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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 December 2003 (04.12.2003)

PCT

(10) International Publication Number
WO 03/100548 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/IB03/02069

(22) International Filing Date: 15 May 2003 (15.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0211897.4 23 May 2002 (23.05.2002) GB

(71) Applicant *(for all designated States except US)*: **KONINKLIJKE PHILIPS ELECTRONICS N.V.** [NL/NL]; Groenewoudseweg 1, NL-5621 BA Eindhoven (NL).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **EVES, David, A.** [GB/GB]; c/o Philips Intellectual Property & Standards, Cross Oak Lane, Redhill, Surrey RH1 5HA (GB). **COLE, Richard, S.** [GB/GB]; c/o Philips Intellectual Property & Standards, Cross Oak Lane, Redhill, Surrey RH1 5HA (GB).

(74) Agent: **TURNER, Richard, C.**; Philips Intellectual Property & Standards, Cross Oak Lane, Redhill, Surrey RH1 5HA (GB).

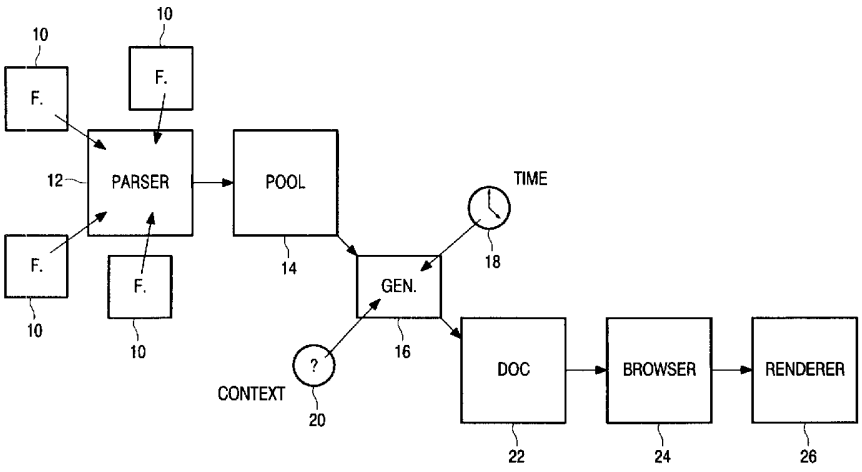
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DYNAMIC MARKUP LANGUAGE



(57) Abstract: A method for generating a markup language document comprises accessing a pool of active markup language fragments, processing the fragments using at least one predetermined factor, and generating a markup language document accordingly. In this way, a declarative markup language document is derived, but with content that is effectively dynamic.

WO 03/100548 A2

DESCRIPTION

DYNAMIC MARKUP LANGUAGE

This invention relates to a method for generating a markup language document.

In order to provide content to a browser, for example web pages to an Internet browser, HTML (HyperText Markup Language) was developed. HTML is a markup language that is solely declarative, containing text and links to other documents. No interpretation by the browser is required. However the content that can be presented in this way is obviously limited. A number of solutions to this problem have been developed, principally JAVA, which is a programming language that allows the creation of small programs (applets) that can be addressed by links in HTML documents. In this way graphics and the like can be easily provided to a browser, and rendered to the user. A known limitation of HTML and JAVA is that the content so provided is effectively fixed. Even if a JAVA applet is authored to provide a sequence of images to imitate movement, whenever that JAVA applet is addressed, the same sequence of images will be displayed. A number of solutions to this problem have been proposed.

International patent application publication WO 02/17082 describes dynamic content assembly on edge-of-network servers in a content delivery network. The disclosure enables a content provider to dynamically assemble content at the edge of the Internet, preferably on content delivery network (CDN) edge servers. Preferably, the content provider leverages an "edge side include" (ESI) markup language that is used to define Web page fragments for dynamic assembly at the edge. Dynamic assembly improves site performance by caching the objects that comprise dynamically generated pages at the edge of the Internet, close to the end user. The content provider designs and develops the business logic to form and assemble the pages, for example, by using the ESI language within its development environment. Instead of being

assembled by an application/web server in a centralised data centre, the application/web server sends a page template and content fragments to a CDN edge server where the page is assembled. Each content fragment can have its own cacheability profile to manage the "freshness" of the content. Once a user requests a page (template), the edge server examines its cache for the included fragments and assembles the page on the fly.

This prior art system however, is still based upon static content. The fragments that are used to fill the template to create a final document for rendering by a browser are predefined. There is no possibility for dynamic content.

United States patent application publication US 2001/0037359 describes a system and method for a server-side browser including markup language graphical user interface, dynamic markup language rewriter engine and profile engine. The proposal is a system and method for customizing access and delivery of information distributed in a packet-based network. The system includes a user computer and a destination server computer separated by a server computer hosting a server-side browser (SSB). The SSB includes a markup language graphical user interface (MLGUI), a dynamic markup language rewriter engine (DMLRE) and a profiling engine (PE). The SSB may be configured as an intermediary infrastructure residing on the Internet providing customized information gathering for a user. The components of the SSB allow for controlling, brokering and distributing information more perfectly by controlling both browser functionality (on the client-side) and server functionality (on the destination site side) within a single point and without the necessity of incremental consents or integration of either side.

The system of this prior art publication is a filtering system based upon profiles. Content is taken from a number of sources, filtered, rewritten as a single source and provided to a user. This complicated system has the disadvantage, as before, that the content that is provided to the user is still, nevertheless, static content.

The article "A proposal for Dynamic XML without DOM & Scripts" by A. Sundararajan, available at http://sundararajan.tripod.com/dxml/dxml.htm, states that in

dynamic web pages, the content and/or appearance of the web page varies with time. According to the article, currently, it is possible to create dynamic web pages (DHTML) using HTML DOM & Scripting. It states that there are known disadvantages with this approach, such as:-

- Knowledege of scripting & DOM is essential for writing dynamic web pages.
- HTML editors/tools generate script/DOM for dynamic web pages. However script generated by one tool cannot be processed by other tools. Specialized HTML comments are embedded by tools to even re-edit the HTML/JavaScript generated by the same tool.

The article describes the possibility of tag based, declarative, dynamic XML documents. A dynamic XML document is a document whose content and/or appearance varies with time. At an XML document level, dynamism involves changing an attribute or an element or text content of an element, at some time T.

While the ideas discussed in this article show that dynamic content for markup language documents is known, the solution offered by the ideas in this article have a number of drawbacks. Firstly, in order to maintain the dynamism of the content the client browser is required to have the additional functionality to process the tags that control the content. This prevents penetration of the technology amongst consumers and adds in a level of complexity that makes the authoring and handling of such documents more complicated. Secondly, once such a document is created, there is still only a small degree of dynamism achieved. The document and tags are fixed, the content will always act in a predictable fashion, and the process of creating the document is relatively inflexible.

It is therefore an object of the invention to provide a method of generating a markup language document that allows presentation of dynamic content, but does not have the drawbacks of the prior art.

According to the present invention, there is provided a method for generating a markup language document comprising accessing a pool of

active markup language fragments, processing said fragments using at least one predetermined factor, and generating a markup language document accordingly.

Owing to the invention, it is possible to provide a markup language document that is dynamic in content, but does not require any adaptation on the part of the client browser to render that document. The choice of fragments for the pool and factors for the processing allow flexibility in the authoring process.

Advantageously, a predetermined factor is time. Another possibility for a predetermined factor is context. Preferably, the pool is generated from a plurality of sources. Advantageously, the pool is created by parsing of markup language sources to obtain active markup language fragments.

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings, in which:-

Figure 1 is a schematic diagram of a system for generating a markup language document, and

Figure 2 is a schematic diagram of a method for generating a markup language document.

In the diagram of Figure 1, active markup fragments 10 are provided to a parser 12. These fragments can originate from a plurality of sources, and are active in the sense that they refer to content that is dynamic, for example, requiring reference to variables to determine their actual output. An example of such a fragment would be:

```
<fragment> example
    <object> object1
        <when> flag  then <location> 100, 100 </location> </when>
        <when> NOT flag then <location> 0,0 </location> </when>
    </object>
</fragment>
```

This fragment determines the location of object1 depending upon whether a variable flag is true or false. This fragment is obviously very simple; the complexity of fragments is only limited by the ingenuity of their author(s). A number of such fragments are provided to the parser 12, which ascertains that they are of a suitable format for passing to a pool 14 of fragments. The parser 12 would generally be an XML parser that checks that each fragment 10 is XML compliant, discarding those that are not. This ensures robustness of the method.

A snapshot generator 16 is for generating a markup language document. The generator 16 achieves this by accessing the pool 14 of active markup language fragments, processing the fragments using at least one predetermined factor, and generating a markup language document accordingly. In the Figure, a first predetermined factor is time, shown schematically at 18, and a second predetermined factor is context, shown at 20. These factors are set either by an author, or by a suitable computer program working under predefined conditions.

Taking the example of an active markup fragment given above, if there is included in the context information the logical conditional, flag = TRUE, then the output from the snapshot generator for that fragment would be:

```
<fragment> example
        <object> object1
                <location> 100, 100 </location>
        </object>
</fragment>
```

All of the fragments 10 in the pool 14 are processed in this way. Any fragment 10 that refers to a variable or conditional for which the generator 16 does not have the requisite information is discarded. A markup language document 22 is therefore generated from the fragments 10 that are in the pool 14. This document 22 is totally declarative, and can be rendered by a

conventional browser, without the requirement for any modification or adaptation of the client side browser. The snapshot generator 16 would normally be located on the server side of any network system, creating a document 22 for supplying to a browser 24, which passes the document 22 to a renderer 26 for rendering said document 22.

The method is summarised in Figure 2, which shows the method for generating the markup language document 22 comprising accessing 30 the pool 14 of active markup language fragments 10, processing 32 the fragments 10 using at least one predetermined factor, and generating 34 the markup language document 22 accordingly. The method can further comprise supplying 36 the document 22 to the browser 24 and rendering 38 the document 22.

A significant advantage of this method is the flexibility in generating documents that is facilitated by the method. The snapshot generator 16 can access the same pool 14 of fragments, but using different values and conditionals for the factors time and/or context, to obtain new markup language documents. So, for example, the time value can be changed, and a new document is easily generated based upon this changed factor. Equally fragments 10 can be added or subtracted from the pool 14, as desired. The generator 16 can then access the pool 14 and process the fragments 10 as described above to produce a new updated document 22. The snapshot process is repeated as necessary to generate new snapshots as time passes and context changes.

A further advantage of the method is that, when creating documents using changing factors, it is not necessary to maintain a timeline. It is only relevant to know when the next single possible change takes place, whether this is a start or end time, or a context change. Adding new fragments or removing old ones has a similar effect. This results in an efficient procedure for the generation of multiple documents.

CLAIMS

1. A method for generating a markup language document (22) comprising accessing (30) a pool (14) of active markup language fragments (10), processing (32) said fragments (10) using at least one predetermined factor (18, 20), and generating (34) a markup language document (22) accordingly.

2. A method according to claim 1, wherein a predetermined factor (18) is time (18).

3. A method according to claim 1 or 2, wherein a predetermined factor (20) is context (20).

4. A method according to claim 1, 2 or 3, wherein said pool (14) is generated from a plurality of sources.

5. A method according to any preceding claim, wherein said pool (14) is created by parsing of markup language sources to obtain active markup language fragments (10).

6. A method according to any preceding claim, and further comprising supplying (36) said document (22) to a browser (24).

7. A method according to claim 6, and further comprising rendering (38) said document (22).

8. A system for generating a markup language document (22) comprising a generator (16) for accessing a pool (14) of active markup language fragments (10), for processing said fragments (10) using at least one predetermined factor (18, 20), and for generating a markup language document (22) accordingly.

9. A system according to claim 8, and further comprising a browser (24).

10. A system according to claim 9, and further comprising a renderer (26).

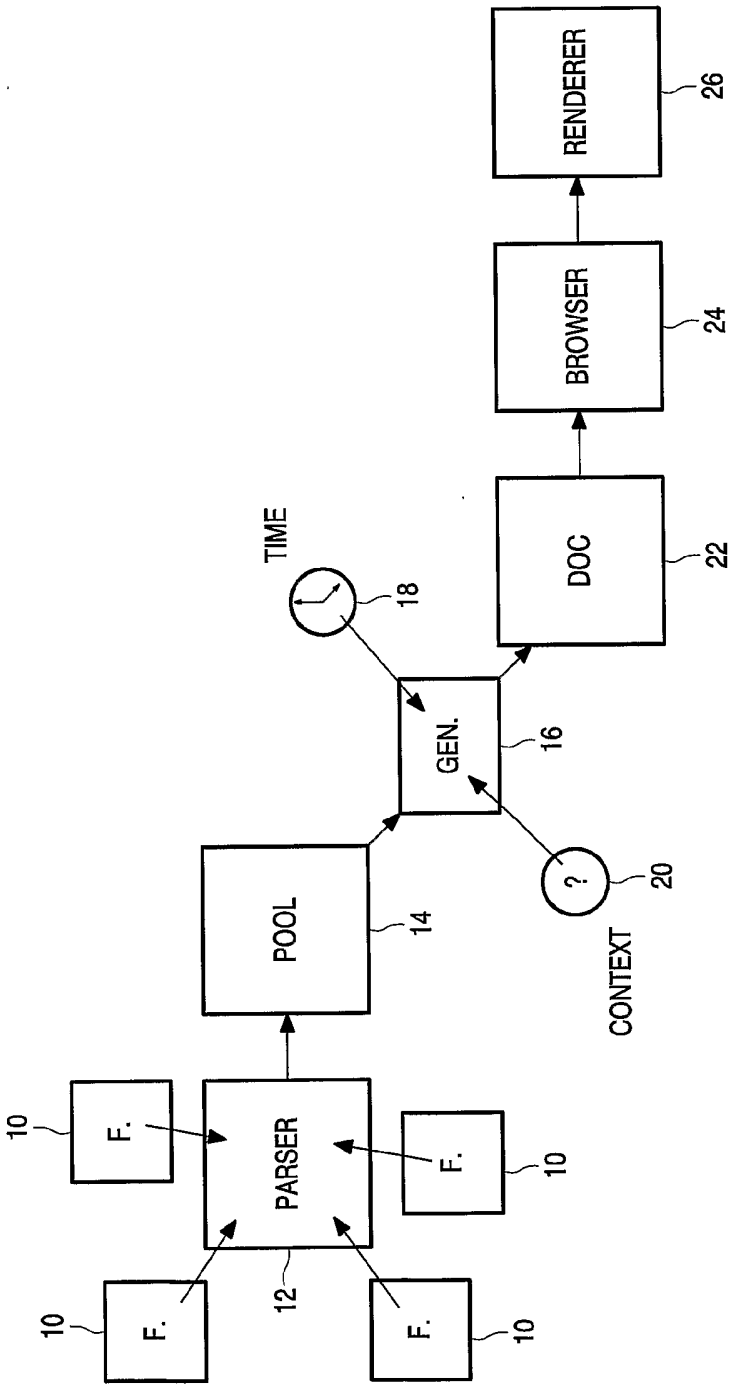

RENDERER
26
BROWSER
24
DOC
22
TIME
18
GEN.
16
POOL
14
?
20
CONTEXT
F.
10
PARSER
12

FIG.1

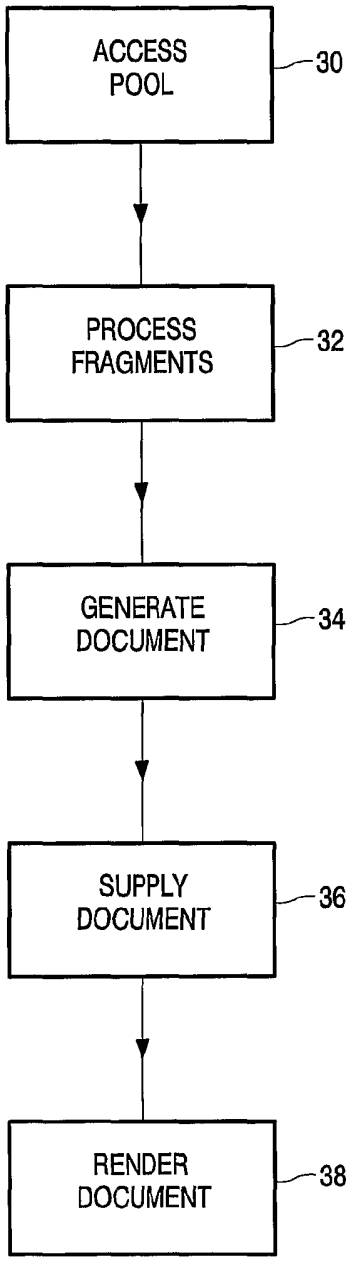

ACCESS POOL
30
PROCESS FRAGMENTS
32
GENERATE DOCUMENT
34
SUPPLY DOCUMENT
36
RENDER DOCUMENT
38

FIG.2