United States District Court,
N.D. California.

VMWARE, INC,
Plaintiff.
v.
CONNECTIX CORPORATION and Microsoft Corporation,
Defendants.

Nos. C 02-3705 CW, C 03-0654 CW

March 25, 2005.

James Chung-Yul Yoon, Michael A. Ladra, Ron Eleazer Shulman, Christopher R. Parry, Julie M. Holloway, Monica Mucchetti Eno, Wilson, Sonsini, Goodrich & Rosati, Palo Alto, CA, Leanne J. Fitzgerald, EMC Corporation, Mountain View, CA, Allen C. Nunnally, Dominic E. Massa, Michael R. Dube, William F. Lee, Wilmer, Cutler, Pickering, Hale and Dorr, LLP, Boston, MA, John M. Gunther, EMC Corporation, Hopkinton, MA, Michael A. Oblon, Wilmer, Cutler, Pickering, Hale and Dorr, LLP, Washington, DC, Stephen M. Muller, Hale & Dorr, LLP, Boston, MA, for Plaintiff.

James J. Elacqua, Dechert, LLP, Mountain View, CA, Brian K. Erickson, Dewey, Darryl Jedd Adams, Ballantine, LLP, Kevin S. Kudlac, Attorney at Law, Austin, TX, Jeannine Yoo Sano, Howrey, LLP, East Palo Alto, CA, Cecil E. Key, Cono A. Carrano, Dewey, Ballantine, LLP, Washington, DC, for Defendants.

CLAIM CONSTRUCTION ORDER

CLAUDIA WILKEN**, District Judge.**

Plaintiff VMWare, Inc. and Defendants Connectix Corporation and Microsoft Corporation dispute the meaning of several terms and phrases in the claims of U.S. Patent Nos. 6,397,242 ('242 patent), 6,496,847 ('847 patent), 5,768,593 ('593 patent), 5,761,477 ('477 patent) and 6,108,715 ('715 patent). Each party requests that the Court adopt its proposed construction of the disputed terms and phrases. The claim construction hearing was held on February 4, 2005. Having considered the parties' pre-and post-hearing papers, the evidence cited therein and oral argument, the Court construes the disputed terms and phrases as set forth herein.

BACKGROUND

I. '847 Patent

Plaintiff is the assignee of the '847 patent, entitled "System and Method for Virtualizing Computer Systems." According to Plaintiff, the '847 patent discloses new technology that allows a virtual machine monitor (VMM) to operate directly on a central processing unit (CPU) along with the host operating system (HOS), thereby allowing more than one operating system to function independently and alternatively on the

same CPU.

Conventional computer systems have a single operating system that runs various compatible applications. Generally speaking, virtual machine technology allows for more than one operating system to run on the same CPU. The '847 identifies two prior art virtual machine systems: (1) a system in which a VMM operates as a conduit between the CPU and various guest operating systems, where there is no host operating system running directly on the CPU, and (2) a system in which a host operating system runs directly on the CPU, and a VMM, with a guest operating system, communicates with the CPU through the host operating system. In the first prior art system, no operating system communicates directly with the CPU because the VMM intercepts all requests and commands. In the second prior art system, the guest operating system cannot communicate directly with the CPU because the host operating system intervenes.

The '847 patent purports to solve these problems. In the system claimed by the '847 patent, the VMM and host operating system each run directly on top of the CPU; a "context switch" exchanges primary access to the CPU between both control-level systems. Thus, both the host operating system and the VMM can, alternatively and independently, communicate directly with the CPU.

II. '242 Patent

Plaintiff is also the assignee of the '242 patent. The '242 patent discloses how the VMM claimed by the '847 patent executes instructions on the CPU. The '242 patent identifies two methods by which operating systems execute instructions on CPUs: direct execution and binary translation. Direct execution is fast and efficient, although potentially "unsafe" in that it sometimes yields unexpected or unwanted results. Conversely, binary translation is a relatively slower process that is safer than direct execution. The '242 patent discloses an execution subsystem that determines whether instruction sequences executed by the VMM are directly executable or non-directly executable, thereby requiring binary translation.

III. '593 Patent

Defendant Connectix is the assignee of the '593 patent. According to Defendants, the '593 patent discloses an invention that dramatically speeds up the operation of certain computer programs. Specifically, the patent discloses a technique called "cross-compilation" that allows for faster execution of computer instructions, which were written for one type of computer processor, on a different type of computer processor. Defendants maintain that the method disclosed in the '593 patent is applicable to virtual machine software that requires execution of instructions in an environment other than that for which the instructions were originally intended to run.

IV. '477 Patent

Defendant Microsoft is the assignee of the '477 patent, entitled "Methods for Safe and Efficient Implementations of Virtual Machines." According to Defendants, the '477 patent discloses a computer-implemented method of using checking instructions while executing virtual machine instructions in order to signal when an instruction may attempt to perform an illegal memory operation. The technology purports to protect the memory of a computer that runs virtual machine software.

V. '715 Patent

Defendant Microsoft is also the assignee of the '715 patent, entitled "Method and System for Invoking

Remote Procedure Calls." The method described in the patent involves the passing and mapping of data; when a client process initiates the transfer of data from a server process, a third process maps the data into its address space and then copies it to the client process's address space.

VI. Procedural History

On August 1, 2002, Plaintiff filed a complaint against Connectix alleging that it was infringing the '242 patent by, *inter alia,* making and selling a product known as "Virtual PC for Windows." C 02-3705 CW. Connectix answered and counterclaimed, alleging infringement of its '593 patent and invalidity of the '242 patent, and seeking a declaration of patent non-infringement.

On February 14, 2003, Plaintiff filed another lawsuit against Connectix, this time alleging infringement of the '847 patent. C 03-0654 CW. On February 20, 2003, Plaintiff amended this complaint, adding Microsoft as a defendant. The two cases were ordered related on March 13, 2003. On June 20, 2003, Defendants filed a second amended answer and counterclaims, alleging that Plaintiff was infringing the '477 and '715 patents.

After the February 4, 2005 claim construction hearing, the parties each submitted supplemental briefing in order to address some of the questions raised by the Court at the hearing.

LEGAL STANDARD

The interpretation of patent claims is a question of law to be decided by the Court. Markman v. Westview Instruments, Inc., 517 U.S. 370, 371-73, 116 S.Ct. 1384, 134 L.Ed.2d 577 (1996). "In construing the claims, the analytical focus must begin and remain centered on the language of the claims themselves, for it is that language that the patentee chose to use to 'particularly point [ ] out and distinctly claim[ ] the subject matter which the patentee regards as his invention.' " Interactive Gift Express, Inc. v. Compuserve, Inc., 256 F.3d 1323, 1331 (Fed.Cir.2001)(quoting 35 U.S.C. s. 112, para. 2).

Words in the claim are generally given their ordinary meaning. Texas Digital Sys. Inc. v. Telegenix Inc., 308 F.3d 1193, 1201-02 (Fed.Cir.2002) ("The terms used in the claims bear a 'heavy presumption' that they mean what they say and have the ordinary meaning that would be attributed to those words by persons skilled in the relevant art."). "The ordinary meaning of a claim term may be determined by reviewing a variety of sources, including the claims themselves, other intrinsic evidence including the written description and the prosecution history, and dictionaries and treatises." Teleflex, Inc. v. Ficosa N. Am. Corp., 299 F.3d 1313, 1325 (Fed.Cir.2002) (internal citations omitted).

While words in the claim are generally given their ordinary meaning, the specification or prosecution history may indicate otherwise. Vitronics Corp. v. Conceptronic, Inc., 90 F.3d 1576, 1582 (Fed.Cir.1996). "[A] patentee may choose to be his own lexicographer and use terms in a manner other than their ordinary meaning, as long as the special definition is clearly stated in the patent specification or file history." *Id.* However, claims are not limited to the preferred embodiment described in the specification. SRI Int'l v. Matsushita Elec. Corp. of Am., 775 F.2d 1107, 1121 (Fed.Cir.1985) (*en banc,* plurality opinion).

DISCUSSION

I. '847 Patent

A. System Level

The parties dispute the meaning of the term "system level" as it is used in claims 1 and 10 of the '847 patent. Plaintiff originally argued that "system level" meant "level of operation on the hardware processor at which a program has the capability to execute any of the processor's instructions," while Defendants' original proposed construction was "level at which software is able to independently issue both privileged and non-privileged instructions and to independently modify the Host processor state." After the February 4 *Markman* hearing, the parties revised their proposed constructions. Plaintiff clarifies that its construction of the term means "level of operation on the hardware processor at which a program has the capability to execute all of the processor's instructions," and Defendants' new construction is "level at which software is able to independently issue at least some but not necessarily all privileged instructions and at least some but not necessarily all non-privileged instructions and to independently modify the host processor state."

The principal dispute, the parties agree, is whether a program running at "system level" is required to have access to all privileged instructions of the CPU. Plaintiff argues in the affirmative, relying primarily on extrinsic evidence, including a document published by Microsoft, that the plain meaning of "system level" requires that all privileged instructions are accessible. Defendants argue that their construction comes directly from the definition of "system level" that Plaintiff submitted in a response to an office action by the PTO.

Defendants' argument is not persuasive; in the same PTO response that they cite in support of their construction, Plaintiff stated, in an effort to define "system level," that the host processor is "the processor on which all system-level components run." Adams Decl., Ex. 9. Plaintiff's expert Professor Bershad testified that operating at "system level" means having the ability to execute all privileged instructions, and Defendants' expert Barbara Frederiksen testified that a program was not operating at system level if it did not have the ability to issue all privileged instructions. Moreover, the '847 patent specification states that "system level" means to have "essentially complete control of the system." '847 patent at 14:56-65.

Thus, the ordinary meaning of running at "system level," according to Plaintiff's expert, at least one document published by Microsoft, and Defendants' expert, is having the ability to issue all privileged instructions. The Court construes the term "system level" to mean "level of operation on the hardware processor at which a program has the capability to execute all of the processor's instructions."

B. Virtual Machine Monitor (VMM)

The parties dispute the meaning of the term "virtual machine monitor," or "VMM," as it is used in claims 1, 2, 10, 11, 12, 13, 20 and 21 of the '847 patent. Plaintiff originally construed the term to mean "software for running a sequence of virtual machine (VM) instructions." After the *Markman* hearing, Plaintiff added the following phrase to its construction: "A VMM works to ensure that the VM instructions are properly processed." Defendants proposed construction, which remained unchanged after the hearing, is "a thin piece of software that runs directly on top of the hardware and virtualizes all the resources of the machine."

Plaintiff derives its construction from the '242 patent. However, Defendants' construction defines the term "virtual machine monitor" exactly as the '847 patent does: "A virtual machine monitor is a thin piece of software that runs directly on top of the hardware and virtualizes all the resources of the machine ." '847 patent 1 :47-47 (contained in the section entitled "Virtual Machine Monitors").

Plaintiff offers several unpersuasive arguments why the '847 patent's own definition of the disputed term should not apply. First, Plaintiff argues that the patent definition describes only prior art. However, the '847 patent, after defining a "virtual machine monitor," does not thereafter distinguish or redefine the term as it is used in the patent's claims. Second, Plaintiff argues that Ms. Frederiksen acknowledged that virtual machine monitors need not virtualize all resources of a CPU. However, in the deposition testimony that Plaintiff cites, Ms. Frederiksen stated that, while the customary meaning of "virtual machine monitor" does not require the virtualization of all resources, the '847 patent's definition does. Lastly, Plaintiff argues that Defendants' construction uses the term "virtualizing," which is inherently ambiguous. However, Plaintiff also acknowledges that it first introduced the term "virtualizing" in describing "virtual machine monitor" during the preliminary injunction phase of this action.

The definition for the term "virtual machine monitor" found in the ' 847 patent specification is explicit and clear. "Virtual machine monitor" is construed to mean "a thin piece of software that runs directly on top of the hardware and virtualizes all the resources of the machine."

C. Independently Issue ... and ... Modify

The parties dispute the meaning of the following phrase, found in claims 1, 10, 20 and 21 of the '847 patent: "both the VMM and the HOS thus being able to independently issue both privileged and non-privileged instructions for execution by the processor and to independently modify the processor state." Plaintiff originally construed the disputed phrase as follows:

The HOS and the VMM, when active, must each have the ability to control the system without being constrained by the other, including (a) issue both privileged and non-privileged instructions for execution by the processor, and (b) modify the processor state.

Following the *Markman* hearing, Plaintiff substituted the word "constrained" in its initial construction with the word "stopped." Prior to the hearing, Defendants proposed the following construction:

Because the VMM and HOS are at the system level, both programs can independently issue both privileged and non-privileged instructions for execution by the processor and independently modify the processor state. "Independently issue both privileged and non-privileged instructions for execution by the processor" means "directly execute privileged and non-privileged instructions on the host processor." "Independently modify the processor state" means "change the processor state of the host processor by directly executing an instruction on the host processor."

Following the Markman hearing, Defendants proposed the following:

Because the VMM and HOS are at the system level, both programs can independently issue both privileged and non-privileged instructions for execution by the processor and independently modify the processor state. "Independently issue both privileged and non-privileged instructions for execution by the processor" means "directly execute privileged and non-privileged instructions on the host processor without the assistance of another program." "Independently modify the processor state" means "change the processor state of the host processor by directly executing an instruction on the host processor without the assistance of another program."

Here, Plaintiff notes, Defendants have improperly imported the limitation "directly execute" into their

construction.

Plaintiff has cited substantial evidence from the '847 patent specification to support its argument that "independently," in the context of these claims, means "without being stopped by the other." *See, e.g.,* '847 patent 7 :35-37 ("the invention also provides that the VMM can configure the processor ... independently of any constraints imposed by the host operating system."). Moreover, Plaintiff's revised construction is concise and logical, and the Court adopts it.

D. Host Operating System Context

The parties dispute the meaning of the term "host operating system context" as it is used in claims 2, 11, 12, 20 and 21 of the '847 patent. Before the *Markman* hearing, Plaintiff construed the term to mean "the immediate operating environment for the Host OS program." After the hearing, Plaintiff submitted this proposed construction:

The environment in which the Host OS program operates. This environment includes information that the Host OS uses to operate. It also includes the information necessary to be saved and restored while switching control from the Host OS to the VMM and back, but this information does not necessarily include all of the processor's registers.

Defendants' proposed construction has not changed: "The state associated with the Host operating system that is set or restored during a total context switch including all registers, segments, floating point registers, control registers, and interrupt and exception vectors of the Host processor."

In support of their construction, Defendants cite specification language which states, under the subsection entitled "Total Switch," that " 'context' refers to the state that is set and restored during the total switch, for example, the state of the address space, the general purpose registers, floating-pointer registers, privileged users, interrupt vectors and hardware exception vectors." '847 patent 11 :62-66. Defendants argue that the term "context" is thus defined in the patent as the state set or restored during a "total switch" only.

Plaintiff disagrees, arguing that Defendants are attempting to import improperly limitations from the specification's preferred embodiment; the subsection Defendants cite also states that the "total switch" is an element of the preferred embodiment. Plaintiff also notes that Defendants are defining "context" in accordance with the wrong term; the disputed claim term is "HOS context" and not the "total switch context." Plaintiff contends that the claims do not require "context" to include all registers, as Defendants maintain they do.

The definition of the term "context" relied upon by Defendants is disclosed in the preferred embodiment; thus, its limitations should not be imported to the claim term if the claim does not require it. Here, it clearly does not. Plaintiff's construction is logical and comports with the plain and ordinary meaning of the disputed term. "Host operating system context" is construed to mean "the environment in which the Host OS program operates. This environment includes information that the Host OS uses to operate. It also includes the information necessary to be saved and restored while switching control from the Host OS to the VMM and back, but this information does not necessarily include all of the processor's registers."

E. "Disjoint" Address Space

The parties dispute the meaning of the phrase "second virtual address space, which is disjoint from the first

virtual address space" as it is used in claim 12 of the '847 patent. Plaintiff's initial proposed construction was " 'disjoint' means that the first and second address spaces do not overlap with the exception of commonly-mapped pages used to coordinate between the VMM and HOS." After the *Markman* hearing, Plaintiff proposed the following construction: "The first and second address spaces are generally separate from each other and generally do not overlap. There can be an overlap which includes commonly-mapped pages that are used to switch back and forth between the VMM context and the HOS context." Defendants propose "the virtual address space of the HOS has no addresses in common with the virtual address space of the VMM, except during a total switch."

Here, the parties dispute exceptions to the general rule that the term "disjoint" denotes a complete mutual exclusivity of address space. Plaintiff argues that the only exceptions contemplated by the '847 patent exist in two preferred embodiments in which common address space is shared by "cross pages" that coordinate the context switch between the HOS and the VMM. Defendants argue that the '847 patent does not disclose any such permanent shared address space; rather, the shared address space is only a temporary mapping that occurs during the context switch.

Plaintiff concurs in its reply papers that the patent does not disclose a permanent space for sharing addresses, but notes that Defendants' construction is improperly limiting based upon the patent language that both parties cite in support of their respective constructions. The '847 patent states, "This cross page only needs to be mapped temporarily in the VMM context's address space, just before the total switch to the HOS, and just after the control returns to the VMM." '847 patent 14 :48-51. Thus, Defendants' limitation that address space is shared only "during a total switch" is inappropriate; the patent language they cite states clearly that the mapping also occurs both before and after the switch.

Plaintiff's revised proposed construction contains no such improper temporal limitation. The Court therefore adopts it.

II. '242 Patent

A. Execution Subsystem Forming Decisions Means

The parties dispute the meaning of the following phrase, found in claim 1 of the '242 patent:

execution subsystem forming decision means for discriminating between the directly executable and non-directly executable VM instructions, and for selectively directing the VMM to activate the direct execution subsystem for execution by the hardware processor of the directly executable VM instructions and to activate the binary translation subsystem for execution on the hardware processor of the non-directly executable VM instructions.

The parties agree that the disputed phrase is a means-plus-function limitation governed by Title 34 U.S.C. section 112(6). However, the parties dispute both the function and the corresponding structure of the execution subsystem.

1. Function

Plaintiff argues that the function performed by the execution subsystem is "determining transitions between directly executable and non-directly executable instruction sequences and activating the proper subsystem for execution on the hardware processor." Defendants propose the following function: "The decision means

determines which VM instructions are directly executable and which VM instructions are non-directly executable, and directs the execution of each directly executable instruction by the direct execution subsystem, and directs the execution of each non-directly executable instruction by the binary translation subsystem."

The principal distinction between the parties' constructions is whether the subsystem must analyze *each* instruction to determine if direct execution is safe or binary translation is required (as Defendants propose), or whether the subsystem may consider *sequences* of instructions in making its determination (as Plaintiff maintains). In Plaintiff's proffered construction, the CPU executes instruction sequences in one mode until a condition occurs causing a mode change.

In Defendants' opposition papers, they argue only that the execution subsystem must detect all *transitions* between direct execution and binary translation, not that it must examine each *instruction* to determine which mode should be used. In Plaintiff's construction, the subsystem's function does detect all transitions between direct execution and binary translation; the subsystem operates in one mode until, at an appropriate time, it switches to the other. Defendants thus do not rebut Plaintiff's argument that each instruction need not be analyzed in order to activate an instruction mode. Moreover, Plaintiff cites Defendants' expert, Dr. Vahdat, who testified that the '242 patent does not require the subsystem to examine each instruction sequence in order to make a determination as to whether direct execution or binary translation should be used.

Thus, the function of the execution subsystem is construed as "determining transitions between directly executable and non-directly executable instruction sequences and activating the proper subsystem for execution on the hardware processor."

2. Structure

Plaintiff initially proposed the following structure for the execution subsystem, referring to Figure 2 of the '242 patent: "an execution subsystem that determines which execution mode to use based upon at least aspects of: (1) segment state; and (2) privilege level of the virtual machine." After the Court expressed concern at the *Markman* hearing that this proposed structure did not sufficiently recite the algorithm necessary to perform the claimed function, Plaintiff proposed the following structure:

A Decision Subsystem that performs the following steps:

(1) evaluates the segment state and privilege level of the virtual machine and (2) determines if it is safe to perform direct execution. If it is safe to do direct execution, the Decision Subsystem (3) activates direct execution mode. If it is not safe, the Decision Subsystem (4) activates binary translation mode.

The Defendants describe the subsystem's structure as

x86 compatible hardware running a VMM with an execution subsystem that discriminates between instructions that are directly executable and non-directly executable as follows: (1) Segment Reversibility. The VMM selects between binary translation and direct execution in accordance with the state machine described at 2l:4-44. (2) Operation Mode of the VM. In addition to discriminating based on segment reversability, the VMM further selects between direct execution and binary translation based on the operating mode of the x86 processor, 19:43-20:64:

v80-86 mode-direct execution is selected

real mode-binary translation is selected

system management mode-binary translation is selected

protected mode-binary translation is selected if and only if the processor's CPL=0, the interrupt flag is cleared, or the I/O privilege level is greater than or equal to the CPL.

The corresponding structure for the "selectively directing" function is x86 compatible hardware running a VMM, which includes a direct execution subsystem to directly execute each directly executable instruction and a binary translation subsystem to execute each non-directly executable VM instruction.

In the context of computer software, the structure for carrying out a disclosed function is "a computer, or microprocessor, designed to carry out an algorithm." WMS Gaming, Inc. v. Int'l Game Tech., 184 F.3d 1339, 1348-49 (Fed.Cir.1999). However, the structure is limited to the disclosed algorithm. Id. at 1348. Here, according to the '242 patent specification, "Figure 2 ... shows the general structure of binary translation execution engine or sub-system according to the invention." '242 patent 21 :46-48. Defendants argue that the patent discloses only one microprocessor (the x86) for performing the recited function; thus, the structure of the subsystem should be limited to that particular microprocessor. However, as Defendants acknowledge, the '242 specification clearly states, "Because the Intel x86 architecture is so widespread, it is the best (but not only) example of the applicability of the invention." '242 patent 8 :22-24. Moreover, Plaintiff notes that there is no requirement that the patent identify a microprocessor of a certain type as part of the structure.

Because the patent clearly identifies Figure 2 as the structure of the execution subsystem and provides that the x86 is not the only microprocessor that can perform the recited function, the Court must determine whether the revised algorithm proposed by Plaintiff is sufficient. Defendants argue that Plaintiff's construction is devoid of any algorithm. That is not the case; in Plaintiff's construction, the execution subsystem determines whether the instruction is directly executable or non-directly executable by considering the segment state and the privilege level of the virtual machine. The '242 patent, in a subsection entitled "Execution Mode Decision," states, "the decision of whether to use binary translation or direct execution depends on the segment state. It also depends, however, on the privilege level of the system, which, in the Intel x86 context, is also tied to which operating mode the system is in." '242 patent 19 :38-42. Thus, the algorithm proposed by Plaintiff after the *Markman* hearing is sufficient to perform the function of the execution subsystem as that function is described in the patent.

B. Executing Directly ... Using Direct Execution and Non-Directly ... Using Binary Translation ...

The parties dispute the meaning of the phrase "in the VMM, executing the directly executable VM instructions using direct execution and executing the non-directly executable VM instructions using binary translation" as it is used in claim 17 of the '242 patent. Plaintiff originally proposed the following construction: "The VMM uses direct execution to execute VM instructions that are directly executable. The VMM uses binary translation to execute VM instructions that are non-directly executable." Following the hearing, Plaintiff proposed the following construction: "Directly executable instructions: VM instructions that are executed when the system is in direct execution mode. Non-directly executable VM instructions:

VM instructions that are executed when the system is in binary translation mode." Defendants construe the phrase to mean "using direct execution to execute each directly executable VM instruction and using binary translation to execute each non-directly executable instruction."

Defendants submit the same argument here that they did in support of their construction for the function of the term "execution subsystem forming decisions means," and the Court again rejects it. Plaintiff's construction gives the phrase its plain and ordinary meaning, and the Court adopts it.

C. Binary Translation

The parties no longer dispute the meaning of the term "binary translation" as it is used in claims 1, 17 and 18 of the '242 patent. In their opposition papers, Defendants agree to Plaintiff's proposed construction: "a technique for converting a sequence of virtual machine instruction into a corresponding sequence of output instructions for execution, and using a cache when possible."

III. '593 Patent

A. Native Code

The parties dispute the meaning of the term "native code" as it is used in claims 1 and 7 of the '593 patent. Defendants originally construed the term to mean "code that has been compiled to be executed on the current architecture on which it is being run." After the *Markman* hearing, Defendants offered an alternative construction-"code that can be executed directly by the CPU"-but nevertheless argue that their initial construction was adequate. Plaintiff's proposed construction is

code which has been compiled into the instruction set of the instruction set processor for which the code is native. Native code for an instruction set processor is also referred to as 'machine code' or 'machine language' for that processor. A native instruction for a processor is an instruction that has been compiled into the instruction set of that processor.

The parties' central dispute is over the difference between "native" and "non-native," and whether, in the '593 patent, the term "native" refers to the underlying computer *processor* or to the underlying computer *architecture*. Defendants contend that the instruction set of a processor does not determine what is "native" because code that is written for one operating system may not be native to another operating system, even if the underlying processor is the same. Defendants argue that "native" refers instead to the underlying computer architecture (which may include both the processor and system software), and that Plaintiff is attempting to read a limitation from the preferred embodiment into the claim. Defendants maintain that, while in the preferred embodiment different processors have different computer architectures, that is not always the case, and the '593 patent specification recognizes that. *See* '593 patent 1 :19-25 ("Examples of different architectures include ... Sun Microsystems computers running the Solaris operating system, and computer systems using the Unix operating system.").

Plaintiff argues that the plain meaning of "native code" is code that can be executed on a given processor, and that the plain meaning of the term "architecture" is the architecture of the processor and does not include an operating system. Plaintiff also analogizes a processor to a human being; human beings speak in a "native" tongue, and language must be translated from a "non-native" tongue into a "native" tongue in order for the person to understand it.

However, as Defendants note, Plaintiff's construction does not comport with its explanation of the methods disclosed in the '242 patent. The methods disclosed in the '242 patent are required, contends Plaintiff, because code that is created for a given processor may nevertheless require "binary translation" before it can be executed on that processor if a different operating system is being run (i.e. the architecture is different).

Defendants cite language from the '593 patent which clearly states that computer architecture is not synonymous with processor architecture. Plaintiff should not be permitted to read limitations from the preferred embodiment into the claim terms. The Court adopts Defendants' initial construction of the term "native code."

B. Non-Native Application, Non-Native Instruction

The proper constructions of the terms "non-native application" and "non-native instruction" are dependent upon the construction of the term "native code." Defendants initially construed the terms to mean "an application that is not compiled to be executed on the current architecture on which it is being run," and "a set of instructions within a non-native application," respectively. After the *Markman* hearing, Defendants submitted an alternative construction for "non-native application": "applications that cannot be executed directly by the CPU." However, Defendants again asserted that their initial constructions of the terms were correct. Plaintiff's proposed constructions are "a non-native application is an application containing non-native instructions" and "a non-native instruction is an instruction that is not contained within the instruction set of the processor on which the code is to be executed."

Because the Court has construed the term "native" in terms of the computer architecture and not the computer processor, the Court adopts Defendants' initial proposed constructions of these disputed claim terms.

IV. '477 Patent

A. Virtual Machine; Virtual Machine Instructions

The parties dispute the meaning of the terms "virtual machine" and "virtual machine instructions" as they are used in claims 1, 2, 4, 7, 8, 9 and 10 of the '477 patent. In these claims, the term "virtual machine" is used only in the context of the term "virtual machine instructions"; thus, the Court will construe the terms together. Defendants construe the term "virtual machine" to mean "a representation of a computer architecture created by software," and the term "virtual machine instructions" to mean simply "instructions running on a virtual machine. Plaintiff construes "virtual machine" to mean, "a software-based system that can be used on many different hardware architectures." Plaintiff construes "virtual machine instruction" as follows: "an instruction selected from the set of instructions defined by the virtual machine, which are different from machine executable instructions, also known as native instructions, for the machine on which the virtual machine is being run."

Defendants contend that the disputed terms are entitled to their plain and ordinary meaning, and note that Plaintiff's own expert defined a "virtual machine" in its general context as "a representation of some computer." Defendants also argue that, while it is desirable, a "virtual machine" need not have the ability to run on different hardware architectures.

Critically, Defendants also contend that Plaintiff's construction of "virtual machine" does not comply with the doctrine of claim differentiation. "It is settled law that when a patent claim does not contain a certain

limitation and another claim does, that limitation cannot be read into the former claim in determining either validity or infringement." SRI Int'l v. Matsushita Elec. Corp. of Am., 775 F.2d 1107, 1122 (Fed.Cir.1985). Here, claims 1 and 7 are independent and use the term "virtual machine instructions," while claims 4 and 10 are dependent claims and state "said virtual machine instructions are executable with substantially similar behavior on computers with different central processing units." Thus, Defendants argue, claims 1 and 7 should be understood to claim "virtual machine instructions" that are not necessarily executable on different processors. Plaintiff's construction, argue Defendants, renders claims 4 and 10 superfluous.

Plaintiff argues that Defendants' construction of "virtual machine" does not find support in the '477 patent specification. Plaintiff also argues that its construction of "virtual machine" is appropriate because Defendants disclaimed a "virtual machine" that was hardware-based. Finally, Plaintiff contends that the doctrine of claim differentiation does not apply because the claims are not identical in scope under its proposed construction.

Plaintiff is correct that, under its proposed construction of "virtual machine," the scope of the dependent and independent claims is not identical, but its construction would nonetheless render the dependent claims superfluous. That is so because Plaintiff's construction would erase the distinction between the "virtual machine" in independent claims 1 and 7, which does not necessarily run on different processors, and the "virtual machine" in claims 4 and 10, which expressly does. Furthermore, Plaintiff's construction of "virtual machine instruction," which is long and contains unnecessary and potentially confusing qualifying terms, is not logical when construed together with Plaintiff's short and relatively simple construction of "virtual machine," particularly considering that the latter term is used only in the context of the former term in the '477 patent. Moreover, Plaintiff's construction of "virtual machine instructions" does not comport with the Court's construction of the term "native code." The Court also notes that Defendants' constructions are consistent with the Court's construction of "Virtual Machine Monitor," a closely related term. Defendants' constructions are logical and give the disputed terms their plain and ordinary meaning; thus, the Court adopts them.

B. Encapsulating a Machine Executable Instruction ... with a Predetermined Sequence of Machine Executable Checking Instructions ...

The parties initially disputed the meaning of the phrase "encapsulating a machine executable instruction ... with a predetermined sequence of machine executable checking instructions" as it is used in claims 1 and 7 of the ' 477 patent. Defendants construed the phrase to mean "associating a machine executable instruction ... with a predetermined sequence of machine executable checking instructions." Plaintiff's proposed construction, prior to the *Markman* hearing, was

generating a sequence of machine executable instructions by bracketing a machine executable instruction ... with a sequence of machine executable checking instructions, determined prior to encapsulating the machine executable instruction, resulting in a contiguous machine executable instruction sequence consisting of the machine executable preceded or succeeded by checking instructions.

Plaintiff now construes the phrase to mean

generating a sequence of machine executable instructions by inserting a predetermined sequence of machine executable checking instructions into a sequence of instructions that contains a machine executable instruction to be checked. The predetermined sequence of machine executable checking instructions may be

inserted before and/or after the machine executable instruction to be checked.

Defendants do not dispute Plaintiff's revised construction of the disputed phrase. Thus, the Court adopts it.

C. Preventing Modification of Illegal Memory Addresses

The parties dispute the meaning of the phrase "preventing modification of illegal memory addresses" as it is used in claim 7 of the '477 patent. Plaintiff construes the phrase to mean "preventing the virtual machine instruction from writing to illegal memory addresses," while Defendants' proposed construction is "not allowing the alteration of illegal memory addresses."

Neither party gives a persuasive argument as to why this claim phrase should be given anything other than its plain and ordinary meaning. Defendants merely rewrite the claim phrase, while Plaintiff offers no support for its argument that only "the virtual machine instruction" is prevented from modifying illegal memory addresses. The claim discloses a "method for preventing modification of illegal memory addresses *during execution of a sequence of virtual machine instructions,*" '477 patent 16 :10-12 (emphasis added), but Plaintiff has cited no intrinsic or extrinsic evidence for its proposed construction.

The disputed claim phrase is given its plain and ordinary meaning.

V. '715 Patent

A. Kernel

The parties dispute the meaning of the term "kernel" as it is used in claims 12 and 42 of the '715 patent. Defendants initially construed the term to mean "core part of a system software." After the *Markman* hearing, Defendants submitted the following revised construction: "core part of a control program that can manage memory (and perhaps allocate one or more system resource, as described below)." Plaintiff's proposed construction is "the core of an operating system; responsible for managing memory, launching applications, and allocating system resources."

Defendants' revised construction significantly narrows the dispute between the parties. Previously, the dispute was whether a "kernel" referred to the core part of an operating system, or whether it could refer to the core of any "system software." At the *Markman* hearing, the Court asked Defendants to clarify what, other than an operating system, could constitute "system software." In their supplemental briefing, Defendants seem to acknowledge that the "kernel" claimed by the '715 patent is the kernel of an operating system. Nevertheless, they propose to substitute the term "control program" for "operating system" without intrinsic or extrinsic evidence to support the change. Thus, the substitution is not appropriate.

Further, Defendants, in their supplemental brief, accept that the "kernel" manages memory and allocates system resources. The only function they do not accept from Plaintiff's construction is "launching applications." In its papers, Plaintiff offers no evidence to support including that function in the construction of the term. Thus, "kernel" is construed to mean "the core of an operating system, responsible for managing memory and allocating system resources."

B. Kernel Address Space

The parties dispute the meaning of the term "kernel address space" as it is used in claims 12 and 42 of the

'715 patent. Defendants originally construed the term to mean "grouping of addresses to which the kernel has access," and Plaintiff's initial construction was "the address space *that contains the kernel* and that cannot be referenced by applications." After the *Markman* hearing, Plaintiff revised its construction to "grouping of addresses that the core of an operating system can access, but which other programs cannot access." Defendants now propose "grouping of addresses to which the kernel has access, *including the grouping of addresses containing the kernel*."

Plaintiff argues that its limitation is appropriate because, although "address space" may share memory, the "kernel address space" disclosed in the '715 patent does not. Plaintiff cites the deposition testimony of Defendants' expert Dr. Kelly in which he states that, in the preferred embodiment of the '715 patent, the "kernel address space" cannot be accessed by application processes. Plaintiff also cites prosecution history which states that the '715 patent discloses a method by which data is transferred "without the use of any shared memory or shared stack between the address spaces of the client and server processes." Pl.'s Opp' n, Ex. D.

Defendants argue that "address space" in the disputed claim term should be given its plain and ordinary meaning, which both parties agree would render Plaintiff's limitation inappropriate. Defendants also argue that Plaintiff's construction would exclude the preferred embodiment disclosed in Figure 6 of the '715 patent. In the preferred embodiment, a portion of the server process is mapped into the kernel address space. '715 patent 12 :19-21. Defendants note that Plaintiff's expert Dr. Bershad has acknowledged that the mapped page is part of both the kernel address space and the server address space; both address spaces have access to the data in the mapped page.

Plaintiff's construction thus contains an inappropriate limitation; Defendants' revised construction conveys the disputed claim term's plain and ordinary meaning, and the Court adopts it.

C. Authorized Address Space

The parties dispute the meaning of the term "authorized address space" as it is used in claims 24 and 25 of the '715 patent. Defendants construe the term to mean "address space that contains a program/process that is authorized to access other address space(s)." Before the hearing, Plaintiff proposed: "the address space that contains the kernel and that cannot be referenced by applications." In its supplemental brief, Plaintiff construes the term to mean "grouping of addresses that the core of an operating system can access, but which other programs cannot access."

Again, Plaintiff may not import improperly the limitation "but which other programs cannot access" into the claim term. Defendants also assert that Plaintiff cannot construe "authorized address space" as containing the "kernel"; in other words, Plaintiff maintains that "kernel address space" and "authorized address space" are synonymous. Plaintiff's central argument on this point is that the patent does not disclose any "authorized address space" other than the "kernel address space." However, Plaintiff's cites to the patent specification do not support its contention. Moreover, apparatus claims 24 (containing "authorized address space") and 42 (containing "kernel address space") are very similarly worded; thus, construing the terms identically could render one or more claims superfluous.

Defendants' construction gives the disputed term its plain and ordinary meaning, and the Court therefore adopts it.

CONCLUSION

The Court construes the disputed terms and phrases as stated above.

IT IS SO ORDERED.

N.D.Cal.,2005.
VMWare, Inc. v. Connectix Corp.