miller_usenix_2000.pdf

Claim 1 of US7472398B2 vs. disclosure in miller_usenix_2000.pdf

#	Claim Limitation	Supporting Text	Explanation	Support
1.0	**A computer system comprising:**	Page 9: 'The LAPIS browser-shell has been tested on **Linux, Solaris, and Windows NT**.' Page 9: 'LAPIS is written in **Java 1.1**… The LAPIS browser-shell has been tested on Linux, Solaris, and Windows NT.'	The reference explicitly states that LAPIS was implemented in Java 1.1 and successfully tested on the general-purpose operating systems Linux, Solaris, and Windows NT. A POSITA would understand that Java code runs on a JVM hosted by a physical computer system that necessarily includes a CPU, memory, and storage. Therefore the disclosed LAPIS embodiment is inherently executed on such a computer system.	Implicit
1.1	**a central processing unit (CPU);**	Page 9: 'LAPIS is written in Java 1.1… **The LAPIS browser-shell has been tested on Linux, Solaris, and Windows NT.**' Page 4: 'The web browser we used to prototype the browser-shell is called LAPIS (Figure 1), part of a system of generic tools for structured text…'	The reference discloses that LAPIS is a Java 1.1 application that was prototyped and tested on standard desktop operating systems (Linux, Solaris, Windows NT). A POSITA understands that Java applications run on a JVM that in turn runs on a general-purpose computer containing a CPU. While the reference does not explicitly mention 'CPU' or processor hardware, the disclosed embodiment is inherently executed on such hardware, making the CPU an implicit component of the system.	Implicit
1.2	**a memory unit coupled to the CPU;**	Page 9: 'LAPIS is written in Java 1.1… The LAPIS browser-shell has been tested on Linux, Solaris, and Windows NT.'	The reference never describes physical memory hardware or any CPU-memory coupling. While a POSITA would infer that a Java application running on the listed operating systems ultimately executes on a general-purpose computer that contains memory, the document itself supplies no disclosure of a 'memory unit coupled to the CPU.' Consequently, the element is not supported by the reference.	None
1.3	**an application stored in the memory unit and**	Page 9: '**The browser-shell features added about 2,000 lines of code.**'	The reference discloses that LAPIS is a software prototype whose 'browser-shell	Implicit

#	Claim Limitation	Supporting Text	Explanation	Support
	executable by the CPU;	Page 9: '**The LAPIS prototype is not robust enough for everyday use, largely because JEditorPane renders many web pages poorly.**' Page 4: '**The web browser we used to prototype the browser-shell is called LAPIS... part of a system of generic tools for structured text...**'	features' were implemented with 'about 2,000 lines of code.' A POSITA understands that executable code of this size must be loaded into memory before it can run. The statement that the prototype 'is not robust enough for everyday use, largely because JEditorPane renders many web pages poorly' confirms that the code is actually executed at runtime. Thus, the disclosed embodiment inherently includes an application (the LAPIS browser-shell code) that is stored in memory and executed by the CPU, even though the reference does not recite the magic words 'stored in memory' or 'executable by the CPU.'	
1.4	**a facade server** stored in the memory unit **and executable by the CPU;**	Page 7, col. 1, para. 2: 'LAPIS always sets the CGI environment variables when it invokes an external program. A program can use the presence or absence of these variables to determine whether it was invoked from the browser-shell...' Page 3, col. 2, para. 1: 'LAPIS can submit forms to local programs by the Common Gateway Interface (CGI), an existing standard used by web servers. This opens the possibility of running HTML applications entirely locally.' Page 7, col. 1, para. 3: 'Whether a form is being submitted or not, LAPIS always sets the CGI environment variables...'	The reference never discloses a distinct, persistent 'facade server' module. What it describes is a code path inside the LAPIS browser that (i) intercepts form submissions, (ii) populates CGI environment variables, and (iii) immediately exec()s the external program. There is no resident server process that listens for or routes requests, and nothing states that this mechanism avoids opening network ports. Consequently, the claimed 'facade server stored in memory and executable by the CPU' is not supported by the cited text.	None
1.5	**a program** stored in the memory unit **and executable by the CPU, wherein the program creates an interface between the facade server and a web-browser for exchanging**	[Page 7] LAPIS always sets the CGI environment variables when it invokes an **external program. A program** can use the presence or absence of these variables to determine whether it was invoked from the **browser-shell**... [Page 7] To submit a filled-out form back to the **external program**, LAPIS passes form parameters **using the Common Gateway Interface (CGI)**...	The reference does not disclose a distinct 'program' whose sole purpose is to create an interface between a 'facade server' and a web-browser. What it describes is an internal code path inside the LAPIS browser that (i) intercepts form submissions, (ii) populates CGI environment variables, and (iii) immediately exec()s the external program. There is no separate, persistent program module that mediates data exchange	None

#	Claim Limitation	Supporting Text	Explanation	Support
	data associated with the application,	[Page 3] LAPIS can **submit forms** to local programs by the **Common Gateway Interface (CGI)**, an existing standard used by web servers. This opens the possibility of running HTML applications entirely locally.	between a facade server and the browser; the browser itself performs the mediation on-the-fly. Therefore the claimed 'program stored in memory and executable by the CPU' that 'creates an interface between the facade server and a web-browser' is not supported.	
1.6	**wherein the facade server hosts the application without utilizing network protocols and without opening network ports.**	Page 3, column 2, paragraph 1: 'LAPIS can submit forms to local programs by the Common Gateway Interface (CGI), an existing standard used by web servers. This opens the possibility of **running HTML applications entirely locally**.' Page 7, column 1, paragraph 2: 'Whether a form is being submitted or not, **LAPIS** always sets the CGI environment variables when it **invokes an external program**. A program can use the presence or absence of these variables to determine whether it was invoked from the browser-shell, in which case it can present an HTML interface and act like a CGI program, or from the ordinary typescript shell, in which case it should present a text-only or command-line interface.' Page 7, column 1, paragraph 3: 'To submit a filled-out form back to the external program, LAPIS passes form parameters using the Common Gateway Interface (CGI).'	Although the reference does not disclose a persistent 'facade server,' it does describe LAPIS locally populating CGI environment variables and immediately exec()ing external programs without any mention of socket creation or TCP/IP traffic. The statement that this 'opens the possibility of running HTML applications entirely locally' (p. 3) implicitly confirms that no network protocols are utilized and no network ports are opened. A POSITA would therefore understand that the disclosed mechanism operates without network activity, satisfying the negative limitation.	Implicit