Portfolio Media. Inc. | 860 Broadway, 6th Floor | New York, NY 10003 | www.law360.com
Phone: +1 646 783 7100 | Fax: +1 646 783 7161 | customerservice@law360.com

How Structural Claim Limitations Can Save Software Patents

Law360, New York (February 9, 2016, 11:09 AM ET) --

Michael D. Stein

Critics of software patents argue that patents for software stifle innovation and impose unnecessary costs.[1] A major area of concern with software patents is the way in which the claims are written, i.e., using arguably ambiguous and overly broad functional claim limitations. In this article, I won't attempt to argue the issue of whether patents are needed to incentivize innovation and public disclosure in the software industry. Instead, I will review why functional claim limitations are used and how they can sometimes be unclear or overbroad. In this regard, I will then discuss how structural claim limitations may be used effectively to avoid the alleged problems created by the use of functional limitations in software patents. I believe that the proper use of structural claim limitations can save many a claim from invalidity on §101 or §112 grounds. My hope is that this will generate further thought and discussion about how we might obtain strong and reliable protection for software-related inventions.

Software Patents and Functional Claim Limitations

Background

Commentators have argued that patents do less good and cause more harm in the software industry than in other industries. A major area of concern with software patents is the way in which the claims are written using functional, rather than structural, terms. It is frequently argued that such claims are overbroad and unclear, and that the aggregation of such claims can result in innovation-stifling patent thickets.[2]

As discussed by Mark Lemley, supra, patent law has faced this problem before. According to Lemley, when Congress rewrote the Patent Act in 1952, it said that patentees could write their claim language in functional terms, but when they did so the patent would not cover the goal itself (i.e., the ultimate function being claimed), but only the particular means of implementing that goal described by the patentee and equivalents thereof. Such "means-plus-function" claim elements are now governed by 35 USC § 112(f).

Most software patent claims today are written in functional terms. Lemley argues that courts could prevent overclaiming by software patentees, and solve much of the patent thicket problem that besets software innovation, if they would apply 35 USC § 112(f) to all claim elements that recite functional limitations. Lemley and others further argue that a problem with functional claiming in software patents is the uncertainty associated with the meaning and scope of the claims. The software field is somewhat unique, e.g., compared to chemistry and biotechnology, because there is no standard language for software patents. Accordingly, it can be difficult to know what a software patent covers until a judicial body, such as a court or the Patent Trial and Appeal Board, has construed the claims and issued a ruling.

The alleged problems created by software patents have led commentators to argue that that software should not be patentable, or at least the scope of software patents should be severely limited. Such commentators support this position by arguing that patent protection for software innovations is not necessary to spur innovation.[3]

The attack on software patents has been very effective as we've seen a dramatic increase in the number of patents invalidated based on the U.S. Supreme Court's decision in Alice, which is now a poster child for the anti-software patent sentiment.[4] Since this decision was handed down last year, courts applying Alice are invalidating software patents at a previously unheard of rate under the so-called "abstract idea exception" of Section 101.[5]

USPTO-led Executive Action for Clarity in Patent Claims

On June 4, 2013, President Obama announced five executive actions "to help bring about greater transparency to the patent system and level the playing field for innovators."[6] In this regard, the White House stated that "stakeholders are concerned about patents with overly broad claims — particularly in the context of software." Then, on Feb. 20, 2014, the president announced three new initiatives aimed at encouraging innovation and strengthening the "quality and accessibility of the patent system."[7] The initiatives undertaken by the U.S. Patent and Trademark Office are enumerated as Executive Actions 1-7, and a summary can be found here.

Of the initiatives undertaken by the office, Executive Action 2, Clarity in Patent Claims (formerly "Tightening Functional Claiming"), is relevant to the present article. A rationale behind this effort is the notion that patent claims with clearly defined boundaries help to avoid costly and needless litigation. Consequently, the Office is taking steps to improve claim clarity and scrutinize functional limitations. The patent office has implemented a multiphased training program for all examiners, which focuses on evaluating functional claiming and improving the clarity of the examination record. The USPTO has also created a webpage identifying key decisions from the PTAB involving functional claiming. The decisions include (a) Precedential Decisions reviewing functional claim language under 35 U.S.C. §112; (b) Informative Decisions interpreting 'processor for' and/or means-plus-function claim language under 35 U.S.C. § 112; and (c) Representative AIA Trial Decisions interpreting functional claim language in trial matters. We review and discuss these decisions in the Appendix below.

Why Functional Claim Limitations Are Used in Software Patents

There are two likely explanations for why functional claiming is so common in software patents:

1. As a matter of technology, a principal feature of computer software is that structure and function can be separated. A software developer can often write new software without knowing details of the hardware on which the software will run.

2. As a matter of language, a distinguishing characteristic of software is a lack of a commonly accepted "vocabulary" for defining software elements. One way to draft broad patent claims is to define a broad group of things, which works well in chemistry and biotechnology, where we have a standard language that allows us to define a group of chemicals and determine later whether a particular chemical is in the group.[8] But in software, a broad claim requires defining the invention at a higher level of abstraction. It is common for software developers to coin new terms to define the functional elements of the software, and the meaning and scope of such new terms is often not explicitly defined in the patent. This can make it very difficult for a reader to understand the precise scope of protection of the claims.

Patent claims directed to software-related inventions are now receiving harsh treatment in the courts. For example, on June 11, 2015, and again on June 23, 2015, the Federal Circuit affirmed district court decisions dismissing software patent lawsuits at the pleading stage.[9]

Patent claims reciting purely functional limitations are arguably unlimited as a matter of structure, since on their face they appear to effectively cover any computing device that performs the recited function(s). Moreover, software patents often go further by reciting "capability claims" that purport to cover any device capable of performing one or more recited functions. There are numerous examples of claims reciting phrases such as "programmable selection means for," "capable of engaging," "adapted to," "for ... -ing," "operable to," and the like. See Lemley, supra n.9, at 923. The courts have read such claims to require the functionality to be programmed into the system, as opposed to covering computers that would have to be reprogrammed to perform the identified function(s). Id. Still, one could argue that the combination of a function with a trivial, or universally required, structural element can lead to patents that are overbroad. Id.

In sum, software claims are typically drafted using functional terms; i.e., the claim recites what the software does rather than what it is. The software claim elements generally do not specify particular coding approaches or modules that must be used, much less the code that implements those modules.

Functional Claim Limitations and 35 USC §112(f)

A discussion of the issues raised by functional claim limitations would be incomplete if it omitted a review of 35 U.S.C. § 112(f). This is a fairly large and evolving body of law, and a lengthy discussion is beyond the scope of present paper. However, for the interested reader, an overview of recent case law is presented in Appendix B.

For our purposes, the points I'd like to make in relation to § 112(f) are: First, the law seems to be evolving such that purely functional claim limitations will be construed as § 112(f) elements, i.e., as limited to the structure described in the specification and equivalents thereof. In addition, such claims will be limited by the scope of the written description, or invalidated as being indefinite under § 112(b) if insufficient structure is described. Accused infringers can now use § 112(b) and § 112(f) arguments to invalidate many patent claims that include a broad and largely functional recitation — a situation frequently encountered in claims involving software.

The cases discussed in the Appendix B suggest that we can think of structure, as distinguished from function, as connoting a level of constraint, as in having a specific organization or configuration. In software cases involving § 112(f) limitations, some type of algorithmic support has been deemed to be generally sufficient to satisfy the requirement for a description of corresponding structure. This algorithmic support can take the place of the traditional structure requirement used for other technology spaces.

I believe that structural limitations, whether in the form of algorithms or other constraints as discussed below, can be utilized to strengthen software patents against attacks under § 112 as well as § 101. The logic behind this thesis is straightforward: First, the recitation of structure in a claim element should be sufficient to avoid means-plus-function treatment under § 112(f). Second, the recitation of structure in a claim necessarily constrains the scope of protection and arguably makes the claim non-abstract. These ideas are examined below.

Structural Claim Limitations for Software

One could argue that a functional claim element need not be unclear/indefinite under §112(b), or overbroad under § 112(a), and that an inventor of a software product that performs a novel/unobvious function should be able to broadly protect the product. A claim including a "module for [performing the novel function]" could be drafted in such a way as to be clear and yet wide enough to cover other software products that perform the inventive function in various ways. Unfortunately for inventors in the software field, the courts and the USPTO seem to be taking the law in a different direction. By

adopting new approaches for applying the laws of §§ 112(a, b, and f), along with new interpretations of § 101, the courts and PTAB are finding various ways to invalidate functional claims for indefiniteness, over-breadth (lack of enablement), or as being directed to patent-ineligible subject matter. This leads to the question: Could "structural" claim limitations serve as a suitable alternative for functional claim limitations in software cases? To answer this, we need to understand what is meant by "structure", and then we need to devise a way to describe software in terms of such structure.

Describing Computer Software in Terms of "Structure"

In fields as diverse as mechanics, cell biology and anatomy, it is often said that structure dictates function. This maxim applies to the software field as well, but requires extra work to appreciate how a programmed computer can be described in terms of "structure." I believe that patents directed to software-related inventions could be improved by including a structural description of the software elements in the specification, and that this would be a good alternative to providing only an algorithm or functional description. A structural description, if included in the claim, should be sufficient to avoid means-plus-function treatment under § 112(f). Moreover, structure by definition implies constraint, the opposite of abstractness. Accordingly, under Alice, a structural description would seemingly make a claim much more difficult to invalidate as a patent-ineligible abstract idea.

Computer software refers to organized collections of computer data and instructions, which can be divided into two major categories, system software and application software. The application software interfaces with the user and system software, and the system software interfaces with the hardware, as illustrated in Fig. 1. The system software includes the operating system software and firmware, as well as any middleware and drivers installed in the system.[10] The system software provides the basic non-task-specific functions of the computer. In contrast, the application software is used to accomplish specific tasks. Each native instruction is stored in a memory device and is represented by a numeric value.

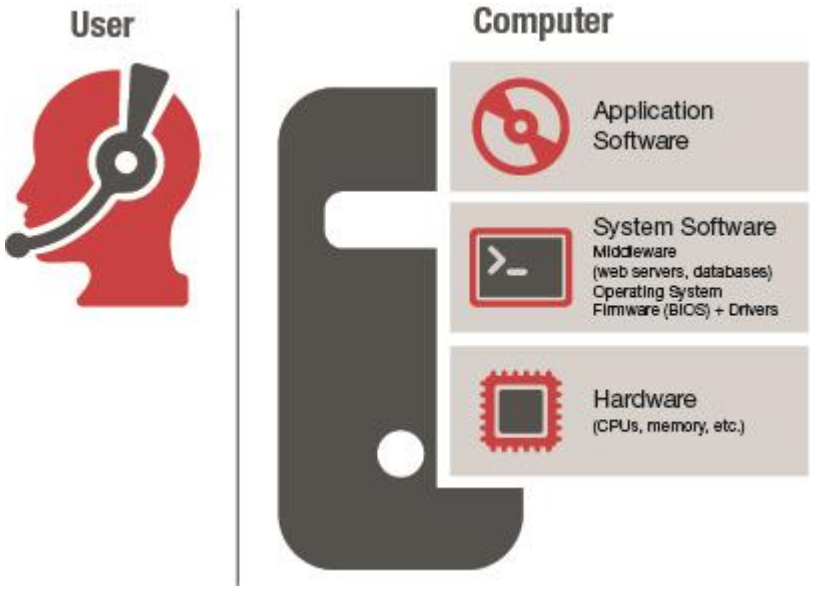


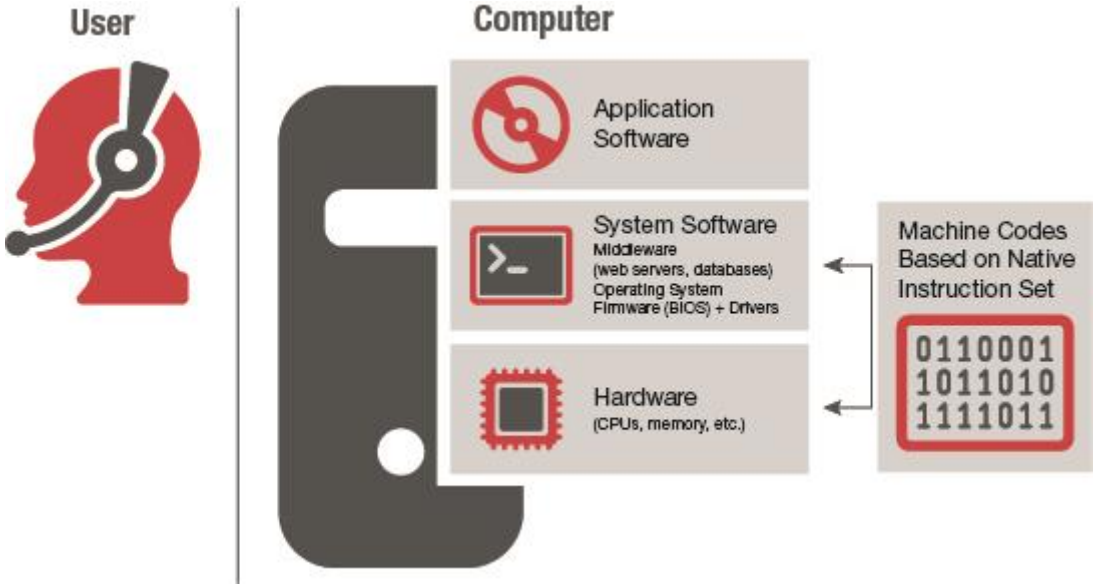
Fig. 1 is obviously an extreme simplification of a working computer system. An essential feature, which is not shown, is the necessary relationship between the executable software and the hardware processor. The software that can actually be executed by the hardware processor is not an unconstrained, abstract set of commands written down by the programmer/developer. The executable code must be committed to memory using "machine codes" selected from the specific machine language instruction set, or "native instructions," designed into the hardware processor. The machine language instruction set, or native instruction set, is known to, and essentially built into, the hardware processor(s), or CPUs. This is

the “language” by which the system and application software communicates with the hardware processors. Each native instruction is a discrete code that is recognized by the processing architecture and that can specify particular registers for arithmetic, addressing, or control functions; particular memory locations or offsets; and particular addressing modes used to interpret operands. More complex operations are built up by combining these simple native instructions, which are executed sequentially, or as otherwise directed by control flow instructions.

The inter-relationship between the executable software instructions and the hardware processor is structural in the sense that it greatly constrains any particular system implementation. In other words, the instructions per se are simply a series of symbols or numeric values. They do not intrinsically convey any information. It is the processor, which by design was preconfigured to interpret the symbols/numeric values, which imparts meaning to the instructions.

The foregoing concepts are illustrated in Fig. 2. It is my thesis that these concepts can be used to prepare a “structural description” of computer software elements. Whereas one could argue that a description of a software element in purely functional terms, i.e., in terms of what it ultimately does or the result it ultimately achieves, is unconstrained and therefore overly broad, the inter-relationship between the executable software instructions and the hardware processor constrains the way in which the hardware processor is controlled in order to achieve the desired function/result. Problems related to abstractness, indefiniteness, and overbreadth can be addressed by describing inventive computer software elements in the specification at this level of detail, and including appropriate limitations in the claims.

To be clear, I do not advocate inclusion of specific machine code or even source code listings in patent applications. (This would be unhelpful to patent examiners and the public, with the possible exception of copyists.) Instead, the technical/structural nature of the invention can be more precisely set out by describing the overall system architecture in the manner explained above. In addition, appropriate flow charts describing the logical operation of the software relevant to the inventive elements and technical features should be included.



Using the Structural Description of Computer Software for Claim Drafting

Now let’s consider how claim 8 of U.S. Patent No. 6,155,840, at issue in Williamson, supra, might be re-drafted using the ideas discussed above. Here below is a copy of claim 8 with markings indicating how it might be rewritten:[11]

8. (Amended) A system for conducting distributed learning among a plurality of computer systems coupled to a network, the system comprising:

a presenter computer system ...

an audience member computer system ... and

a distributed learning server ... comprising:

a hardware processor configured to perform a predefined set of basic operations in response to receiving a corresponding basic instruction selected from a predefined native instruction set of codes;

memory;

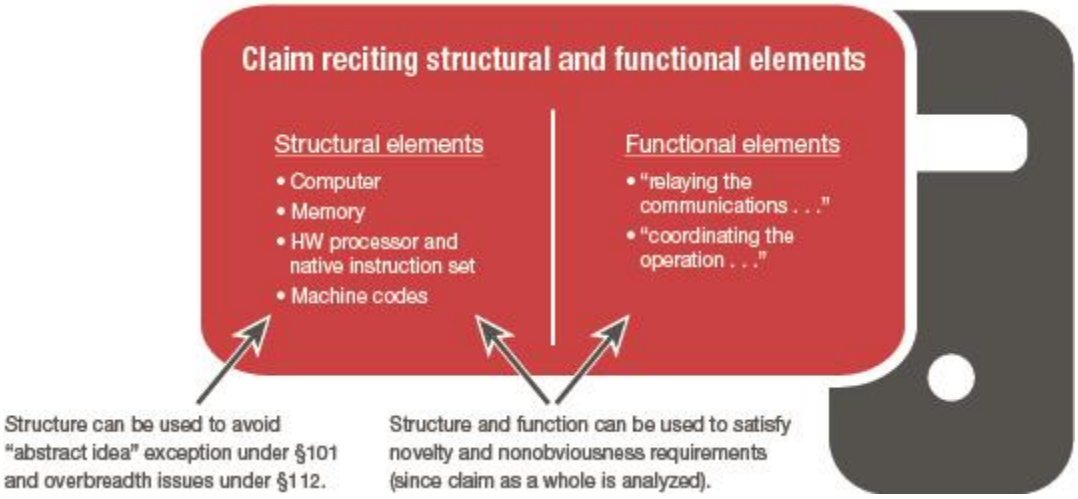
a streaming data module ... and

a distributed learning control module comprising a first set of machine codes selected from the native instruction set for receiving communications transmitted between the presenter and the audience member computer systems, and a second set of machine codes selected from the native instruction set for relaying the communications to an intended receiving computer system, and a third set of machine codes selected from the native instruction set for coordinating the operation of the streaming data module, wherein each of the first, second, and third sets of machine code is stored in the memory.

It is my view that the above version of the "distributed learning control module" element should not invoke § 112(f) because the first, second, and third sets of machine codes constitute sufficient "structure" for the respective function, i.e., so as to avoid falling within the ambit of §112(f). This approach to claim drafting should also be effective in avoiding Alice abstractness issues and overbreadth issues under § 112(a).

A potential problem with the above solution is that an anti-patent cynic could argue that the limitations added to the claim are generic limitations inherent in any modern computer, and that such limitations should be ignored for purposes of §§ 101 and 112. In my view, however, the added limitations do make a difference, since they add structure sufficient to make the claim nonabstract under § 101 and also not purely functional, thus avoiding § 112(f). In other words, whereas functional claim limitations are now generally ineffective in avoiding problems under §101 and § 112(f), structural limitations are, or should be, effective to avoid these same problems. Moreover, both structural and functional recitations should be considered for patentability under §§ 102 and 103. This is somewhat analogous to the situation in Europe, where the dichotomy is not between structural and functional features, but rather between technical and non-technical features.[12]

I have attempted to summarize this claim drafting approach in the diagram below. In the diagram, structural features — such as the computer, memory, hardware processor, native instruction set, and machine codes — are relevant for §§ 101, 102, 103 and 112 purposes. The purely functional features — such as "relaying the communications to an intended receiving computer system" and "coordinating the operation of the streaming data module" — are relevant for §§ 102 and 103 purposes, but are generally given little or no weight for §§ 101 and 112 purposes.



Conclusion

Describing software elements in terms of "structure" may be a good alternative to an algorithm or functional description, or in addition to these. If included in a patent claim, a structural description of the kind described above may be sufficient to issues under §§ 101, 112(a), and 112(f).

—By Michael D. Stein, BakerHostetler

Michael Stein is a partner in BakerHostetler's Seattle office.

*Additional information on the issues discussed in this article can be found **here**.*

The opinions expressed are those of the author(s) and do not necessarily reflect the views of the firm, its clients, or Portfolio Media Inc., or any of its or their respective affiliates. This article is for general information purposes and is not intended to be and should not be taken as legal advice.

[1] See, e.g., A question of utility, The Economist, August 8, 2015.

[2] Mark A. Lemley, Software Patents and the Return of Functional Claiming, 2013 Wis. L. Rev. 905, 906.

[3] Commentators argue that software patents are less necessary to spur innovation than are patents in pharmaceuticals and biotechnology because software innovation is less costly than innovation in the life sciences. Those commentators point to the existence of a vibrant open source community as evidence that software innovation can flourish absent patent protection. See Lemley, supra n.1, at 935.

[4] Alice Corp. Pty. Ltd. v. CLS Bank International, 134 S. Ct. 2347 (2014)

[5] See Curiouser and Curiouser Is Alice the Long Sought Troll Killer (The Legal Intelligencer); and Guest Post: In Rush to Invalidate Patents at Pleadings Stage, Are Courts Coloring Outside the Lines? (Patently-O).

[6] See White House FACT SHEET (June 4, 2013).

[7] See White House FACT SHEET (February 20, 2014).

[8] See Lemley, supra n.9, at 919.

[9] In OIP Technologies v. Amazon.com, the underlying patent claimed offer-based price optimization. The court characterized claim 1 as having the following relevant limitations: (1) testing a plurality of prices; (2) gathering statistics generated about how customers reacted to the offers testing the prices; (3) using that data to estimate outcomes (i.e. mapping the demand curve over time for a given product); and (4) automatically selecting and offering a new price based on the estimated outcome.

In IPC v. Active Network, the underlying patent claimed a method that retained information lost in the navigation of online forms. The patent at issue was US Patent No. 7,707,505.

[10] This description is consistent with the commonly accepted understanding of computer software. For example, see this definition.

[11] For this exercise, I assume that the specification provides support for these amendments.

[12] In Europe, excluded subject matter is defined by Article 52(2) EPC. If a claim recites non-technical features together with at least one technical feature, then the EPO will not reject it for relating to excluded subject matter under Article 52(2) EPC. However, only technical features are considered for novelty and inventive step. See Article 52 (2) EPC; T208/84 (Vicom); and T931/95 (Pension Benefits).