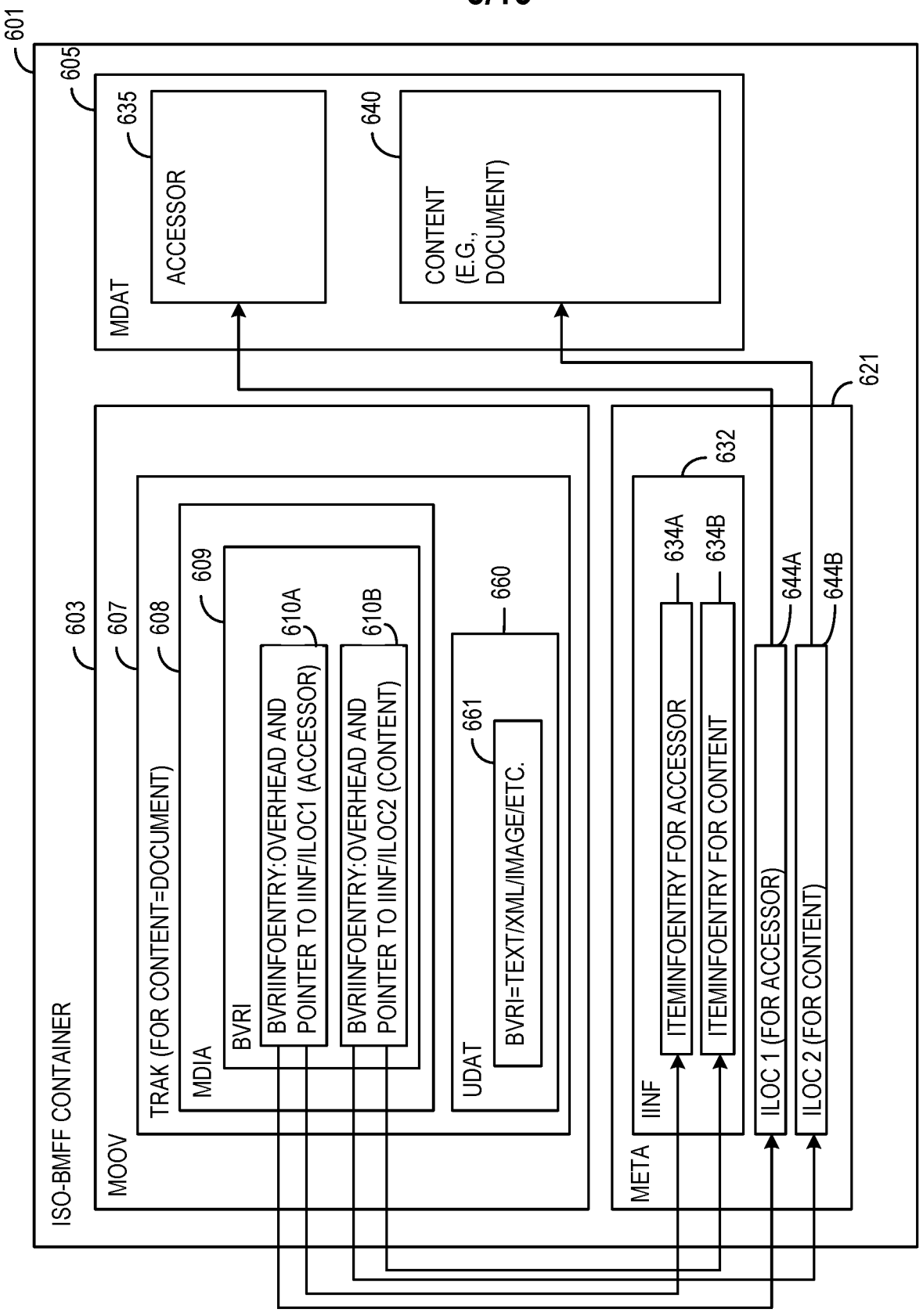

601
ISO-BMFF CONTAINER
605
MDAT
635
ACCESSOR
640
CONTENT (E.G., DOCUMENT)
621
603
MOOV
607
TRAK (FOR CONTENT=DOCUMENT)
608
MDIA
609
BVRI
610A
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC1 (ACCESSOR)
610B
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC2 (CONTENT)
660
UDAT
661
BVRI=TEXT/XML/IMAGE/ETC.
META
632
IINF
634A
ITEMINFOENTRY FOR ACCESSOR
634B
ITEMINFOENTRY FOR CONTENT
644A
ILOC 1 (FOR ACCESSOR)
644B
ILOC 2 (FOR CONTENT)

FIG. 6

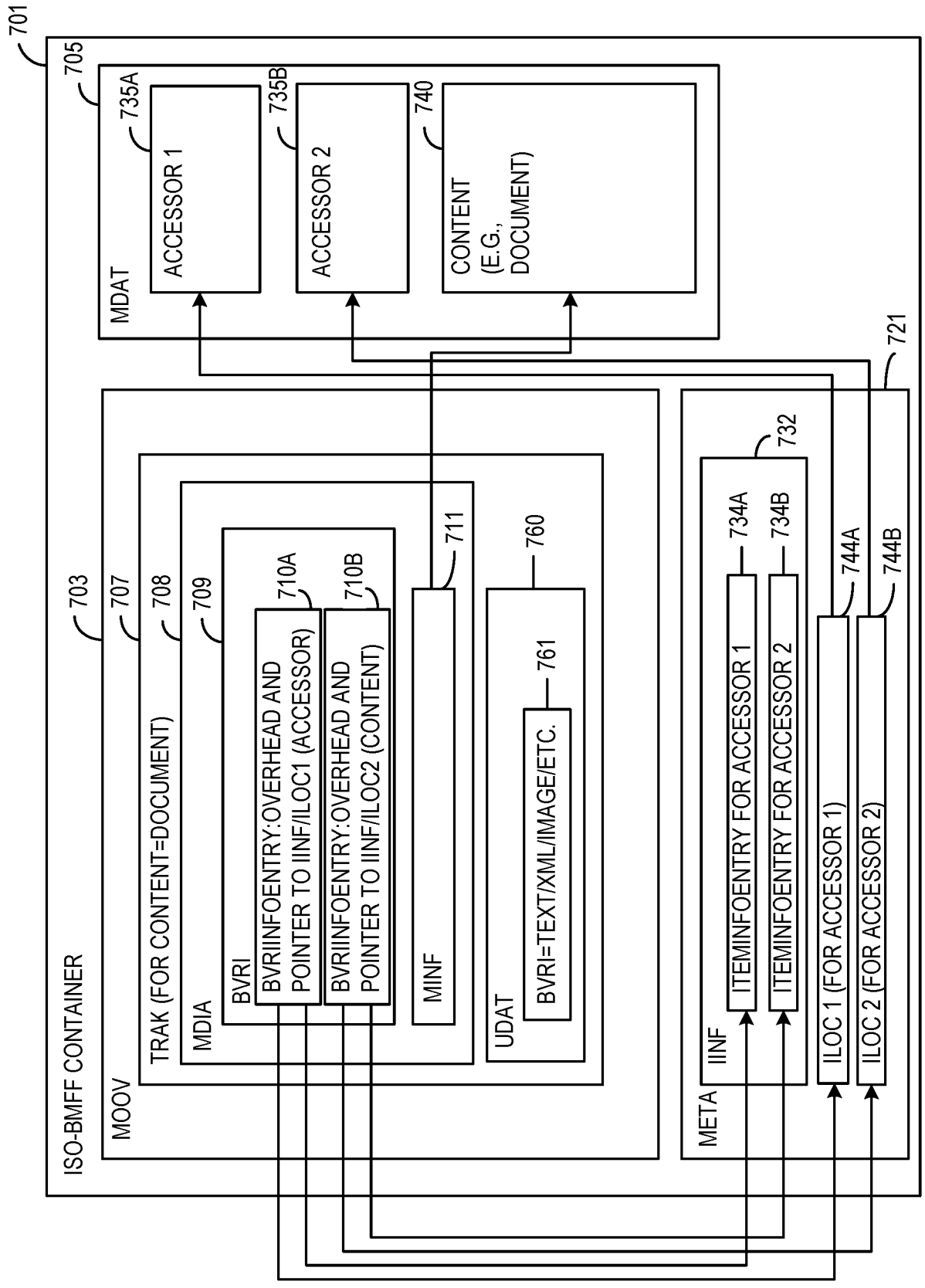

701
ISO-BMFF CONTAINER
705
MDAT
735A
ACCESSOR 1
735B
ACCESSOR 2
740
CONTENT (E.G., DOCUMENT)
721
703
MOOV
707
TRAK (FOR CONTENT=DOCUMENT)
708
MDIA
709
BVRI
710A
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC1 (ACCESSOR)
710B
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC2 (CONTENT)
711
MINF
760
UDAT
761
BVRI=TEXT/XML/IMAGE/ETC.
META
IINF
732
734A
ITEMINFOENTRY FOR ACCESSOR 1
734B
ITEMINFOENTRY FOR ACCESSOR 2
744A
ILOC 1 (FOR ACCESSOR 1)
744B
ILOC 2 (FOR ACCESSOR 2)

FIG. 7

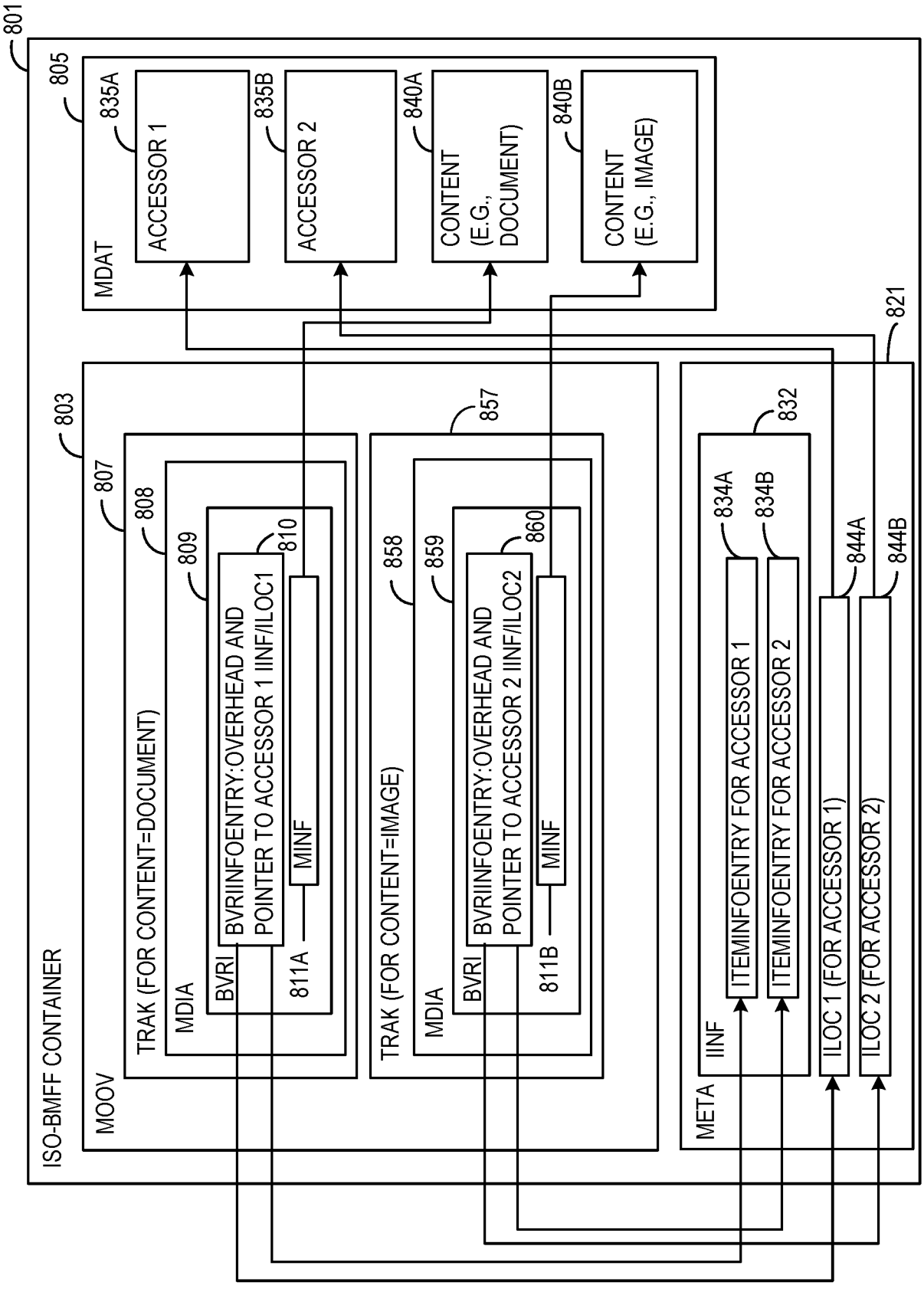

801
ISO-BMFF CONTAINER
805
MDAT
835A
ACCESSOR 1
835B
ACCESSOR 2
840A
CONTENT (E.G., DOCUMENT)
840B
CONTENT (E.G., IMAGE)
821
803
MOOV
807
TRAK (FOR CONTENT=DOCUMENT)
808
MDIA
809
BVRI
BVRIINFOENTRY:OVERHEAD AND POINTER TO ACCESSOR 1 IINF/ILOC1
810
MINF
811A
857
TRAK (FOR CONTENT=IMAGE)
858
MDIA
859
BVRI
BVRIINFOENTRY:OVERHEAD AND POINTER TO ACCESSOR 2 IINF/ILOC2
860
MINF
811B
META
832
IINF
834A
ITEMINFOENTRY FOR ACCESSOR 1
834B
ITEMINFOENTRY FOR ACCESSOR 2
844A
ILOC 1 (FOR ACCESSOR 1)
844B
ILOC 2 (FOR ACCESSOR 2)

FIG. 8

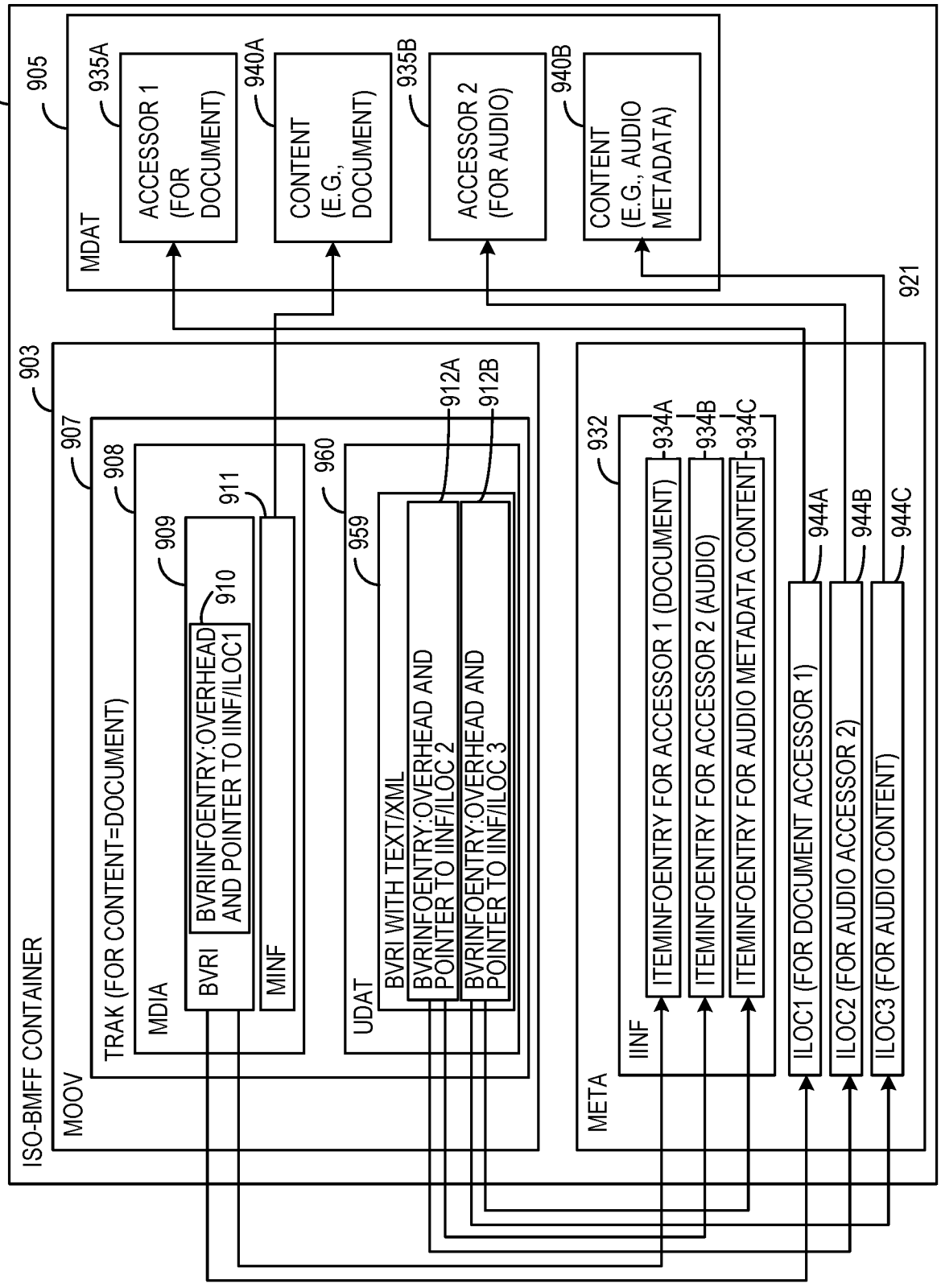

901
ISO-BMFF CONTAINER
905
MDAT
935A
ACCESSOR 1 (FOR DOCUMENT)
940A
CONTENT (E.G., DOCUMENT)
935B
ACCESSOR 2 (FOR AUDIO)
940B
CONTENT (E.G., AUDIO METADATA)
921
903
MOOV
907
TRAK (FOR CONTENT=DOCUMENT)
908
MDIA
909
BVRI
910
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC1
911
MINF
960
UDAT
959
BVRI WITH TEXT/XML
912A
BVRINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC 2
912B
BVRINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC 3
META
932
IINF
934A
ITEMINFOENTRY FOR ACCESSOR 1 (DOCUMENT)
934B
ITEMINFOENTRY FOR ACCESSOR 2 (AUDIO)
934C
ITEMINFOENTRY FOR AUDIO METADATA CONTENT
944A
ILOC1 (FOR DOCUMENT ACCESSOR 1)
944B
ILOC2 (FOR AUDIO ACCESSOR 2)
944C
ILOC3 (FOR AUDIO CONTENT)

FIG. 9

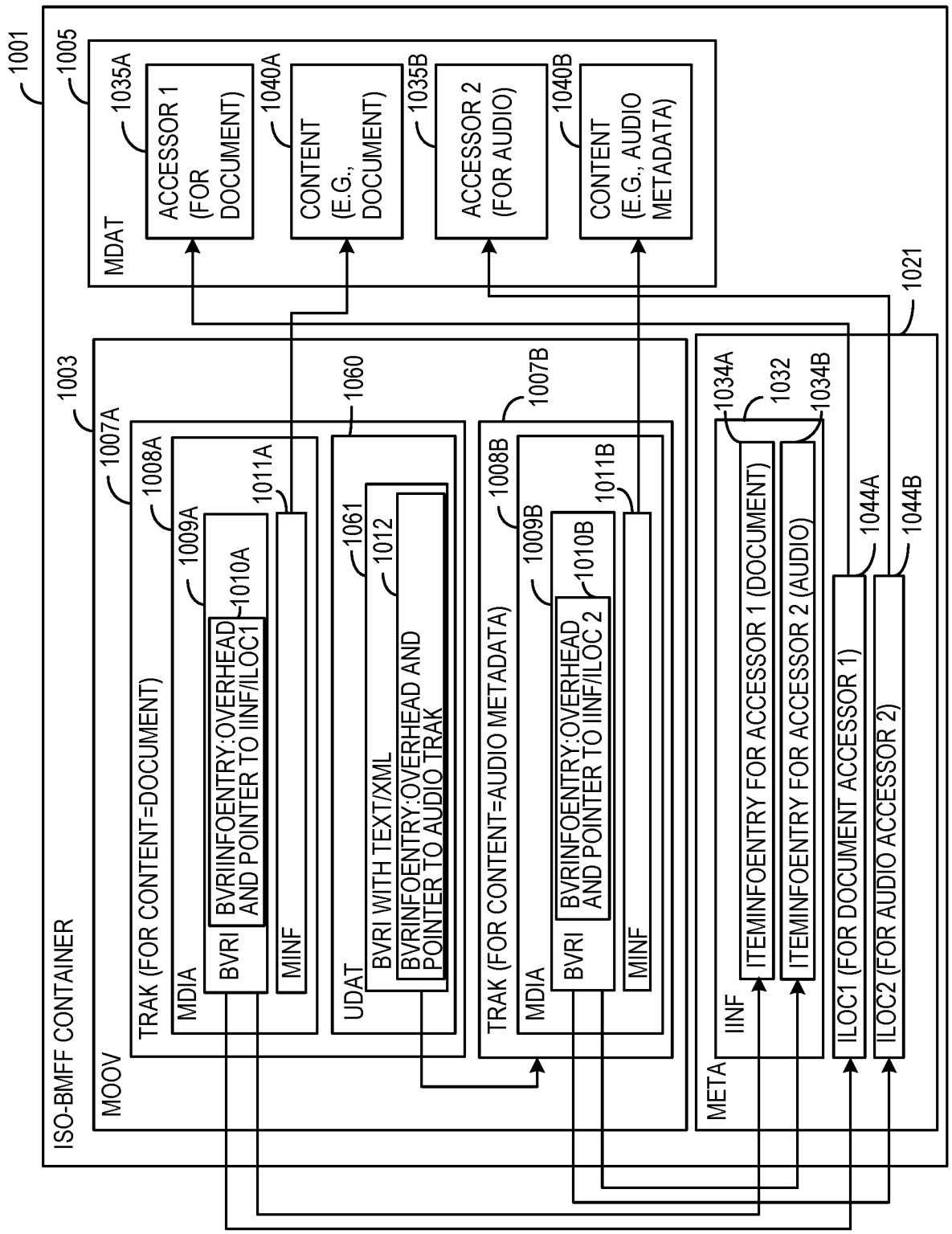

1001
ISO-BMFF CONTAINER
1005
MDAT
1035A
ACCESSOR 1 (FOR DOCUMENT)
1040A
CONTENT (E.G., DOCUMENT)
1035B
ACCESSOR 2 (FOR AUDIO)
1040B
CONTENT (E.G., AUDIO METADATA)
1021
1003
MOOV
1007A
TRAK (FOR CONTENT=DOCUMENT)
1008A
MDIA
1009A
BVRI
1010A
BVRIINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC1
1011A
MINF
1060
UDAT
1061
BVRI WITH TEXT/XML
1012
BVRINFOENTRY:OVERHEAD AND POINTER TO AUDIO TRAK
1007B
TRAK (FOR CONTENT=AUDIO METADATA)
1008B
MDIA
1009B
BVRI
1010B
BVRINFOENTRY:OVERHEAD AND POINTER TO IINF/ILOC 2
1011B
MINF
META
1034A
1032
1034B
IINF
ITEMINFOENTRY FOR ACCESSOR 1 (DOCUMENT)
ITEMINFOENTRY FOR ACCESSOR 2 (AUDIO)
1044A
1044B
ILOC1 (FOR DOCUMENT ACCESSOR 1)
ILOC2 (FOR AUDIO ACCESSOR 2)

FIG. 10

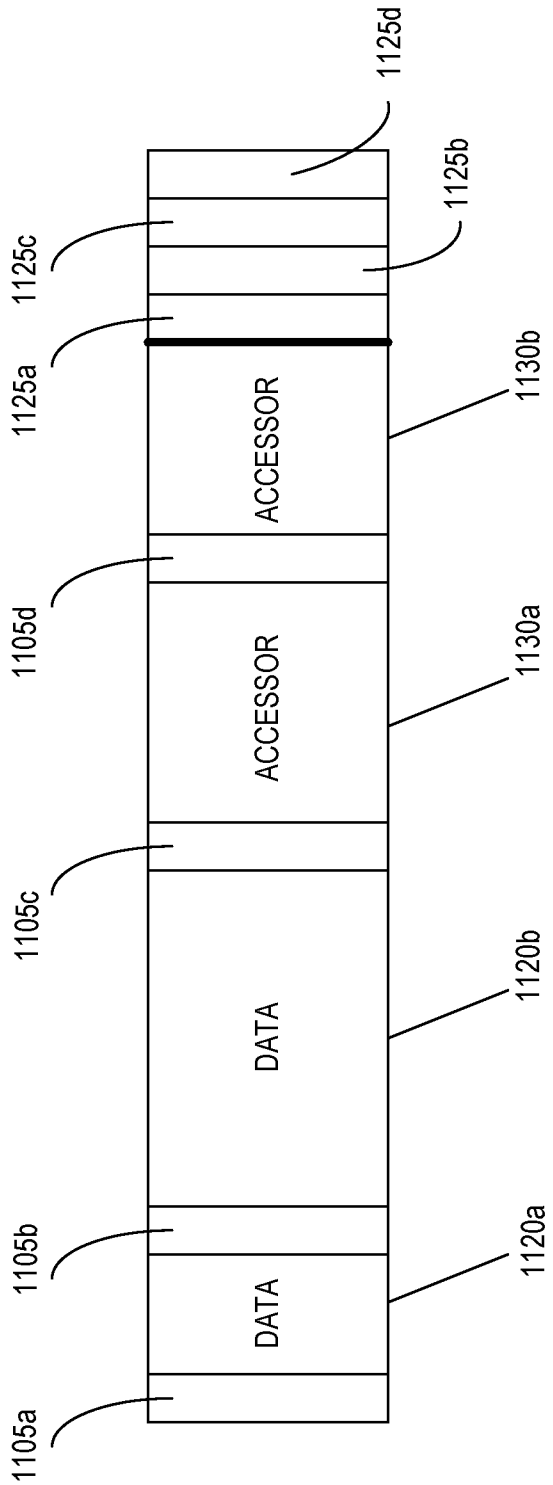

1125d
1125b
1125c
1125a
1130b
ACCESSOR
1105d
1130a
ACCESSOR
1105c
1120b
DATA
1105b
1120a
DATA
1105a

FIG. 11

# INTERNATIONAL SEARCH REPORT

International application No
PCT/US2019/013998

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F16/383
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | SHINYA KAWANAKA ET AL: "Accessibility commons : a metadata infrastructure for web accessibility", COMPUTERS AND ACCESSIBILITY, 1 January 2008 (2008-01-01), page 153, XP055568322, 2 Penn Plaza, Suite 701 New York NY 10121-0701 USA DOI: 10.1145/1414471.1414500 ISBN: 978-1-59593-976-0 Chapter 4; page 158 ----- | 1-23 |

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 13 March 2019 | 20/03/2019 |

| Name and mailing address of the ISA/ | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040,<br>Fax: (+31-70) 340-3016 | Krawaritis, Achilles |

1