# UCC Bulletin

A monthly newsletter highlighting and commenting upon recent noteworthy developments relating to the Uniform Commercial Code

February 2001 West Group

## IN THIS ISSUE . . .

## THE PROBLEM OF EMBEDDED SOFTWARE IN UCITA AND DRAFTS OF REVISED ARTICLE 2*

*Philip Koopman, Ph.D., Associate Professor, Electrical & Computer Engineering Department, Carnegie Mellon University; and Cem Kaner, J.D., Ph.D., Professor, Department of Computer Sciences, Florida Institute of Technology*

*[This is the first part of a multipart article. The first two parts focus primarily on UCITA; the rest will focus more on Article 2. All references to UCITA are to the amended, commented draft dated July 28–August 4 2000, available at www.law.upenn.edu/bll/ulc/ucita/ucita1200.htm.—Ed.]*

### 1. Introduction

If one of the key purposes of commercial law is to facilitate commerce, we should listen carefully to the repeated pleas from business advocates (such as many Chambers of Commerce) to simplify the law and avoid needless regulation. Regulations should serve a societal purpose whose value outweighs the cost of compliance. The treatment of software embedded in goods in UCITA and proposed revisions to Article 2 does not serve this purpose.

UCITA and some recent drafts of Article 2 attempt to make distinctions between embedded and non-embedded software. This issue was discussed repeatedly in the Article 2B/UCITA drafting committee meetings (Kaner attended 15 of the 16 meetings). The drafters and most of the observers repeatedly agreed that embedded software, such as the software that controls the fuel injectors in your car, should be governed under Article 2 rather than UCITA. Recent meetings of the Article 2 drafting committee and the 1999 annual meeting of the American Law Institute considered proposals to take non-embedded software (such as financial application software on mainframes or office productivity software on desktop PCs) outside of the scope of Article 2.

We believe that these distinctions are

* Copyright 2001 Philip Koopman and Cem Kaner; permission to reproduce in UCCSEARCH™ granted.

Vol. 43, Release 1

fundamentally flawed. They are based on an outdated view of embedded software, from a time when computer processors and memory were expensive, energy consumption of computer components was high, and the computational power of inexpensive processors was very limited. To complicate the problem further, the proposals treat software that is embedded in a computer (or in a computer peripheral) as non-embedded. This exception (software embedded in a computer or peripheral) swallows the rule (embedded software should be treated differently).

It is increasingly difficult to distinguish between embedded and non-embedded software as embedded systems and desktop computing merge into a more integrated computing environment. The simplest approach is to make the distinction a matter of contract. If the software and the associated goods are sold together, under the same contract, treat the software as embedded in the goods. Alternatively, if the software is sold or licensed under a separate contract, treat the software as being separate and non-embedded. (We've been told repeatedly that this is essentially the distinction made by the United States military when it acquires hardware and software.)

We're not fond of this distinction because it says that a vendor can bring any software within or out of UCITA simply by choosing to use one contract or two. However, this rule has two important advantages:

- It is absolutely clear. Both parties to the contract and the court will be able to quickly decide whether software sold (or licensed) with goods is to be treated as part of the goods.
- The rule doesn't drive companies into making artificial engineering design trade–offs in order to bring a product within (or out of) UCITA.

Based on those advantages, Kaner has repeatedly proposed, at meetings of the Article 2B/UCITA drafting committee, the Article 2 drafting committee and at the 1999 ALI annual meeting, that this be the distinction made within UCITA and Article 2. The proposal was repeatedly rejected.

We are not advocating this simple distinction in this article. Instead, we are raising it here as a baseline for comparison.

Suppose that Criterion X is proposed as a distinction between embedded and non-embedded software, such that software is considered non-embedded (and is thus governed by UCITA) if it meets Criterion X and is considered embedded (Article 2) if it fails Criterion X. One way to evaluate the merit of Criterion X is to ask how hard it would be for a designer of a product to make the product meet or fail Criterion X intentionally. For example, how hard would it be for a car manufacturer to make its fuel injector software satisfy Criterion X? If the manufacturer can pick its governing law by making relatively simple engineering choices, then the law is equivalent to the rule that the software is embedded if it is sold with the hardware, and non-embedded if it is the subject of a separate contract. For a manufacturer in this situation, Criterion X is merely a regulation and the cost of bringing the software within UCITA is merely the cost of complying with the regulation. Such a regulation has social value only if the actions required to achieve compliance create some value, such as by making the product safer or more reliable, or by bringing income to the state.

## 1.1 Three Questions to Evaluate Criteria for Distinguishing Embedded from Non-Embedded Software

The drafting committees keep entertaining new proposals for criteria that purport to distinguish between embedded and non-embedded software. We'll consider some of them in this paper, but we can't anticipate all the new variations that will be proposed. We suggest that when you see a proposed criterion, you ask three questions:

- Does this distinction go to the heart of the difference between embedded and non-embedded software or does it merely reflect a difference in how these types of software are (as far as you or the drafters know) commonly implemented today?
- What would it take for a manufacturer to redesign its product in a way that brings the embedded software under UCITA? Are there examples already on the market that most people would consider to be embedded products that would fall under UCITA without such modifications?
- Suppose that a manufacturer made the least-cost design changes that bring its embedded software under UCITA. What are the expected impacts of the changes? For example, do we expect the resulting product to be safer? Easier to set up and use? Less likely to need repairs?

We respectfully suggest that a distinction that doesn't go to the heart of the difference between embedded and non-embedded software, that can be worked around easily and whose likely workarounds won't improve the product, is a bad distinction, in effect a regulation of the kind that is reviled by businesses because it imposes a cost on the business for no good purpose.

**2. Why Does This Difference Matter?**

An argument that surfaced repeatedly in the Article 2 meeting in St. Louis (November 18–19, 2000; Kaner attended the meeting) was that the distinction is not very important. It was suggested that the Article 2 rules are not so different from common law or UCITA rules that it would be worthwhile to intentionally bias the design of a product in order to bring it within one law instead of another.

However, at the same meeting it was repeatedly suggested that Article 2 might fail due to industry opposition if we did not exclude software from its scope. Evidently, the distinction between Article 2, UCITA, and common law rules is material to some influential people.

Let us suppose that a manufacturer is designing a product that includes software and can bias the design in a way that brings the software under Article 2 or under UCITA. Here are just a few of the advantages that the manufacturer would gain by selecting UCITA. (Note: for detailed references, please see Cem Kaner, *Software Engineering & UCITA*, 18 J. Comp. & Info. Law 435 (Winter, 1999; available online at www.badsoftware.com/engr2000.htm.)

- For non-mass-market products, there is no perfect tender rule.
- The clickwrap/shrinkwrap contract formation model is explicitly adopted in UCITA but not in current Article 2 and not so completely embraced in recent drafts of revised Article 2.
- The implied warranty of merchantability in UCITA is more vendor-friendly than the one in Article 2.
- Under UCITA, a disclaimer of implied warranty can meet a requirement of conspicuousness even if it is unavailable for inspection by the customer prior to the sale.
- Under UCITA, it is easier to argue that some feature or aspect of performance that appeared in a product demonstration does not give rise to an express warranty than under Article 2.
- UCITA defines material breach differently from the common law (Restatement of Contracts, Second, Section 241), and the definition is more favorable to the vendor.
- UCITA allows the vendor to place significant use restrictions on the product, restricting how, when, where, how often and for what purpose the customer can use the product. Such restrictions are unusual in a sale of goods and not explicitly authorized in Article 2.
- UCITA allows the vendor to place transfer restrictions on the product, blocking the customer from reselling the used product. Such restrictions on alienation are foreign to a sale of goods and have historically been unenforceable in mass-market sales of intellectual property-based products such as books and records. UCITA has been sharply criticized because it "circumvents copyright's first sale doctrine." Committee on Copyright and Literary Property of the Association of the Bar of the City of New York (cosponsored by the Communications and Media Law Committee and the Entertainment Law Committee), Report on a proposal of the National Conference of Commissioners on Uniform State Laws to adopt a proposed Uniform Computer Information Transactions Act (June 21, 1999 at 18), available online at www.2bguide.com/docs/Copy.Comm1.pdf.

We think that some manufacturers will consider these differences significant enough to be willing to spend some design and implementation money in order to bring their products under UCITA.

Here's an example. The modern automobile often has over a million lines of embedded code. Software controls the feel and effectiveness of the brakes, the smoothness of the ride, the feel of the suspension, the feel of the steering, the fuel efficiency and responsiveness of the car, security features, many aspects of the maintainability of the car and so on. These all play significant roles in buyers' decisions between car models, and so they are arguably material to the transaction.

Imagine the consequences of bringing a car's embedded software within UCITA. Apart from the liability issues, just think of the transfer issue. The car manufacturer would be able to say that when you buy your new car, you cannot transfer the car's software to another person without paying a substantial transfer fee to the manufacturer. You or your customer would either have to pay the fee or buy (at substantial expense) replacement software for dozens of computers scattered throughout the car. This will drive up the price of used cars, making them less competitive with new cars.

Additionally, note that under UCITA section 104, if the embedded software is governed by UCITA and if access to that software is a material part of the transaction, then the vendor can also bring the rest of the transaction within UCITA. If the software in your car is material to the transaction, the manufacturer can bring the whole car sale within UCITA.

We believe that this capability alone (the ability to require customers to get the manufacturer's permission and pay a fee in order to resell a used product) would be a significant incentive to many manufacturers.

### 3. UCITA's Treatment of Embedded Software

UCITA purports to apply only to what might be called "general purpose" computers and their peripherals. To a casual computer user, this brings to mind visions of a Windows™ PC or a Macintosh™, keyboards, mice and printers. However, the actual wording in the UCITA documents is vague and inconsistent and actually includes a wide variety of everyday items used by ordinary people, professionals and businesses. The official comments do not remedy this situation and, in fact, contain substantial technical inaccuracies and apparent contradictions on the point of embedded systems.

We cannot walk through a detailed analysis of the comments in the space available in this paper. The comments are not law, they were not reviewed in the drafting committee meetings and so we believe that judges will rely much more heavily on the black letter of UCITA than on the comments. However, we invite the interested reader to review Philip Koopman's analysis of the comments in his paper, "Why UCITA Falls Short for Embedded Systems" available online at www.ices.cmu.edu/koopman/ucita/embedded_ucita.pdf.

#### 3.1 The Exclusion of Embedded Software

UCITA defines its scope (and thus includes or excludes embedded software and associated goods) in Sections 103 and 104. Here are the most relevant parts:

> 103(b)(1) If a transaction includes computer information and goods, this [Act] applies to the part of the transaction involving computer information, informational rights in it and creation or modification of it. However, if a copy of a computer program is contained in and sold or leased as part of goods, this [Act] applies to the copy and the computer program only if:
>
> 103(b)(1)(A) the goods are a computer or computer peripheral; or
>
> 103(b)(1)(B) giving the buyer or lessee of the goods access to or use of the program is ordinarily a material purpose of transactions in goods of the type sold or leased.
>
> 104 The parties may agree that this [Act], including contract-formation rules, governs the transaction, in whole or part, ... if a material part of the subject matter to which the agreement applies is computer information or informational rights in it that are within the scope of this [Act], or is subject matter within this [Act] under Section 103(b) .... However, any agreement to do so is subject to the following rules:
>
> 104(a)(4) A copy of a computer program contained in and sold or leased as part of goods and which is excluded from this [Act] by Section 103(b)(1) cannot provide the basis for an agreement under this section that this [Act] governs the transaction.

Section 103(b)(1) appears to exclude embedded software, but the exception to the exclusion (software that is embedded in a computer or computer peripheral) swallows the rule.

#### 3.2 Defining a "Computer"

UCITA Section 102(a)(9) defines "computer" as

> an electronic device that accepts information in digital or similar form and manipulates it for a result based on a sequence of instructions.

This definition encompasses virtually all digital computers, embedded or otherwise.

The Institute for Electrical & Electronics Engineers Standard Glossary of Computer Hardware Terminology, IEEE 610.10–1994 defines computer as:

> a device that consists of one or more processing units and peripheral units, that is controlled by internally stored programs, and that can perform substantial

> computations, including numerous arithmetic operations, or logic operations, without human intervention during a run. Note: may stand alone, or may consist of several interconnected units.

This definition is generally comparable to the UCITA definition in scope. It is worth noting that IEEE 610.10–1994 sees fit to identify an "embedded computer" as a subset of the class of all computers:

> a computer system that is part of a larger system and which performs some of the requirements of that system; for example a computer system used in an aircraft or rapid transit system.

Thus, using either the UCITA definition of "computer" or the IEEE definition of "computer," the concept of a computer means all computers, not just desktop computers.

Here are some examples of devices that are computers within UCITA's definition.

**FUEL INJECTION CONTROL FOR AN AUTO**

Essentially all future automobile engine controllers will have reprogrammable memory (*e.g.*, flash memory) to reduce the potential cost of recalls in the event of a software defect being discovered, and it is common to have it even today. Currently flash memory is being used by cell phones, hard drives, network hubs, and most automotive engine controllers. (Source: http://www.bizjournals.com/sanjose/stories/1996/09/23/story6.html, September 20, 1996 article on flash memory usage and our discussions with automotive engineers.)

It is possible on newer vehicles to reprogram engine operation by updating an existing flash memory chip with a new software version, much as one updates the "BIOS" software in a desktop or laptop computer. On older cars, the equivalent operation can be accomplished by replacing either the memory chip itself or the entire engine control computer (both hardware and software). The Crossfire upgrade kit is an example of a combined software/hardware upgrade to an old vehicle's fuel injection control:

> Our Turbo City Corvette '82–'84 Crossfire Upgrade Kit gives you: a 90's computer to make the Crossfire Fuel Injection react more quickly and more accurately. The package includes the 90's Crossfire upgrade computer with Eprom, crimping pliers, connectors and instructions to change-out the '82–'84 ECM. This will improve low and midrange performance, and fuel and spark delivery. The ECM will have our custom stock Crossfire chip. Once your upgrade computer is installed, we can provide special programming for replacement chips that will compensate for almost any type of performance modification or added power accessory. Any number of custom performance or special requirement chips will be made for your request, for only $129.00 + S&H per chip. (Turbo City Performance Headquarters, Hey There Corvette Crossfire Owners! www.turbocity.com/CorvetteCrossfileECMUpgrade.htm.)

This is being sold by a third party, not by the makers of the Corvette. Other discussions on the site make it clear that the software comes with a license. Note that this chip is an EPROM (erasable programmable read only memory). Further updates can be made by bringing the car to a service center to have the chip reprogrammed or by simply purchasing a new EPROM and throwing the old one away as one would do with a disk or CD for an out-of-date piece of desktop software.

The Crossfire fuel injection software comes on a computer chip. This chip is not sold as part of any other goods—you use it by installing it in your car, but that's not how you buy it. You buy software that is contained in a good (the Crossfire replacement computer) in a manner no different than buying a video game cartridge or software on CD-ROM for a desktop computer. Note also that the primary purpose of the transaction is to give the buyer use of a computer program (the one that will change the performance characteristics of the car). Thus, the software also falls within UCITA under Section 103(b)(1)(B).

This software transaction is clearly within UCITA when you buy it from Turbo City.

- What if you bought a used Corvette and this chip was already installed in the car when you bought the car? The software is still contained in the Crossfire computer. Shouldn't UCITA still cover the transaction?
- What if you signed two contracts when you bought the used car? One contract covers the car in general, but not the Crossfire computer. The other is a transferred license to the Crossfire computer and its software. Now the software is contained in a Crossfire computer that has been sold separately to you. How could UCITA *not* cover this transaction?

What if General Motors wanted to bring its original equipment fuel injectors within UCITA? Let us consider this possibility in terms of the three questions of part 1.1 above:

- *Does the distinction go to the heart of embedded software?* The distinction between a fuel injector control chip supplied by General Motors as original equipment and an equivalent chip supplied by a third party cannot go to the heart of the difference between embedded and non-embedded software. Both chips are plugged into the same machine, and they do essentially the same things in the same ways.
- *What would it take to bring the product within UCITA?* If we accept that the Turbo City chip is covered within UCITA, then GM could achieve the same result for its own chips by selling its fuel injector chips and software as options. For example, the Corvette dealer could offer customers three different choices: one would optimize the car for fuel efficiency, the other could optimize the car for responsive handling and the third could be the Turbo City chip. On making the choice, the customer would sign a separate contract, write a separate check and receive a separate license and warranty page that is specific to the chip.
- *What is the benefit of changes made to bring the product within UCITA?* The consumer benefits gained from GM's offering a selection of fuel injector chips might outweigh the additional costs in design, testing, manufacturing, inventory management and support—or they might not.

*[Next month: More examples of "computers" as defined by UCITA.—Ed.]*

## *MATTERS OF MAJOR INTEREST*

### FAILURE OF ESSENTIAL PURPOSE BY A LIMITATION OF LIABILITY CLAUSE

*[See UCC Case Digest ¶2719.11(4)]*

The limitation of liability clause in a licensing agreement, under which the defendant was to provide computer software to assist the plaintiff's business, fails of its essential purpose. Judge Lowell A. Reed Jr. writes the opinion for the United States District Court for the Eastern District of **Pennsylvania**. Under Pennsylvania **UCC §2–719**, an exception to the limitation of liability clause in a commercial contract arises, for example, where

# • UCC Bulletin

A monthly newsletter highlighting and commenting upon recent noteworthy developments relating to the Uniform Commercial Code

March 2001 West Group

## IN THIS ISSUE . . .

## THE PROBLEM OF EMBEDDED SOFTWARE IN UCITA[1] AND DRAFTS OF REVISED ARTICLE 2[2]

*Philip Koopman, Ph.D., Associate Professor, Electrical & Computer Engineering Department, Carnegie Mellon University; and Cem Kaner, J.D., Ph.D., Professor, Department of Computer Sciences, Florida Institute of Technology*

*[This is the second part of a multi-part article. The first part and this part focus primarily on UCITA; the rest will focus more on Article 2. The first part ended in the middle of citing examples of what qualifies as a "computer" under UCITA. This part continues with more such examples.—Ed.]*

[1] All references to UCITA are to the amended, commented draft dated July 28–August 4 2000, available at www.law.upenn.edu/bll/ulc/ucita1200.htm. [Last month, all numbered footnotes along with their references in the text were inadvertently omitted during production. We regret the error.—Ed.]

[2] Copyright 2001 Philip Koopman and Cem Kaner; permission to reproduce in UCCSEARCH™ granted.

**LASER PRINTERS**

A laser printer is an electronic device. It accepts information in digital form. It stores the information in memory (often two megabytes or more). The printer processes the stored information in a sequence, interpreting it as a series of commands with associated data. The printer might interpret the stored information using Hewlett-Packard's PCL (printer control language) or Adobe's PostScript language or some other language. Whatever language is used, the information is interpreted as a sequence of instructions and the printer's execution of those instructions leads to printouts, displays or messages on the printer's control panel, or messages back to the application that requested printing services. Therefore, a laser printer is a computer within UCITA's definition, section 102(a)(9).

The software that comes with the printer, for example in the printer firmware, is a classic example of embedded software. But UCITA will not treat it as embedded because it is embedded in a

WEST GROUP
Bancroft-Whitney • Clark Boardman Callaghan
Lawyers Cooperative Publishing • WESTLAW® • West Publishing
Customer Service: 1-800-328-4880
Fax: 1-800-340-9378

computer (or in a computer peripheral).

**HOME MEDICAL DEVICE: BLOOD GLUCOSE TESTING**

Consider Johnson & Johnson's SureStep Blood Glucose Monitoring System. (For current information on this product line, go to www.lifescan.com.) Initially this device had very limited functionality. Prick your finger (to draw blood). Wipe the blood on a test strip. Insert the test strip into the LifeScan reader and the device presents you with a reading of blood sugar level. More recent versions of the device come with diagnostics and store the last several readings. Now, you can download software, get a cable, and transfer data from the monitor to your personal computer. One version of the device, which sells for under $100, is available at many pharmacies and does not require a prescription, stores up to 150 tests and automatically creates 14-day and 30-day test averages. An even newer model, the Profile, can hold 250 tests, records additional information about the user (such as insulin type and dosage), and labels tests by events (such as exercise), creating fairly detailed database records that carry an activity code, a reading, a time stamp, a date stamp, and probably other information. This can all be downloaded to a computer.

This is an electronic device. Inputs to the device are from the buttons (which are probably encoded, like a keyboard, to send a digital signal back to the central processor) and from a scanner that analyzes the blood on a test strip. The scanner output is probably digitally encoded and passed to a central processing unit that further interprets, stores, and transfers the results. The handling of the button inputs, the display of information on the device's LCD screen, analysis of the blood, and the transfer of results to another computer are all done in accordance with a stored series of instructions. Therefore, we believe, this device is a computer within the definition of UCITA section 102(a)(9).

By the way, even if you don't think of this device as a computer, you should think of it as a peripheral because it collects data that it can transfer to a computer. Therefore, even though this seems to be an obvious example of an embedded computer running embedded software, the software will be governed by UCITA.

Another consideration for this device, if you don't think of it as a computer, is to ask what would be needed to make it a general enough purpose computer to fit within your interpretation of UCITA section 102(a)(9) (or some other appropriate definition of a computer). For example, we could add other inputs. Maybe it would take your blood pressure (computerized blood pressure measurement tools are popular in the home medical market). Or it could test you for pregnancy. Check the potassium levels in your blood. Check your children's blood for evidence of drug abuse. Check your blood alcohol levels before you drive. Or it could connect to your family doctor through a web connection (embedded web controllers are available inexpensively, see below), uploading data and perhaps browsing your medical records for you. Perhaps you could use it to check the status of your prescriptions at your pharmacist's web site. These "enhancements" are possible and at some point this device must fit any reasonable definition of computer. However, the functions all look like traditional embedded hardware/software functions to us. We certainly *wouldn't* want such a device covered by UCITA, and in terms of societal value, the more functionality that you add to a device, the more risk of error. A manufacturer might inflate the feature count in order to bring the product under UCITA but thereby achieve a less safe or less reliable device as a result.

**OTHER DEVICES THAT ACCEPT ANALOG INPUTS**

The Motorola 68020 is the central processing chip that controlled the Macintosh II series computer, as well as the Sun Microsystems series 3 Workstation, both classic general purpose desktop machines. As faster central processors came to the market, the 68020 moved into laser printers, becoming a workhorse for running Adobe Postscript. Today, versions of this chip are still used in many industrial applications, and in those cases the chip and its software look like an embedded computer and software.

The 68020 doesn't interface directly with the physical world, but an embedded version of that same processor called the 68332 does. The 68332 is a 68020 processor with additional features, including 16 digital Input/Output pins to interact with the external world. (The fact that the pins are digital does not prevent them from being used to accept analog inputs and produce analog outputs with appropriate support circuitry added external to the processor chip.) Both chips can in general execute the same software and use the same engineering development tool set. In the last decade, millions of 68332 chips have been used as engine controllers in cars from General Motors and Ford, among other uses. The newest version of this processor family (now marketed as the Coldfire processor by Motorola) also includes an Ethernet network controller on the same chip as the processor, and emphasizes the fact that it can run the same general software as the 68020 as a major selling point.

The 68332 is clearly "an electronic device that accepts information in digital or similar form and manipulates it for a result based on a sequence of instructions." The recently announced MCF5272 Coldfire chip can do not only that, but also run the Linux operating system and communicate directly with the Internet. Surely, these chips are no less a computer simply because they can control a car engine or a cash register. On this argument, then, despite the definition of computer in UCITA section 102(a)(9), an electronic device that accepted information in analog form but manipulated it in digital form for a result based on a sequence of instructions should also be a computer, and it certainly would be a computer within the definition provided by the IEEE.

## ELECTRONIC GAMES AND SEWING MACHINES

A product like the Nintendo Game Boy lets you play computer games within the UCITA definition of computer. The Game Boy is an electronic device. It accepts inputs (such as movements of a joypad) that are converted to digital values, and responds to your inputs according to a stored program. Additionally, the Game Boy can run many different programs (computer game cartridges contain programs) and new games can be developed at any time that can run on the Game Boy, without requiring any modification of the Game Boy itself. Game Boys can even be expanded with computer peripherals, including a printer and a digital camera.

The Game Boy's versatility as a computer shows up in its use within Singer's Izek sewing system (www.singershop.com/whats_new.html, accessed 12/26/00). This system will automatically sew stitch patterns, buttonholes, and lettering. The system "includes a sewing machine, Game Boy, connection wire and special cartridge that contains stitch patterns and designs." These programs are not computer games. The Game Boy is therefore capable of running applications of various types (games, sewing, and therefore probably many others).

One would normally consider the Game Boy to be an embedded computer in a situation like this. The programs to sew stitch patterns, buttonholes, and lettering are focused on driving the output device (the sewing machine) in a limited number of ways. However, UCITA makes no provision for a device that is sometimes a general purpose computer and sometimes a narrow embedded device. Therefore, the sewing machine software should be, under UCITA, treated as if it were non-embedded because it runs on a computer.

### Products That Use Embedded Web Servers

A web server is normally a big machine.[3] Recently, companies have been offering embedded web servers—a full web server on a chip.

The world's (allegedly) smallest web server is featured at www-ccs.cs.umass.edu/~shri/iPic.html. This device is smaller than a matchbox and costs less than $1. The author suggests a wide range of applications; for example:

> The iPic chip can be embedded in every appliance ... in the house. These devices and appliances can all then be controlled from your web-browser. ... Once this is all set up, you do not need the computer to control the appliances—they can communicate with each other through the power wiring and coordinate each other's activities. For instance, your alarm clock might tip off the rest of the house that its alarm time-setting has changed.

As another example,

> Many industrial computers and devices are equipped with their own remote management facilities. With technology like the iPic they can be connected to common network facilities, instead of using dedicated wires and a dedicated control terminal, for each device or equipment. All these devices, which may include HVAC equipment, climate control in offices and large buildings, lighting and power management, security surveillance and monitoring, process control equipment, and many others can now all be controlled and managed using a unified terminal and with simplified procedures. This can lead to lower costs, better management, and sometimes, even increased safety.

Another small, inexpensive web server that claims to be the smallest is the SitePlayer SP1 at www.siteplayer.com/products.htm, available in quantity for $19.95 each. The SP1 is based on the venerable 8051 embedded processor chip design, which has been shipped in more than a billion embedded systems to date. According to

[3] For descriptions of inexpensive web servers, check online at www.dell.com/us/en/bsd/products/line_servers.htm.

the SitePlayer manual:

> The SitePlayer is "the first product in a family of embedded web servers designed to enable any microprocessor-based device to become web enabled easily and inexpensively. In approximately one square inch, SitePlayer includes a web server, 10baseT Ethernet controller, flash web page memory, graphical object processor, and a serial device interface.... Example applications include audio equipment, appliance thermostats, home automation, industrial control, process control, test equipment, medical equipment, automobiles, machine control, remote monitoring, and cellular phones.... SitePlayer is a web server coprocessor that handles web protocols and Ethernet packets.... SitePlayer can also be used in some applications in a 'stand alone' mode where simple I/O can be performed."

These are electronic devices. They receive digital input. They manipulate the input, following some sophisticated programs. They are, under UCITA's definition, computers. Software that runs on them is therefore not to be treated as if it were embedded software. But these are classic examples of embedded computers and embedded software applications.

**Other Examples**

- The Palm is an example of a personal digital assistant (PDA). This type of handheld device offers several simple applications, such as managing address books, calendars, and to-do lists. This is a computer, under UCITA's definition.
- Windows CE (consumer electronics) is an operating system that is used in hand-held PCs for general purpose software such as PowerPoint, but can also be used in embedded systems. The main web site for MS embedded systems is www.microsoft.com/windows/embedded/default.asp. Their end user license agreement for embedded use of the software is at www.microsoft.com/windows/embedded/ce/licensing/sampleeula.asp. It contains all the usual disclaimers that we see for desktop software. (It should be noted that all the embedded operating system vendors we have contacted have similar licensing terms; this is not simply an issue with Microsoft's approach.) Microsoft lists companies that use Windows CE as embedded software at http://www.microsoft.com/windows/embedded/ce/guide/casestudies/default.asp. For example, at www.microsoft.com/windows/embedded/ce/guide/casestudies/idexx.asp, Microsoft describes a milk-quality testing device to ensure that only wholesome milk is sold. *Anything powerful enough to run Windows CE will easily meet the UCITA definition of a computer.*
- A digital telephone or a digital answering machine is probably a computer under UCITA.
- A digital camera is probably a computer. We recently saw an advertisement for a digital camcorder that can record digital video, take single snapshots, display the images you are about to tape or shoot on a color LCD, and print color snapshots (on a printer that is built into the camera). It can also download data to a connected computer or store data on memory sticks.
- Coauthor Kaner recently bought a Timex alarm clock as a gift. It plays (digitized) nature sounds and CDs. It has about 25 buttons. You can set several different alarms, for different days. This is electronic. It accepts what is probably digital input from its buttons. It saves settings in its memory, displays them on its LCD screen, and then acts on them by sounding an alarm or a piece of music. It appears to be a computer (as defined in UCITA). If you think it doesn't quite meet the criterion, ask yourself what would be the cheapest enhancement to the clock to make it meet the criterion. It would not take much. Then ask, what social benefit was created by that enhancement?

In conclusion, by saying that software that is bundled with a computer is not to be treated as embedded software is to say, because the definition of computer is so broad, that an extremely wide selection of embedded software will not be treated, in UCITA, as embedded. *We will review the Article 2 proposals in more detail in our next paper but we will note here that this same problem applies to them.*

Additionally, if the software that is embedded in the computer is legitimate UCITA subject matter (under Section 103(b)(1)), then under UCITA Section 104, the vendor can specify that UCITA will also cover the computer in which the software is embedded. Looking back at the list of examples, an enormous range of goods are being pulled out of Article 2 and put into UCITA.

### 3.3. Defining a "Computer Peripheral"

If the definition of "computer" was not broad enough, UCITA section 103(b)(1)(A) also specifies that software is to be treated as non-embedded if it is sold with goods that are a computer peripheral. UCITA doesn't define "peripheral."

There are two possible ways to clarify the meaning, both of which are too imperfect to be practical: definition by listing items, and definition by connectivity.

An attempt to list computer peripherals has the advantage of superficial clarity. A nice tidy list such as: "printer, scanner, keyboard, hard disk drive, floppy disk drive, CD-ROM drive, DVD-ROM drive, mouse, or trackball" is reasonably specific. However, it suffers in that new peripherals are continually being introduced (e.g., joystick, parallel-port video camera, virtual reality glove). Furthermore, even a seemingly obvious list and complete list would be complicated by the issue of dual-purpose items such as hard disk drives that are used both in computers and embedded systems.

The only plausible default definition of "computer peripheral" other than a list of specific items is "anything that is attached to a computer." IEEE standard 610.10–1994 defines a peripheral as:

> ... a device that operates in combination or conjunction with the computer but is not physically part of the computer and is not essential to the basic operation of the system; for example, printers, keyboards, graphic digital converters, disks, and tape drives.

This definition conveys the same general sense and has the virtue of being internationally standard technical terminology. But this approach is fraught with problems.

Let us take the example of an ordinary laser printer, which most people would agree is a computer peripheral when placed in a normal office setting. This printer could be connected via a cable directly to a computer, making it a classical peripheral device. Or it could be attached to a local area network. Does being attached via a network make a printer not a peripheral? Most would probably say not, since it does not change the inherent property of being a printer (especially models that are factory configured to work both via network or via dedicated peripheral cable right out of the box). But, does that make anything connected to a computer via a network or other indirect connection a peripheral? If so, then very soon a dizzying array of items could become computer peripherals merely by adding a network connection to an existing class of product that most clearly is not a computer peripheral, including:

- Internet-enabled household appliances ("smart" refrigerators, ovens, and so on, which are now coming on the market)
- gas and electric meters incorporating modems to dial in meter readings (these are already widely installed)
- digital television recording devices that use a modem or cable modem to download new programming information (these are already established in the marketplace)
- sensors that use embedded web servers to display status information (or perhaps in that case a household thermostat would really be a "computer" instead of a "peripheral"—it will be difficult to really know until each and every such type of device is dealt with by litigation)

If connectivity or the ability to send or receive information to a general purpose computer is used as the defining quality of being a computer peripheral, then a number of devices that are usually not considered to be peripherals would then be transformed, by fiat, into computer peripherals. These would include such software-bearing items as:

- Telephones (cordless, corded with electronics, cellular—essentially all phones sold today), which are used as computer input devices for phone menus, whether input is touch-toned or spoken. Similarly, voice mail systems are implemented with general purpose computers and use telephones as input/output devices.
- Hotel doorknobs, which use magnetic cards or other devices to send request for entry into a general purpose computer in the hotel office.

- Smoke detectors, fire alarms, and other such safety critical devices in large buildings, which are often monitored by a general purpose computer.
- Laptop computer batteries that have integrated monitoring chips to provide charge level information to the laptop computer. While part of a computer, they are hardly what is normally thought of as a "peripheral device."

### 4. Conclusions

UCITA and Article 2 are attempting to distinguish between embedded and non-embedded software. Software that is embedded in goods will be treated as part of the goods (under Article 2) whereas software that is not embedded will be (it is proposed) taken out of the scope of Article 2 and (in states that have adopted UCITA) left within UCITA.

There are five fundamental problems with this distinction:

- In general, it is almost impossible to distinguish between embedded and non-embedded software. We can think of prototypic examples of embedded software that are not being treated that way under UCITA or draft Article 2.
- UCITA and Article 2 both treat software that is embedded in a computer (used by distinct embedded processors in its keyboard, disk drives, monitor, and so on) as if it were not embedded software. Worse, the definition of "computer" is so broad that a large portion of classically embedded software will be arguably non-embedded within UCITA.
- UCITA and some of the Article 2 proposals treat software that is embedded in a computer *peripheral* as if it were not embedded software. This pulls even more embedded software out of the scope of Article 2.
- Non-embedded and embedded software are converging. Complex collections of functionality, such as web servers, are now available as embedded software. Similarly, complex operating systems (Windows CE for example) and programming languages (Java, Postscript) are being embedded in devices. Distinguishing between embedded and non-embedded software as they become more similar will be increasingly difficult.
- Finally, because the distinctions are so fuzzy, it is often cheap and easy to adjust some nonessential aspect of the product in order to make a stronger argument that the product belongs under UCITA or outside of Article 2. Such changes will rarely, if ever, achieve any benefit other than the benefit to the vendor of having pulled the product out of Article 2.

In the rest of this article [to appear in a future issue—Ed.], we will look at distinctions that the Article 2 drafting committee has been trying to make. The same problems apply.

## *MATTERS OF MAJOR INTEREST*

### "FREE FROM DEFECTS FOR STATED TIME" EXTENDS TO FUTURE PERFORMANCE

*[See UCC Case Digest ¶¶2313.3(2), 2725.21(6), 2725.21(20)]*

According to the **Maryland** Court of Appeals (the state's highest court), a warranty that goods will have a certain quality or be free from defects for a stated time explicitly extends to future performance and is subject to the exception stated in **UCC §2–725(2)**. In relevant part, a mobile home manufacturer warranted the home "to be free from substantial defects of material and workmanship under normal use and service for a period of twelve (12) months from the date of delivery to the first retail purchaser." Thus, explains Judge Alan M. Wilner (writing for the court), the four-year limitations period did not begin to run until the expiration of that one-year warranty term. While the buyers thus had five years from delivery to file their breach of warranty action, they took over nine years to do so. For that reason, their action was properly dismissed by the circuit court.

Maryland's intermediate court, the Court of Special Appeals, had previously affirmed the dismissal, but on different grounds [40 UCC Rep Serv 2d 937]. Relying on further contract language limiting the manufacturer's obligation to a duty "to repair or replace ... any defective part or parts ..., provided that written notice of the defect is received from the purchaser ... within one (1) year and ten (10) days from the date of delivery," the Court of Special Appeals decided that the warranty did not extend to future performance because it was just one to "repair or replace." While agreeing that a

# • UCC Bulletin

A monthly newsletter highlighting and commenting upon recent noteworthy developments relating to the Uniform Commercial Code

April 2001 West Group

**IN THIS ISSUE . . .**

## THE PROBLEM OF EMBEDDED SOFTWARE IN UCITA AND DRAFTS OF REVISED ARTICLE 2*

*Philip Koopman, Ph.D., Associate Professor, Electrical & Computer Engineering Department, Carnegie Mellon University; and Cem Kaner, J.D., Ph.D., Professor, Department of Computer Sciences, Florida Institute of Technology*

*[This is the third part of an article begun in* ***UCC Bulletin****, February, and continued in* ***UCC Bulletin****, March. The first two parts (also* available online as "Part 1" at www.badsoftware.com/embedd1.htm) *focused primarily on UCITA; this one focuses more on Article 2. All references to UCITA are to the amended, commented draft dated July 28 through August 4, 2000 and available online at www.law.upenn.edu/bll/ulc/ucita/ucita1200.htm. References to the latest scope proposal for Article 2 are to the February 2001 Draft 2–103 Scope available at www.law.upenn.edu/bll/ulc/ucc2/scope01.htm.—Ed.]*

### 1. Introduction

Having failed to develop a workable distinction between embedded and non-embedded software, the UCC Article 2 drafting committee now proposes to throw the problem at judges and tell them to make the distinction on a case-by-case basis. The draft provides four factors to help the judge decide when software is embedded. Unfortunately, as we will show below, these factors will not be of much help.

**Why is this such a hard problem?**

The Article 2B/UCITA drafting committee worked long and hard on this issue. Its meetings were attended by lawyers who had computer law sophistication. *Why did they ultimately fail to make a meaningful distinction between embedded and non-embedded software?* (See *UCC Bulletin*, February and March, for a detailed discussion of the UCITA problems.)

The Article 2 drafting committee has made substantial additional efforts to tackle this issue, especially over the past year, but they haven't had any greater success. *Why not?*

In our view as computer scientists, the reason nobody has succeeded is because they are

* Copyright 2001 Philip Koopman and Cem Kaner; permission to reproduce in UCCSEARCH™ granted.

WEST GROUP
A THOMSON COMPANY

Customer Service: 1-800-328-4880
Fax: 1-800-340-9378

Vol. 43, Release 3

working on a problem that is fundamentally impossible to solve. There is no principled distinction between embedded and non-embedded software, not even if you give "embedded software" a new name, like "smart goods" or "integrated software."

## 2. Turing Equivalence and Article 2

The reason that there is no principled distinction between embedded and non-embedded software is that the decision to make a program "embedded" (whatever "embedded" means) is merely an implementation choice, subject to the usual cost/benefit tradeoffs that influence engineering design decisions. Change the costs and benefits (for example, by changing the regulatory structure associated with software) and the engineer can replace one solution with an equivalent alternative. From a Computer Science viewpoint, two alternative programs and the machines they run on can easily be equivalent even if one appears to be "embedded" and the other does not.

The concept of Turing Equivalence is based on one of the early discoveries in Computer Science (CS), the Church/Turing thesis (A. Church, "An Unsolvable Problem of Elementary Number Theory," 58 Am. J. Mathematics 346, 1936; Alan M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," Series 2, 42 Proc. London Mathematical Soc. 230, 1936.) This is a basic result, studied by all CS majors, typically in their sophomore or junior year. For a detailed presentation, see Jack Copeland, "The Church-Turing Thesis" at www.alanturing.net/pages/Reference%20Articles/The%20Turing-Church%20Thesis.html. For a gentler introduction, see Steven Harnad, "Lecture Notes" at www.cogsci.soton.ac.uk/~harnad/Hypermail/Foundations.Cognitive.Science2000/0055.html.

The Turing Equivalence principle holds that any computer can perform the same computations as any other computer, given that it has enough memory and is given enough time. The general argument to support this principle describes a very simple computer called a "Turing Machine," and then demonstrates that it can compute the same functions as more complicated machines. Thus, all computers are said to be "Turing Equivalent."

One implication of Turing Equivalence is that a program written for one computer can be made to run on any other reasonable computer. This can be accomplished by recompiling the program, by simulating the instruction set of the first machine (such as PC emulation software running on a Macintosh), or by writing the program for a so-called virtual machine (such as the Java virtual machine) that runs on any computer. The important result of this is that there is no reason why a program written for a desktop computer can't be used in an embedded system, and no reason why a program written for use in an embedded system can't run on a desktop computer. And, indeed, we now see things such as the Windows CE operating system being used for both purposes, and desktop PC designs used in embedded products to simplify development efforts. (*See* Rebecca Buckman, "Microsoft Renews Push to Develop Chips that Run Embedded Systems, Wall Street Journal (Interactive Edition), Feb. 6, 2001.)

Another implication of Turing Equivalence is that any particular computer function can be implemented in a wide variety of ways, and each of these different programs is equivalent (except for perhaps the amount of memory used and the length of time to complete the computation). As a simple example, if you want to multiply the numbers 3 and 27, you can do it in hardware by using a multiply instruction; by using a program with a set of bit-shifting instructions that simulates the way people do multiplication with pencil and paper; by using only addition (adding the number 27 three times); by using a table of precomputed products for single digits similar to the times tables we all learned in grade school; or even by combinations of these techniques. The multiplication program could be written in any of dozens of programming languages, and compiled to one specific computer, to a virtual computer running on any computer, or simply interpreted directly from the program at run time without ever being compiled to a machine-executable language. All these techniques can and have been widely used for decades in software for both embedded computers and desktop computers. The choice of implementation technique is based on economics, ease of development, and sometimes other factors such as regulatory and legal constraints. Additionally, if one has sufficient computing power, it is always possible to replace software with hardware, and special-purpose hardware with software.

Thus, attempts to legislate distinctions between embedded and desktop computing applications based on the way software and/or hardware are implemented must fail because they attempt to deny the implications of the Turing Equivalence principle. While such distinctions might be useful in describing the way systems are implemented *in the absence of artificial influences*, they are merely the results of engineering tradeoffs. They do not indicate differences in terms of theory of computation. In fact, if there were an incentive to change implementations (such as protection from legal liability or expanded intellectual property rights), it would be a very simple matter to change one implementation to another that receives the more favorable treatment. Thus, any attempt to distinguish which software should receive favorable treatment that is based on implementation approach is doomed to failure from the outset.

## 3. The New Article 2 Factors

Proposed Article 2 Section 103(b) provides:

"(b) If a transaction includes goods and a copy of a computer program, the following rules apply:

"(1) This article applies to the goods.

"(2) If appropriate, this article, including provisions that by their terms are applicable to 'goods,' also applies to the product consisting of the goods and the copy considered as a whole. Factors that may be considered in deciding whether it is appropriate to apply this article to that product include whether acquiring the copy is incidental to acquiring the goods, the manner in which the copy is associated with the goods, the nature of and circumstances surrounding the transaction, and whether the copy is subject to a license."

Let us consider these factors in turn.

### 3.1 Acquisition is incidental to acquiring the goods

How should the judge decide "whether acquiring the copy is incidental to acquiring the goods?"

The handling of a modern car is heavily influenced by the software that controls it. If you buy the car because you love how it feels when you drive it, you may be buying the car largely because of its software.

If you pay substantial amounts of money to get an Internet-enabled kitchen, is the software incidental to the appliances? What about the Electrolux "Screenfridge" (http://www.electrolux.com/screenfridge/)? This beauty, reportedly due to hit the market this year, can recognize food that is past its expiration date, can suggest recipes, order food over the Internet, etc. In these cases, the software is not incidental; it is central to the purchase.

A classic example of software being the central driving force behind purchases is the evolution of Japanese rice cooker appliances in the past decade. Old-style rice cookers had no software, and produced varying results depending on the ingredients used. Then fuzzy logic rice cookers were designed that used sophisticated control software to adjust cooking cycles to variations in water and rice. Japanese consumers turned out in droves to replace their old, perfectly usable, rice cookers with ones having this new software technology. This stimulated a flurry of "fuzzy logic" labels on products of all kinds, turning the type of control software used into the most important selling point on many consumer products. More recently, neuro-fuzzy logic rice cookers have appeared that use neural net software to dynamically adapt fuzzy logic to produce even better rice. And many people have tossed out perfectly usable fuzzy logic cookers to upgrade to cookers with neuro-fuzzy software. Thus, the introduction of new control techniques (software) has been a major distinguishing feature in the rice cooker market for years.

Software can often replace functional hardware at a lower cost of goods. More capabilities become economically feasible as chips get cheaper. The use of software to replace hardware is accelerating with flash memory or equivalent solutions that look to the customer like non-erasable, read-only memory, but afford the manufacturer or a third party the ability to upload new software as needed to improve operation or correct product defects. Your personal computer has flash memory for the "BIOS" software. Your car probably has flash memory in its engine controller, as do DVD players and even some sewing machines. More capabilities become feasible as sophisticated operating systems (such as Windows) and the associated software design tools become available as inexpensive embedded software platforms. As software plays an increasing role in your kitchen appliances, your home heating system, your car, your television, and your bed, the capabilities of the software will play a significant role in your choice and perception of the product. Furthermore, a significant fraction of the engineering effort spent on developing new products is often spent on creating software, and this trend is accelerating (Would manufacturers spend that much effort on a mere incidental aspect of a product?). As we understand the meaning of the word "incidental," over time, software will become less and less "incidental" to the sale of many traditional consumer goods. In fact, increasingly as in the case of rice cookers, the market for goods is fundamentally driven by innovations in software.

### 3.2 The manner in which the copy is associated with the goods

How should the judge interpret "the manner in which the copy is associated with the goods?"

The Article 2 draft comments suggest that a consideration is "whether its [the software's] range of functions is pre-set and cannot be modified during operation." Most off-the-shelf software products have a pre-set range of functions. It is difficult to modify most programs, especially off-the-shelf programs, while they are running. This doesn't make these programs embedded or suggest that they are like embedded programs. Some programs have options—functions that allow you to select how other functions will work—but these can be provided just as well to a person driving a car as to a person driving a word processor.

An obsolete view of embedded software is that it comes on Read Only Memory chips that can never be modified, just replaced. Some software does come this way, but increasingly embedded software is delivered and stored in a way that allows for updating of the software as necessary. The software might come on flash memory. Or it might come on disk and be uploaded to a device by the customer. Or it might come on ROM cartridges (as many computer games do) that can easily be swapped out by the customer. The choice among delivery/storage media is a classic example of engineering cost tradeoff. That choice should not determine whether we call something "embedded" or not.

### 3.3 The nature of and circumstances surrounding the transaction

How should the judge evaluate "the nature of and circumstances surrounding the transaction?"

The comments ask (a) whether the transaction occurs in the consumer market, (b) whether an alternative program is available that performs a similar function, and (c) whether it is available from a source other than the seller of the goods.

The consumer/non-consumer distinction has absolutely nothing to do with whether software is embedded. Large commercial aircraft, tanks, and nuclear reactors have a great deal of embedded software, are subject to Article 2, but are not consumer goods.

It is easy to provide customers with alternative versions of a program to control a device. This is a marketing choice, not an engineering choice. Customers can buy fuel efficient versus high performance versions of cars, and a key difference often lies in the software. Cars could just as well come with antilock brake and stability control options that use different algorithms depending on the experience and driving style of the driver.

The issue of availability from a third party also involves a marketing choice, not an engineering choice. If the law provides a strong enough incentive to manufacturers to allow others to develop and sell software that is compatible with their devices, those manufacturers will publish appropriate specifications and third parties will be able to create the software. Third-party component suppliers already write much of the software in cars, so whether they sell it directly or via car manufacturers is simply an issue of distribution and marketing agreements, not technology. Independent third parties already sell aftermarket software to control many cars' fuel injectors. (See, for example, Turbo City Performance Headquarters, Hey There Corvette Crossfire Owners! www.turbocity.com/CorvetteCrossfileECMUpgrade.htm.)

Suppose that Customer A buys a stock Corvette. Customer B buys the same car but then buys fuel injector control software from Turbo City and installs it in her car. Customer B's fuel injector software has limited functionality and exists to control a device. To us, it looks somewhat like embedded software (whatever that is). But the transaction is independent of the car sale, and the software is licensed. So under proposed Article 2, this is probably not embedded software. But if the Turbo City software is not embedded, then why should the original software be considered embedded? The one is a direct replacement for the other, they serve the same functions, they control the same device, and they are for the same customer. If one is not embedded (and therefore escapes the scope of Article 2), the other must not be embedded either. But suppose Customer C buys a Cadillac and suppose that its fuel injector software is very similar to the Corvette's (perhaps even running on the very same computing hardware), but there is no third-party replacement software on the market yet. Would this mean that Cadillac fuel injector software is embedded even though Corvette fuel injector software is not? What if Customer C's fuel injector software is then upgraded by the dealer to a newer software version that comes with a software license (and that is identical to preloaded software being shipped with newer vehicles)? What if an independent garage does the upgrade? Why should any of these circumstances have any relevance to whether software that runs a car's fuel injection is in or out of scope for Article 2?

The "nature and circumstances surrounding the transaction" factor will create enormous confusion because it sweeps into relevance facts that we think would otherwise be irrelevant in deciding whether software is embedded or not.

**3.4 Whether the copy is subject to a license**

Finally, the judge must determine whether the copy of the software was provided subject to a license.

This factor, at least, is easy. Either there is a valid licensing contract or there is not.

## 4. Evaluating Distinctions between Embedded and Non-Embedded

There is no engineering basis for a principled distinction between embedded and non-embedded software. Software is software, no matter how it is stored. However, this issue has been argued so many times, in so many meetings of the Article 2B and Article 2 committees that we doubt that we are seeing the last attempt at a distinction in the February 2001 proposal.

This section suggests a few other approaches to evaluating any proposed distinction between embedded software (feel free to substitute the equivalent phrase du jour) and non-embedded software.

We have published two other detailed analyses of proposed distinctions. The first was in November 2000, when the Article 2 drafting committee floated a distinction between integrated (goods and software) and non-integrated products. (Phil Koopman & Cem Kaner, Why the Proposed Article 2 Revisions Fall Short for Embedded Systems, available online at www.badsoftware.com/embedd0.pdf.) The second was our analysis of the UCITA distinctions in *UCC Bulletin*, February and March.

In that analysis, we suggested that you ask three questions of any proposal to distinguish embedded from non-embedded software:

1) Does this distinction go to the heart of the difference between embedded and non-embedded software, or does it merely reflect a difference in how these types of software are (as far as you or the drafters know) commonly implemented today?
2) What would it take for a manufacturer to redesign its product in a way that brings the embedded software under UCITA (or out of Article 2)? Are there examples already on the market that most people would consider to be embedded products that would fall under UCITA without such modifications?
3) Suppose that a manufacturer made the least-cost design changes that bring its embedded software under UCITA (or out of Article 2). What are the expected impacts of the changes? For example, do we expect the resulting product to be safer? Easier to set up and use? Less likely to need repairs?

We respectfully suggest that the four-factor approach of the February 2001 draft would not fare well under these questions. The factors appear not to go to the heart of any differences between embedded and non-embedded software. The engineering and marketing changes needed to change an "embedded" product covered by Article 2 to one outside the Article 2 scope would be minimal and there is no reason to expect those changes would provide anything more than a legal benefit to the company, without a corresponding benefit to society.

Gail Hillebrand, of Consumers Union (letter to Lance Liebman, director of the American Law Institute, November 30, 2000) also suggested several factors for evaluating a proposed distinction between embedded and non-embedded software (her letter called them integrated versus non-integrated products). We are rephrasing her questions as assertions:

1) Software that operates features of consumer goods like cars, stereos, home appliances, and home medical devices should always be covered by Article 2.
2) It should not be easy to "engineer around" the scope rule by changing the location or manner of delivery of programs contributing to the features of goods.
3) Merely changing the storage medium of the software, such as by using flash memory, should not make a difference to the determination of whether software is embedded.
4) Competing products in the marketplace should not be treated differently under the definition of embedded software. For example, consumers of one brand of camera should not find their transactions fully under Article 2 while consumers of another brand of camera with similar features find their purchase partially excluded from Article 2 because of how the software is delivered to or used in the product.
5) The current technological trends dealing with embedded software should not have the effect of excluding more and more goods from Article 2. Article 2's scope should not automatically and predictably be shrinking as more of the functionality of goods is implemented in

software rather than in equivalent hardware.

6) The application of the test for an embedded product, and thus for the scope of Article 2, should not depend on factual questions which will have to be litigated on a product-by-product basis.
7) The characterization of a traditional consumer product as information rather than goods should not cause a change in the warranty or intellectual property rights of the parties.

We respectfully suggest that the four-factor approach of the February 2001 draft would not fare well under these questions either.

**5. Conclusions**

As technology advances, more functions of everyday products will be implemented in software. In our view, functional products that are sufficiently finished that they could be sold in a mass market should be considered as goods, whether they consist partially of software or not. To say otherwise is to say that eventually most things we use in everyday life will no longer be goods. We don't really believe anyone is trying to intentionally remove most everyday goods from coverage under UCC 2. But the problem is, nobody has been able to propose a way to distinguish different types of software that preserves the important principle of everyday goods staying entirely within the scope of UCC 2 as one would expect.

While several approaches to distinguishing between embedded and non-embedded software based on implementation approach have been proposed, none have withstood technical scrutiny. Nor are any such proposals likely to have a sound engineering basis, because fundamental principles of Computer Science hold that if there is an advantage to any particular implementation approach, it is possible to modify any software to use that approach, whether "embedded" or not. The current proposed wording for Article 2 has the (in our view undesirable) effect of encouraging manufacturers to skew their designs in order to take products out of Article 2 (courts would probably reason by analogy to the more vendor-friendly UCITA). Furthermore, the current proposed wording would result in an ever-increasing number of products being driven out of scope from Article 2 with the normal passage of time and the routine evolution of technology.

States that have adopted UCITA have made a different policy choice, but in states that have not adopted UCITA, we see no reason for the drafters of the Uniform Commercial Code to compromise long-standing protections for customers and competitors, nor the doctrines of exhaustion, fair use, and first sale. If something looks like a first sale, and the state has not adopted UCITA, the UCC should let it stay a first sale.

Mere technical implementation decisions should not be the primary basis for a powerful public policy change that will remove the functionality of whole classes of goods from UCC protection. The existing UCC should not be weakened by removing an ever-increasing number of everyday products from its scope. Copyright and patent laws already protect the industry against piracy—state laws need not provide these essentially federal protections, especially if they are unable to do so without undermining the applicability of the UCC to everyday goods.

## *MATTERS OF MAJOR INTEREST*

### PAYMENT ARTICLES REVISION AGENDA

*[Please post your response to this agenda to the UCC e-mail discussion group. To join, go to http://lists.washlaw.edu/mailman/listinfo/ucclaw-l and follow the instructions there.—Ed.]*

**To:** Drafting Committee To Revise Payment Articles of The UCC, Observers And Advisers
**From:** Edwin E. Smith & Ronald J. Mann
**Subject:** 2001 Agenda
**Date:** March 2, 2001

*In response to the resolution adopted by the Executive Committee of the Conference in January of 2001, the Drafting Committee met in San Diego in February. The principal activity at that meeting was to formulate an agenda for the possible continuation of the project. The resolution calls for a report to the Executive Committee from the Chair and the Reporter. This memorandum constitutes a first cut of the report that the Chair and the Reporter plan to present to the Executive Committee. Part of it is in draft form as a convenient method of communication. To ensure the broadest possible opportunity for comment, this memorandum will be disseminated widely before a final version is produced for transmission to the Executive Committee.*

At the San Diego meeting, the Drafting Committee decided to recommend to the Executive Committee that the project continue. The Drafting Committee believes that there are a number of significant enough issues on which the Drafting Committee already has reached or is relatively close to reaching consensus to support that recommendation. The March 2001 Draft to be