Carla Scaletti
Symbolic Sound Corporation
P.O. Box 2549
Champaign, Illinois 61825-2549, USA
carla@symbolicsound.com

Looking Back, Looking Forward: A Keynote Address for the 2015 International Computer Music Conference

Abstract: The invention of software in the mid-20th century was as big a breakthrough for modern humans as the mastery over fire was for our prehistoric ancestors; the addition of cognitive fluidity to hardware has resulted in an explosion of experimentation and creativity (which also poses some challenges). As is the case with any new technology, humans immediately set about using software to connect with one another and extend our networks of distributed cognition. Computer musicians are uniquely positioned to predict the future by composing it and coding it, because as a group we combine the imagination and daring of artists with the technology that can make the imaginary real.

What follows is a transcript of a keynote speech given at the 2015 International Computer Music Conference (ICMC). A copy of the original slides is available online at www.carlascaletti.com/Main/LookingBackLookingForward.

Nostalgia: Not What It Used to Be

I've never been a big fan of nostalgia, but when I found out the theme of this year's conference was going to be "Looking Back, Looking Forward," I decided to do a little research on the topic.

I found out that the word "nostalgia" comes from the Greek word *nostos*, meaning homecoming or the idea of returning home from a long journey, combined with the word *algos*, meaning sorrow, grief, or pain. Historically it was regarded as a brain malfunction. Johannes Hofer, the Swiss doctor who coined the term in 1688, described nostalgia as "a neurological disease of essentially demonic cause," and some military physicians attributed it to brain damage caused by the incessant clanging of cowbells in the Alps [*New York Times*, 8 July 2013, "What Is Nostalgia Good For?"]. So far, I was still not a fan.

Then I ran across the work of Tim Wildschut and Constantine Sedikides. They lead a group of research psychologists at the University of Southampton called the Nostalgia Group, and they've published quite a bit of new research on nostalgia. Their studies seem to indicate that you can use nostalgia to establish the benchmarks of your biography, giving you a sense of meaningfulness and continuity, a connection with your past and optimism for the future. And that engaging in nostalgia can even enhance creativity by making you more open to new experiences. So I thought, "Well . . . OK, maybe nostalgia isn't all bad."

Most of these studies were focused on personal nostalgia. But I thought maybe we could engage in a form of collective nostalgia, because I know that we have some shared stories of overcoming challenges: stories where the protagonists are computer musicians. Maybe telling these stories can give us a connection to our history, a clearer idea of how to overcome current challenges, and a sense of optimism about the future of computer music.

Later on in this talk, I'm going to try sharing a couple of my computer music stories in the hopes that they'll remind you of some of your stories and that you'll share them with the rest of the computer music community.

[Editor's note: Readers are invited to send such stories to cmj-editor@mit.edu for possible publication as letters to the editor. Please include "[CMJ] story" in the subject line and mention this article in the letter itself.]

Wildschut and his colleagues say that "nostalgia entails fond evocation of momentous events in which the self and significant others occupy central roles—evocations that are often characterized

Computer Music Journal, 40:1, pp. 10–24, Spring 2016
doi:10.1162/COMJ_a_00341

by redemptive narratives where one conquers adversity" (Wildschut et al. 2006).

Which sounds to me a lot like the songs the Hobbits would sing in *Lord of the Rings*, or the inspirational stories we all heard from parents and grandparents when we were growing up, or the myths and epics and sagas that our ancestors told (or more likely sang) around the fire at night. I'm guessing that the epic sagas were sung because it's easier to remember long episodic sequences when you set them to music.

Looking (Way) Back

One of those epic myths, the tale of two brothers named Prometheus and Epimetheus, has an uncanny connection to our theme. Because Epimetheus means, literally, looking or thinking back, and Prometheus means looking or thinking forward. According to the story, Prometheus gave humans the gift of fire and taught us language and mathematics, and Epimetheus opened Pandora's box to give us evil, pain, and disease.

Immunologist Susumu Ohno used Epimetheus and Prometheus as metaphors to describe the innate and adaptive immune systems (Silverstein 2009). He called the innate immune system "Epimethean" because, like all results of natural selection, it reflects the past: It's a reaction to challenges encountered by our predecessors, so it's like looking back. And he named the adaptive immune system Promethean because it's forward-looking: The adaptive immune system can generate millions of new sequences by recursively combining a few hundred basic building blocks or modules. In that way, the adaptive immune system is able to anticipate new challenges that have never been encountered in the past. But how did it get that way? How did the adaptive immune system become modular?

Modularity

It's one thing to notice modularity in a circuit designed by a human engineer, but how could a modular system have evolved spontaneously under a process of natural selection? To try to answer that question, systems biologist Uri Alon did a series of experiments using genetic algorithms (Kashton and Alon 2005).

The idea was to use a genetic algorithm (GA) to evolve a logic circuit using only NAND gates. Alon set a goal in the form of a logic expression, then he ran the genetic algorithm and kept testing the results. It took ten thousand generations to converge on a solution, but when it finally did converge, it had found the optimal solution: the circuit with the fewest gates and connections for computing that logic expression. But it never evolved a modular solution.

Then Alon made one change to the genetic algorithm, and this one change caused the GA to converge on a modular solution. Instead of giving the GA a fixed goal, he changed the goal every 20 generations. When he did that, the genetic algorithm converged on a modular circuit. Not only that, but it converged ten times faster and didn't get stuck in long plateaus. The modular solution was never quite as optimal as the fully wired solution, but it was able to rewire itself within only five generations whenever the goal changed again.

So it appears that life on Earth may be a modular solution that evolved in response to changing conditions. And, because it is modular, it can quickly rewire itself in search of new solutions when necessary.

Recombinance (Cognitive Fluidity)

In *The Singing Neanderthals*, Steven Mithen (2006) writes that our brains probably have some specialized, domain-specific modules that developed in response to pressures in our evolutionary past, but that can't explain how we are able to do things like read a book or operate an iPhone—things that didn't exist in the Pleistocene environment. That could only be accomplished through what Mithen calls cognitive fluidity (what I would call "recombinance"), through the seamless combinations of domain specific modules.

Distributed Cognition

In 900 BCE, with the story of Prometheus, the ancient Greeks seemed to be saying that mastery over fire is what made us human. In 2009, primatologist Richard Wrangham seems to agree, primarily because fire provided the means for cooking food, and that this "outsourcing of digestion," as he calls it, was far more efficient at extracting nutrients and thus could support a larger brain (Wrangham 2009).

Which is pretty much what happened over the last 3.8 million years. A modern human brain is about three times larger than the brain of our *Australopithecus* ancestor, Lucy, and most of that growth occurred in a part of the brain called the neocortex. So how did we use this monstrously large, cognitively fluid organ in our heads to our evolutionary advantage?

The evolutionary anthropologist Robin Dunbar thinks we used our massive brains primarily to predict the actions of each other (Dunbar 2003). His theory of the "social brain" is that the human neocortex evolved in response to the complexity of maintaining social networks and predicting the actions and intentions of other individuals in a social group. He even found a correlation between the size of the neocortex and the size of a species' core social group, as well as in the amount of time that a species spends in grooming activities (like picking lice out of each other's hair). Dunbar thinks that, at some point in our development, we switched over to using a kind of auditory grooming—using laughter, music, dance, and eventually language—as our primary means for establishing relationships and maintaining the cohesiveness of our social groups.

Dunbar believes that it was probably language, since it was far more efficient than physical grooming, that enabled us to extend our core social groups to form groups of groups, and groups that transcend generations and geographical boundaries—bridged by language, music, and other cultural artifacts.

The power of language is that it allows us to learn things we haven't seen directly for ourselves. The power of a social network is that it enables us to exponentially expand upon what we can know, remember, and learn, and it allows us to engage in what the philosopher Mark Johnson calls distributed cognition (Johnson 2007). Human knowledge resides in the entire group rather than in any one single person. A feral child is unlikely to develop language or mathematics in isolation—that knowledge is in the network. Not only do we possess modular, recombinant, and cognitively fluid brains, but we also seem driven to extend our recombinant thinking beyond the boundaries of a single brain.

Language and music gave us the ability to tell stories, but not just about the things that really happened. I'm guessing that the prehistoric computer musicians probably modified and enhanced their stories a little bit—practicing a prehistoric form of digital signal processing. And that some of them even told stories about events that never happened—a kind of prehistoric digital synthesis.

Looking Back on the 20th Century

Viewed in that light, the invention of software in the mid 20th century was as big a breakthrough as the mastery over fire was for our prehistoric ancestors. Language gave humans the ability to tell stories about something that does not yet exist. Software gave us the ability to make that imaginary thing exist—for real—in the actual physical world.

The first time I witnessed a sequence of symbols being transformed into an actual sound pressure wave that I could hear, I felt like I was witnessing a miracle. I thought it was a completely deterministic and explainable miracle but a miracle nonetheless. By simply manipulating symbols, software can effect change in the physical world: It can make a mechanical arm move, it can change the colors of light arrays, it can print images, and it can create three-dimensional objects. It can even write more software and design new circuits on which to run itself.

Software as Hardware with Cognitive Fluidity

When I say software, I mean hardware. Because the invention of software was actually the invention of a particular kind of hardware: a machine with cognitive fluidity.

We've been building machines for a long time. Typically, a machine is built to solve a particular problem or serve a fixed function. In that sense, a machine reflects a known problem, a problem we've encountered before, at some time in the past.

But when you combine a modular machine with software, it becomes a virtual machine. It may still have modules with specific functions but, using the software, you can combine those functions in new ways to solve many different problems—even some problems that have never been encountered before.

What software did was to make it possible for a single machine to take on the behavior of many different machines, which drastically lowered the physical cost of trying out a new idea and opened the floodgates of experimentation and creativity.

Paul Doornbusch places the birth of "soft-hardware" in 1943, when a programmable, vacuum-tube calculator called Colossus was developed by Alan Turing during World War II to help people like Joan Clark, Jean Valentine, Ruth Biggs, and others break the Enigma code (Doornbusch 2009).

In a way, it's no surprise that it was only six years later that someone first used a computer to make sound. We are, as Oliver Sacks put it, "an essentially, profoundly musical species" (see http://youtu.be/HqKry0gh_NA). In 1949, the CSIRAC (Council for Scientific and Industrial Research Automatic Computer) was the fourth stored-program computer in the world. It didn't have a display, meaning there was no easy way to know what a program was doing or when it had finished executing, so they sent the output of the serial bus directly to a speaker and developed a special loop they called a "blurt" to indicate the end of a program. This may be the first example of using a computer for data sonification . . . in 1949! And by 1951 a programmer named Geoff Hill had figured out how to write programs with loops to generate sustained square waves at specific pitches and play popular tunes.

One year later, the ILLIAC I (Illinois Automatic Computer)—the vacuum-tube-based computer that Lejaren Hiller used for his algorithmic music composition, the *Illiac Suite*—was built at the University of Illinois.

PLATO Becomes a Social Network in Spite of Itself

In 1960, Don Bitzer had the idea to use the ILLIAC I to deliver online courses and exams. Over the next ten years, the PLATO (Programmed Logic for Automatic Teaching Operations) system, as he called it, grew into a worldwide network of over a thousand flat-panel plasma display terminals. In 1967, PLATO became more cognitively fluid when they added a programming language called TUTOR. For the first time, PLATO users could program the network to do whatever they wanted it to do—which apparently was to connect with each other, since most of the programs developed over the next five to ten years were for music synthesis, e-mail, news groups, chat rooms, instant messaging, and inter-terminal games.

There was a PLATO classroom on the ground floor of the CERL (Computer-based Education Research Laboratory) building called the Zoo. There were always a few people in the classroom working on required coursework, but every night just before 10:00 PM the room would start to fill up. At precisely 10:00 PM, the systems programmers would lift the restriction on games and the room would be filled with gamers until 4:00 or 5:00 AM, when the system went down for backups.

Witnessing this made me wonder whether people are driven to use software—just as humans had used language—to form ever-more-complex interactive networks of people and knowledge, almost as if we were genetically programmed to use every new technology for purposes of connecting with each other and extending the range of our distributed cognition.

A Nostalgic Story about Hardware Gaining Cognitive Fluidity

Up on the fifth floor of the CERL building that housed PLATO there was an old radar research lab—actually, it was just a temporary floor that had been added during World War II that you had to access using metal fire escape stairs. The fifth floor was where a group of electrical and computer engineering

students were building digital synthesizers to work with PLATO (Scaletti 1985).

I had first stumbled on the CERL Sound Group at an Engineering Open House right around the time I was finishing my music composition degree. This was also around the time that I had started to realize that, to do the things I really wanted to do in computer music, I was going to have to learn how to make the tools myself. So when Lippold Haken offered me a research assistantship at CERL, I jumped at the chance to go back to school and get a degree in computer science.

What I wanted at that time was to be able to work with sound directly and interactively, and to be able to build new kinds of structures based on sound, not based on the notes of music notation and not limited to a model of instruments playing scores. I loved voltage-controlled synthesizers, so I wanted to be able to use outputs of some modules to control or modulate the parameters of other modules. I also loved cutting and splicing tape, so I knew I would have to include ways to work with audio recordings. I was a harpist so I wanted to be able to create environments in which I could perform live, generating sound and processing a live signal from my harp in real time. And I had just come to the same conclusion that many of you have—which is that if you can't find what you need, then you just have to learn to build it yourself.

Sound Hardware Becomes Softer

The history of the CERL Sound Group is practically a case study in hardware gaining cognitive fluidity. In 1974, Sherwin Gooch had designed a digital synthesizer for PLATO with four fixed-waveform, hardware oscillators, and CERL made dozens of them to put into PLATO classrooms for teaching music theory.

In 1978 Sherwin built a new and improved version: the GCS (Gooch cybernetic synthesizer) had 16 oscillators and performed additive synthesis. It was a more powerful machine and implemented more algorithms, but it was still a digital circuit designed to do specific, fixed tasks like oscillators, envelope generators, and mixers.

In 1981, Lippold Haken designed the IMS (interactive music system) as a replacement for the GCS. Rather than designing a circuit with hard-wired oscillators, envelope generators, and mixers, he designed a more-general circuit based on two multiply accumulators and memory, and he used software—in this case machine-level microcode—to emulate the oscillator, envelope generator, and mixer circuits of the GCS. In other words, from the outside, the IMS behaved just like the old GCS. While he was working on the microcode, Lippold realized that he could extend the functionality to include amplitude modulation, frequency modulation, and waveshaping, and because it was software, he was able to expand the functionality without modifying the hardware.

At this point, the software was a virtual machine: Lippold had used software to make generic hardware behave like specific hardware. From the outside, from the user's point of view, though, the architecture of the virtual machine was every bit as fixed and unchangeable as a hardware implementation would have been.

When Kurt Hebel saw the way I had been working—running Music 360 batch jobs on the campus mainframe and making once-a-week appointments to use the shared digital-to-analog converter—he shook his head and said, "You could do this in real time—all we'd have to do is make the IMS microcode programmable by the user!"

So in 1984 Kurt and Lippold started redesigning the IMS to have a user-definable microcode, making it so the architecture of the "simulated synthesizer" could be changed simply by loading a new microcode. That was the Platypus. And Lippold assigned me the task of writing software for Platypus, which was, of course, exactly what I wanted to do. (Thank you, Lippold!)

Figure 1 is a photo of Kurt at the 1989 ICMC in Ohio a few years later with the wire-wrapped Platypus on top of the table and the SCSS (sound conversion/storage system), the disk system that he designed for the School of Music, underneath the table.

The first piece I realized with the Platypus was *sunSurgeAutomata* (1986), which was based on self-organizing one-dimensional cellular automata,

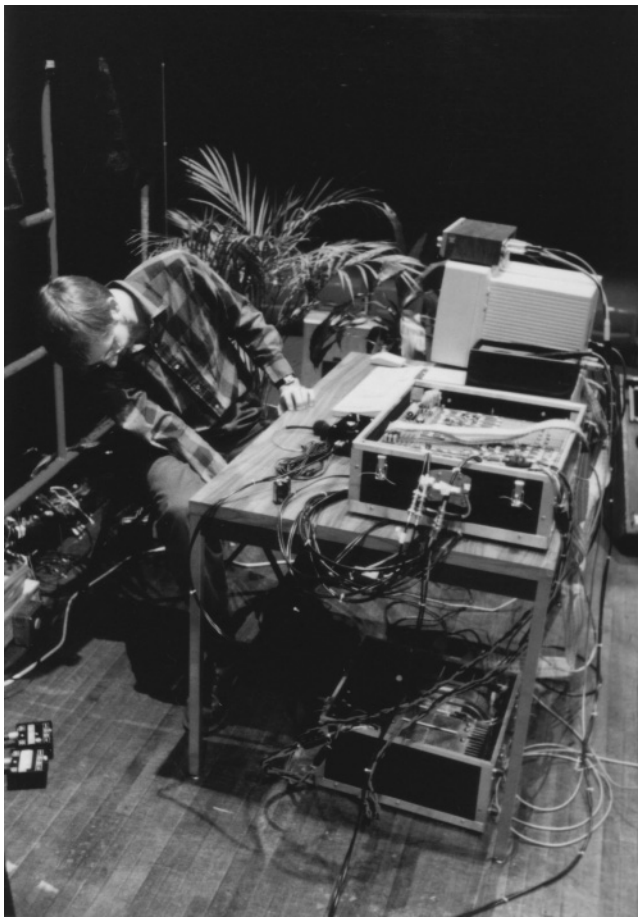


Figure 1. Kurt J. Hebel at the 1989 International Computer Music Conference in Ohio with the Platypus on top of the table and the SCSS (sound conversion/storage system), a disk system he had designed for the University of Illinois School of Music, underneath the table.

clicks, and a quote from Lewis Thomas about how energy surging from the sun fuels the improbable ordered dance we call life on Earth.

To make the piece, I wrote a microcode implementation of a one-dimensional cellular automaton as a rhythmic pattern of clicks using Stephen Wolfram's rule 90; another microcode where I used the cellular automaton as a pattern of gates on a recording of my voice speaking the Lewis Thomas text; and yet another microcode where I tried to apply the cellular automaton rules to a stream of samples by taking an input stream of samples and forming each output sample as a function of the previous and next samples in a buffer (which ended up being pretty close to a Karplus-Strong type resonator).

So, I was happy because there were no instruments and no notes—instead the structure was self-organizing patterns of clicks and algorithmic signal processing.

Each program was a custom-written microcode, meaning that it was a sequence of very long instruction words for the Platypus. I loaded one of the microcodes into the Platypus at a time and recorded the output of the digital-to-analog converter through the analog-to-digital converter and onto Kurt's SCSS disk. At the 1987 ICMC, Barry Truax asked me why the piece was only five and a half minutes long, and the answer was—that's how much time would fit on the disk.

So with *sunSurgeAutomata* I finally I had access to the virtually infinite flexibility of software synthesis and could hear the results in real time. Instead of using a specific machine, I could create any virtual machine (though only one at a time).

Sound Software Becomes More Modular and Recombinant

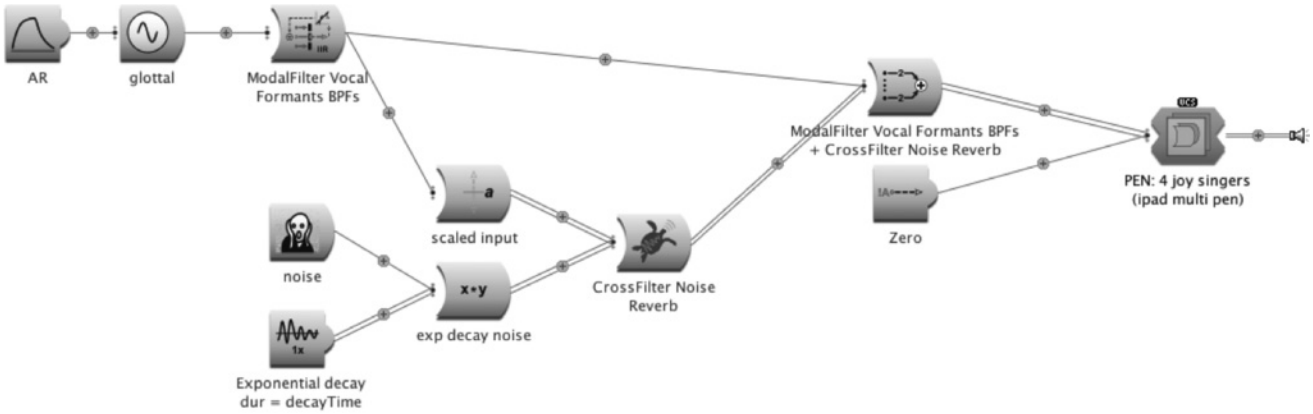
Joel Chadabe likes to talk about the difference between "control" and "power." He describes "control" as being like when you're in a startup and you do everything, from the low-level to high-level jobs; and "power" as being like a big-shot CEO where you "articulate a vision" and delegate all the low-level details to someone else, or in this case something else.

At this point, I had a very high degree of control, because I could write whatever sequence of machine language instructions I wanted. But I didn't have a powerful way to leverage that control. There wasn't a mechanism for reusing the microcodes or for combining them in new ways. It was time for the system to evolve some modularity and recombinance. And that's when I found Smalltalk.

Smalltalk was invented at Xerox PARC, according to Dan Ingalls, for the purpose of providing "computer support for the creative spirit in everyone." In practical terms, what that really meant was that they were trying to make complex software systems more manageable. One way they did that was to make the code more modular and recombinant.

As you probably already know, Smalltalk is based on modules called "objects" whose internal states

Figure 2. Kyma Sounds are both functional and modular, making it possible to incrementally build or algorithmically generate complex Sound structures and save them for future reuse.



are accessible only through that object's defined protocol, thus greatly reducing the dependencies between objects. That makes it possible to replace an object with a new object without breaking the rest of the system. So you can incrementally improve or optimize parts of the system while it's running! In that way, a Smalltalk image is almost like a living system that you can continually extend and modify.

The syntax of Smalltalk is very simple; what makes it interesting is that it is like a large database of code modules that you can recombine, modify, and add to, and that the things you add and the changes you make become part of the language.

So Smalltalk turned out to be the ideal environment for adding modularity and recombinance to the Platypus microcodes. In Kyma, as in Smalltalk, there's a uniform metaphor: Everything is a Sound (see Figure 2). On the Platypus, a Sound became a data structure with a pointer to the microcode that could generate the next sample in its sample stream (Scaletti 1987).

Because everything is a Sound and the state of a Sound's parameters is internal to that Sound, you can replace any Sound with any other, no matter how simple or complex the Sounds may be. In other words, Sounds are both functional and modular, making it possible to incrementally build complex Sound structures and save them in a library for future use. This also makes it possible to generate Sound structures algorithmically, as in the Kyma 7 Gallery. Kyma ends up being a little like Smalltalk in the sense that it is a database of code modules (Sounds) that one can recombine in new ways.

The extreme late binding in Smalltalk inspired some other early features in Kyma: For example, "lifted" or "abstracted" Sounds could have variables in their parameter fields, and you could use Smalltalk scripts to create concrete instances of the lifted templates and bind concrete values to the variables. The late binding also inspired another kind of variable in Kyma called an EventValue, which serves as an abstract placeholder for a live controller and automatically generates its own widget in a Virtual Control Surface that you can optionally remap to external control sources. For example, see the parameter fields in Figure 3.

Kyma also included some special, temporal modules called TimeOffset and SetDuration, so you could use Kyma to create a schedule of when to load and start Sound objects and specify how long they should run—meaning that I finally had a virtual machine that could change over time.

Although the temporal objects were part of Kyma from the beginning, it wasn't until many years later, when I added the graphical Timeline interface (see Figure 4), that it became lot clearer how to use and manipulate the temporal objects; it was only then that people really started utilizing them extensively (Scaletti 2002).

In 2015, with Kyma 7, I tried to get a little closer to my longtime dream of a sound machine with cognitive fluidity. In a Multigrid (see Figure 5) you can create new virtual machines at arbitrary

Figure 3. The extreme late binding of Smalltalk inspired the inclusion of variables (names preceded by question marks) and EventValues (names preceded by exclamation points) as abstract placeholders in Kyma parameter fields that can be bound by enclosing environments.

times by activating modules with mouse clicks, MIDI program changes, OSC (Open Sound Control) messages, or algorithmically using Capytalk, the event language that runs on the Pacarana (Scaletti 2015).

Origins of Symbolic Sound

Around 1988–1989, a political coup inside the university succeeded in pushing Don Bitzer out and shutting down the Computer Based Education Laboratory. All 200 or so of us were given a year to find another job—which was an amazingly generous amount of time and what made it possible for me to start Symbolic Sound with Kurt Hebel, even if it was on a research assistantship salary.

On the morning of 6 June 1989, Kurt defended his dissertation, and his idea of celebrating was to go straight to his office, sit down at a big drawing table, and start designing the Capybara. At that time, the only commercially available DSP chips ran at half the speed of the Platypus, so if we wanted to match or exceed the power of the Platypus, our only choice was to use more than one of them. Kurt ended up using, not two, but eight DSPs, each one on a plug-in card with its own memory, so it was modular and scalable. Figure 6 shows the inside of a Capybara circa 1990.

When we got our very first printed circuit boards back, we discovered that the board manufacturer had made a mistake on one of the thermals and they had connected the power and ground planes. So all our boards were useless and all our money was gone. In desperation, I suggested we try drilling through the connection using a home power drill. Amazingly enough, it worked! Figure 7 is a photo of Kurt drilling through the boards in our assembly room, which was actually the kitchen of our fourth-floor student apartment.

(Did you notice how I just followed the formula for a nostalgic story: A seemingly insurmountable challenge, the support of close allies, and the triumph over adversity? I'm pretty sure you all have stories like this!)

Luckily, the functional nature and modularity of Kyma Sounds meant that they could be computed on independent processors. There were a lot of ways the processors could have been connected to each other, but to fit the DAG (directed acyclic graph) structure, Kurt connected them in a line, so the outputs of the Sounds on one processor could feed into the inputs of the Sounds on the next. It was a coevolution of both hardware and software.

Six hardware generations later, with the Pacarana, the multiprocessor architecture idea could be extended outside the box, by connecting multiple Pacas (or Pacaranas) to each other (see Figure 8).

This points to another trend in the late 20th and early 21st centuries: a plateau in processor performance. Although from 1986 to 2002, processor performance was increasing at an average of 52 percent per year, ever since 2003, clock speeds have hit a virtual plateau (Bailey 2015), as you can see in a graph published by Intel Corporation at www.gotw.ca/images/CPU.png. So, at least for now, the way forward seems to be multiple, multicore processors, and functional, modular computing fits well within that architecture. That was just one, nostalgic anecdote about the coevolution of

Figure 4. The graphical Timeline interface made it clear how to utilize and manipulate the temporal objects in Kyma.

Figure 5. In a Kyma 7 Multigrid with submixes, signal flow graphs can be re-routed and modified without interruption to the audio signal.

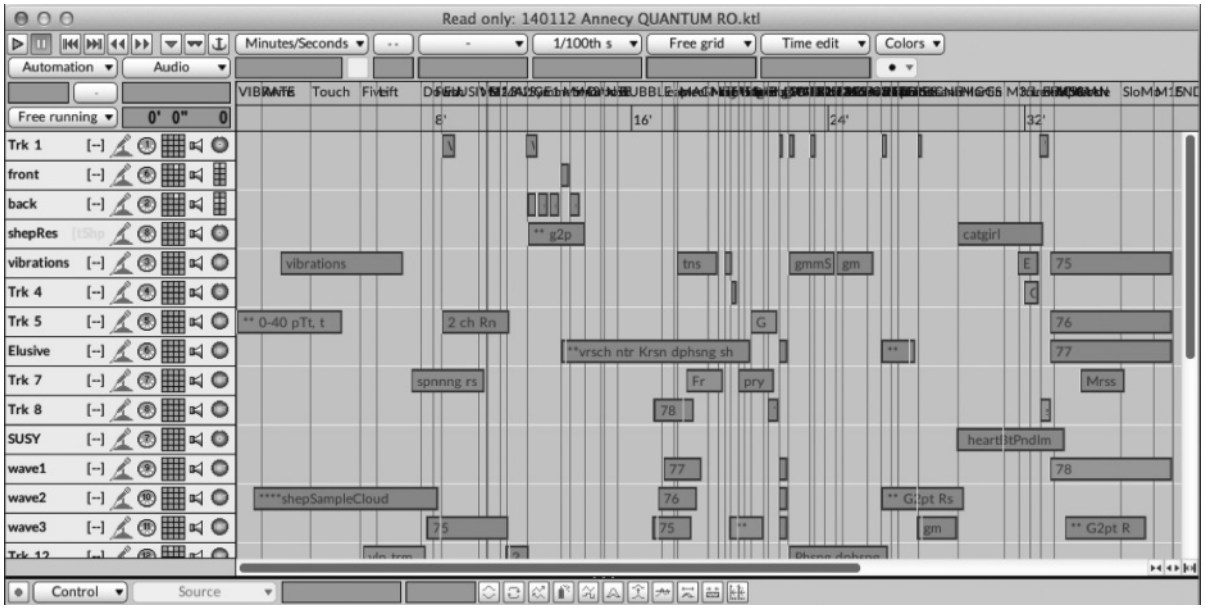


Figure 4

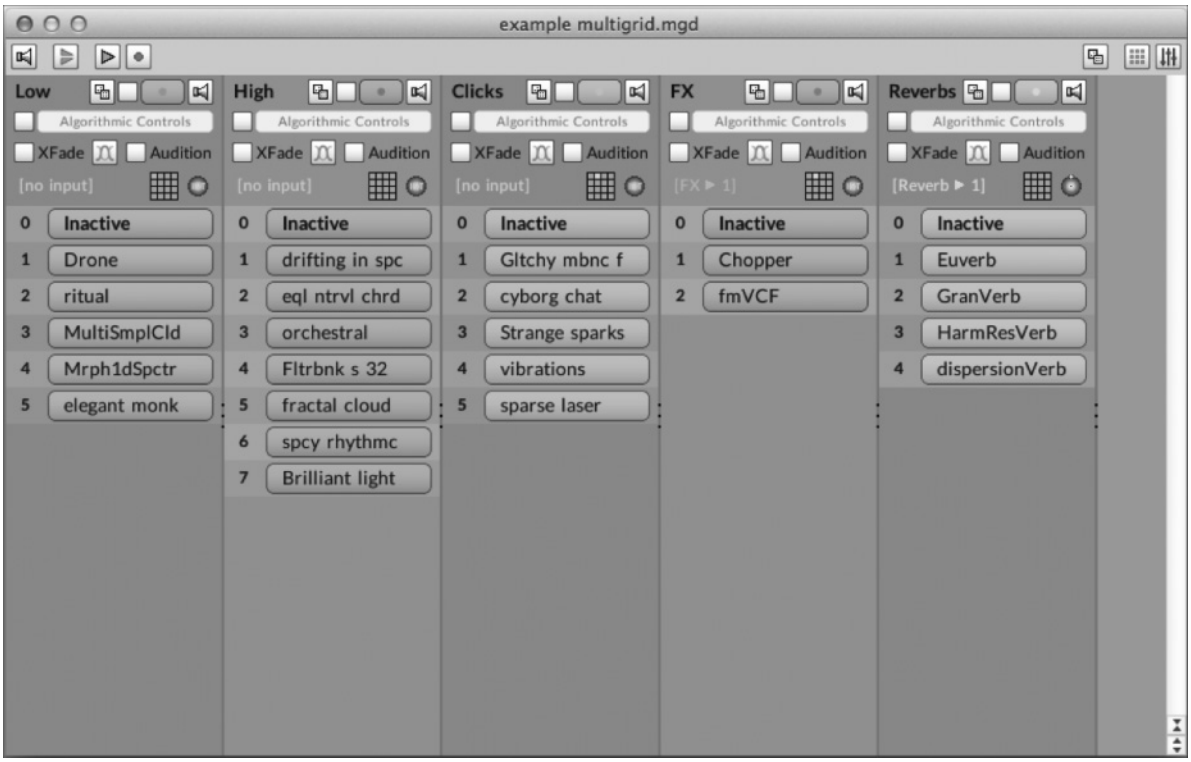


Figure 5

Figure 6. Kurt Hebel's 1989 Capybara design had eight digital signal processors, each on its own card with its own memory, so it was modular and scalable.

Figure 7. Symbolic Sound's first assembly facility was the kitchen of the founders' fourth-floor student apartment.

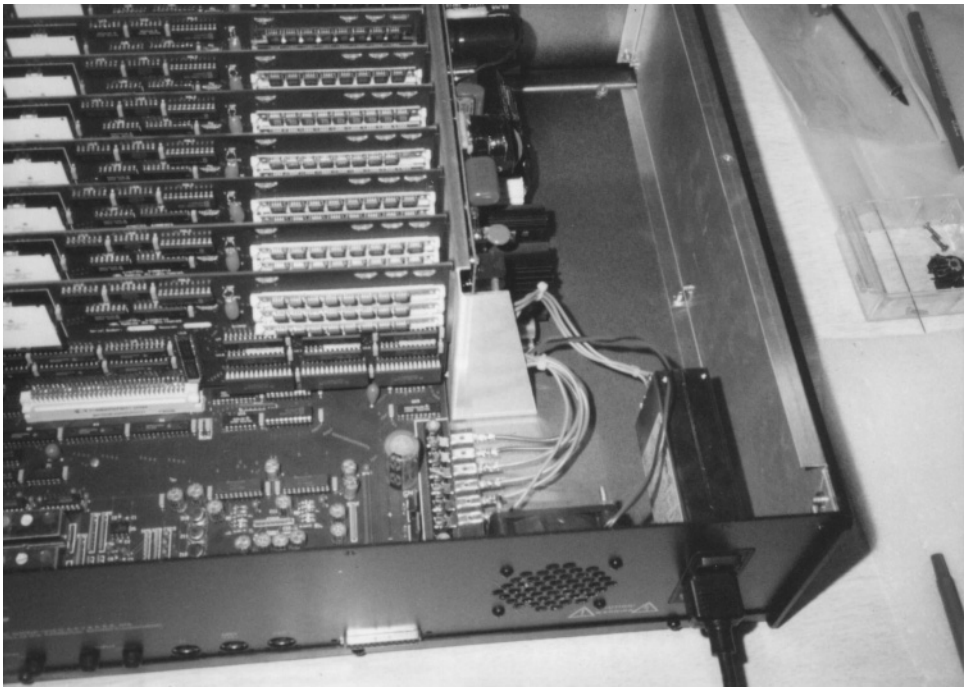


Figure 6

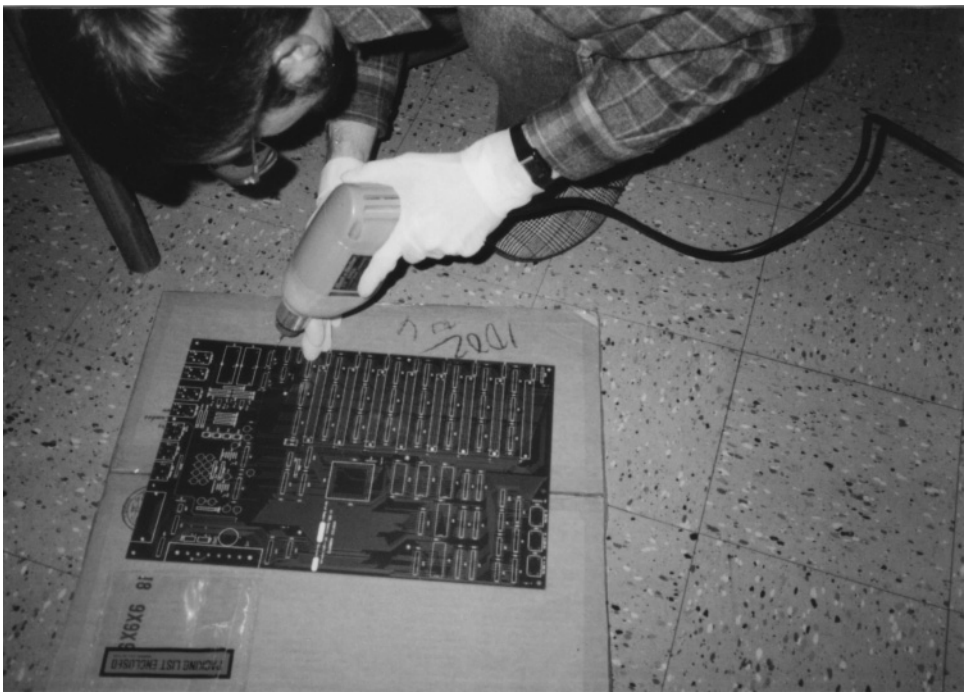


Figure 7

Figure 8. With the advent of the Pacarana, the multiprocessor architecture could be extended outside the box. Multiple Pacaranas can be connected to each other and appear, to the Kyma software scheduler, as a single multiprocessor device. (Photograph by Tobias Enhus.)

computer music hardware and software. This kind of coevolution must have been going on everywhere back then, and it continues today. Alan Kay said that "people who are really serious about software should make their own hardware." Because, when you come right down to it, software is just a special kind of hardware—and the soft and hard parts can evolve together as a system. That's probably why Apple designs their own hardware, and why they make their own version of the ARM processor: so they can optimize it to make their iOS perform well on the iPhone and iPad.

A Nostalgic Story about Distributed Cognition

I remember one day in 1993, when my friend Dan Brady from the National Center for Supercomputing Applications said, "Hey, you should come over and see this thing that Marc Andreessen is working on. It's a kind of multimedia hypertext way of accessing that World Wide Web thing they have at CERN." It turned out that Dan was describing Mosaic—the first Web browser. Looking back on that now, it feels like having been present at Kitty Hawk.

But in a way, it wasn't all that surprising after what I had seen happen with the PLATO network. The Web just looked like yet another case of humans using every new technology for the purposes of connecting to one another. And it started me thinking about how I might be able to express that idea in a new piece.

I ended up using Mosaic in a Web-based installation called Public Organ at the 1995 ICMC in Banff. Public Organ was based on several quotes from Lewis Thomas about how humans seem driven to use language and technology to form networks and engage in distributed cognition. Among other things, you could add your own image to Public Organ, either by sitting down on a piano bench and triggering the camera or by connecting to the installation remotely using CU-SeeMe (for a history of the CU-SeeMe software, see Packetizer 2015).

The real reason I brought this up, though, is that I came across the QuickTime file containing all the images captured during that ICMC 20 years ago. So I thought it might be fun for us to practice a little group nostalgia. See if you recognize anyone . . . [see supplementary materials available online at

mitpressjournals.org/doi/suppl/10.1162/COMJ_a_00341].

Current Challenges

Looking back at the 20th century, it seems that the invention of software and our drive to construct ever-expanding networks of distributed cognition have resulted in an explosion of experimentation and creativity.

It's now easier to experiment and to test new ideas quickly and inexpensively, which means there are more ideas and more products, and they can be more widely distributed more quickly. Which also presents a challenge: The speed of development is now so fast that it's rare to have a chance to get deeply into anything before something new is being dangled in front of us. It requires an almost monk-like ascetic focus to stick with any tool or idea or project long enough to achieve deep mastery.

And the corollary is that the current model rewards projects that have short development times, because one could potentially make a lot of money on a single app if you can sell it to millions of people. However, you only have about two weeks to make most of those sales before the news of your new app recedes back into the noise. In order to sell something to that many people that quickly, the temptation is to make an app that's easy to understand in a short amount of time. One way to do that is to write hardwired, non-recombinant software that looks and acts like something familiar.

I hope everyone here has and will continue to write hugely successful apps! And I hope that we also manage to set aside time for our larger, longer-term projects—projects like large-scale frameworks that provide recombinance and fluidity.

The same model has succeeded in pushing the monetary value of recorded music down to virtually zero. So another challenge is to find alternative models for presenting our sound art to the world. This probably entails inventing new kinds of experiences that can't be captured in an audio recording, whether that has to do with interaction, with spatialization, with learning experiences, with live unrepeatable performances, or perhaps with incorporating what humans seem to love most, which is interacting with each other.

News versus New

Finally, it's useful to keep in mind what Alan Kay calls the difference between something that's "news" and something that's "new" (see youtu.be/FvmTSpJU-Xc). News is always incremental. News assumes you have a pretty good idea of the current state of the world, and it gives you a daily update. As such, news items have to be short and based on familiar information that can be quickly understood.

Something truly new, on the other hand, is not quickly or easily understood. It may require that you learn a body of new information or new ways of thinking, which can be time-consuming and sometimes even painful.

It's tempting to do projects that could get us into the news—projects that are easy for the public to understand because they are already familiar with the ideas. And if you do something that is truly new, it may take time for others to understand what you're saying. And they may not have the time, because they're trying to keep up with the news instead of taking the time to understand something new. So learning how to congratulate yourself can be a pretty useful skill, because it might be a long time before anyone else understands what you're doing.

I still remember how, at the 1987 ICMC, Iannis Xenakis played an excerpt of his piece *Bohor* during his keynote, and at the end of the piece, he let the tape keep running so you could hear the audience booing. It surprised me so much that it's the only thing I remember about his talk. In retrospect, I think he took pride in the fact that his ideas were new enough that not everyone in the audience understood.

Looking Forward

So finally, looking forward . . .

By the way, why is it that so many of our words for remembering and predicting are visual metaphors?

Why are we "looking" instead of "listening," and why is it "forward"? We have words like "envision," "envisage," "imagine," "picture," "visualize," "foresee," and "reflect" to mean anticipating or remembering something in your mind, but when I did a search for "hearing sound in your mind," the first things to come up were terms like "tinnitus," "auditory hallucinations," "hearing voices," "schizophrenia," and "exploding head syndrome."

I wonder why we call someone who can foresee the future a "visionary," but we think someone who hears the future suffers from some kind of pathology.

Instead of articulating a "vision" for the future in this last section, maybe we should invent a new term and call ourselves "auditionaries." Because, to coin a phrase, computer music practitioners predict the future by composing it and by coding it!

Softer Hardware

One of the biggest benefits of "soft-hardware" has been that it has allowed us to increase our failure rates. It's fairly easy and inexpensive to try out ideas in software, because if they don't work, you can try again—like the way the adaptive immune system tries out millions of combinations to find one that fits the new pathogen.

But right now, if you want to design your own processor to fit your software, it's still really expensive, even if you use an FPGA (field-programmable gate array). If some of the barriers to hardware fabrication can be reduced, we'll be able to experiment with alternative hardware ideas almost as quickly and inexpensively as we experiment with software right now, further blurring the boundaries between hardware and software development.

For example, researchers at PARC (the Palo Alto Research Center) are designing a laser printer to print microprocessors, memory, and MEMs (microelectromechanical systems) through a process Eugene Chow calls "xerographic microassembly," where the "ink" is made by breaking silicon wafers into thousands of "chiplets" and "printing" them, much as a laser printer prints toner onto paper. An array of electrodes generates electric fields that control the placement and orientation of the circuits [*New York Times*, 9 April 2013, "Tiny Chiplets"].

Circuit printing is starting to encourage more experimentation and has already changed the way we think about hardware. John A. Rogers and Yongang Huang at Northwestern University are developing techniques for printing a circuit onto a thin, rubbery, water-soluble substrate so you can apply it to your skin using water (Yeo et al. 2013). Sensor arrays, LEDs, transistors, radio frequency capacitors, wireless antennas, conductive coiled wires, and solar cells for power can all be attached directly to the skin, and the circuits are fabricated as squiggly wires so they can bend, twist, and stretch with your skin and still continue to function. It's like taking wearable computing to the next level.

Brain Music

The next step might be to go even deeper—beneath the skin. After all, our sensory organs themselves are just interfaces. In the auditory system, for example, there is a mapping of air pressure variation to physical movement of bones in the ear, to vibration of the basilar membrane, to electrical patterns of neuron firing, and so on up the eighth nerve to temporal electrochemical patterns in the auditory cortex.

If that's true, could we then just skip over the ear and create an interface that maps a numerical model of sound directly to electrical stimulation of the nerves leading to the brain or stimulation of the brain itself?

We're already sort of doing that with cochlear implants. A cochlear implant takes a signal from a microphone, splits it into 22 frequency bands, and feeds the amplitude of each band to an electrode that has been surgically implanted in the cochlea, kind of like a 22-band vocoder. Which is both miraculous and yet at the same time still pretty primitive, considering that they're trying to approximate the behavior of 16,000 hair cells with only 22 electrodes.

As a next step, why not skip the microphone, synthesize the audio input from a model, and route the synthetic signal directly to the brain? We'll have to develop a deeper understanding of how sound

actually maps to activity in the central nervous system, and we'll have to develop entirely new forms of composing—new ways to synthesize and structure a flow of experience for our audiences. So it's going to be really interesting; I can hardly wait!

Distributed Cognition: The Wired Version

So that was input *to* the brain. There are also interfaces that go in the opposite direction—interfaces that monitor signals emanating from the brain, enabling you to do things like control prosthetic hands, and of course, perform music and play computer games.

If we have brain input interfaces and brain output interfaces, then how long until we have a brain-to-brain interface? Apparently not very long, because in 2013, neurobiologists at Duke along with colleagues in Brazil and China published a paper in *Scientific Reports* entitled "A Brain-to-Brain Interface for Real-Time Sharing of Sensorimotor Information." The brain-to-brain interface (or BTBI, as they call it) enables real-time transfer of sensorimotor information between the brains of two rats. An "encoder" rat was given a task with either tactile or visual stimuli and two choices. While the encoder rat performed the task, its cortical activity was sampled and transmitted to the corresponding area of a "decoder" rat's brain. The decoder rat made correct decisions guided solely by the information provided from the encoder rat's brain.

The researchers suggest that BTBIs could enable networks of animal brains to exchange, process, and store information, serving as the basis for what they call "novel types of social interaction" and "biological computing devices" (Pais-Vieira et al. 2013). You can view a video of one of the experiments in the supplementary information included with their paper: www.nature.com/article-assets/npg/srep/2013/130228/srep01319/extref/srep01319-s1.mov.

Hearing the Future

Whatever the specifics of the technology, what I hear for the future is what I hear us doing right now, and what it seems we have always been driven to do: to experiment, to extend our distributed cognition into new areas, and to dare to try to make something truly new—even if it sometimes means falling down seven times and getting up eight.

Computer musicians are uniquely positioned for this kind of experimentation, because we can combine the imagination and daring of artists with the technology that can make the imaginary real.

Postscript

Now that he knows how beneficial nostalgia can be, Constantine Sedikides says that he intentionally files away memories in anticipation of being able to reflect on them in the future—now he's actually looking forward to looking back.

I hope you'll go out now and intentionally create a whole bunch of meaningful memories for your future self to be able to look back on with fondness and inspiration and optimism. Thanks!

Acknowledgments

Thanks to ICMC 2015 conference organizers Jon Nelson, Panayiotis Kokoras, and Richard Dudas for inviting me to reflect on the past, present, and future of computer music. Thanks to Doug Keislar for encouraging me to turn my keynote lecture into an essay to help commemorate the 40th anniversary of *Computer Music Journal*. And special thanks to Kurt Hebel for patient listening and thoughtful comments as I trimmed what was originally a three-hour talk down to the requisite 50 minutes!

References

Bailey, M. 2015. "Parallel Programming: Moore's Law and Multicore." Available online at web.engr.oregonstate.edu/~mjb/cs475/Handouts/moores.law.and.multicore.2pp.pdf. Accessed 7 October 2015.

Doornbusch, P. 2009. "A Chronology of Electronic and Computer Music and Related Events 1906–2013." Available online at www.doornbusch.net/chronology. Accessed 6 October 2015.

Dunbar, R. 2003. "The Social Brain: Mind, Language and Society in Evolutionary Perspective." *Annual Review of Anthropology* 32:163–181.
Johnson, M. 2007. *The Meaning of the Body: Aesthetics of Human Understanding*. Chicago: University of Chicago Press.
Kashton, N., and U. Alon. 2005. "Spontaneous Evolution of Modularity and Network Motifs." *Proceedings of the National Academy of Sciences, U.S.A.* 102(39):13773–13778.
Mithen, S. 2006. *The Singing Neanderthals: The Origins of Music, Language, Mind, and Body*. Cambridge, Massachusetts: Harvard University Press.
Packetizer, Inc. 2015. "The CU-SeeMe Project." Available online at www.packetizer.com/ipmc/history-of-videoconferencing/project.html. Accessed 16 October 2015.
Pais-Vieira, M., et al. 2013. "A Brain-to-Brain Interface for Real-Time Sharing of Sensorimotor Information." *Scientific Reports* 3:1319.
Scaletti, C. 1985. "The CERL Music Project at the University of Illinois." *Computer Music Journal* 9(1):49–58.
Scaletti, C. 1987. "Kyma: An Object-Oriented Language for Music Composition." In *Proceedings of the International Computer Music Conference*, pp. 49–56.
Scaletti, C. 2002. "Computer Music Languages, Kyma, and the Future." *Computer Music Journal* 26(4):69–82.
Scaletti, C. 2015. "What's New in Kyma 7?" Available online at kyma.symbolicsound.com/whats-new-in-kyma-7. Accessed 7 October 2015.
Silverstein, A. M. 2009. "Immune System: Promethean Evolution." *Science* 325(5939):393.
Wildschut, T., et al. 2006. "Nostalgia: Content, Triggers, Functions." *Journal of Personality and Social Psychology* 91(5):975–993.
Wrangham, R. 2009. *Catching Fire: How Cooking Made Us Human*. New York: Basic Books.
Yeo, W. H., et al. 2013. "Multifunctional Epidermal Electronics Printed Directly Onto the Skin." *Advanced Materials* 25:2773–2778.