Assembly And Differentiation

Seth Copen Goldstein
seth@cs.cmu.edu

There are two key challenges to creating the next generation applications: assembly and differentiation. Robust, inexpensive assembly is required at the very lowest levels (assembling nanoscale low-power economical computing devices) and the highest levels (assembling distributed applications from reusable low-cost software components). Differentiation is less obvious, but in some sense a more basic primitive that will enable low-cost assembly. By differentiation I mean the ability to create an aperiodic complex system from relatively homogeneous and possibly defective parts.

Robust inexpensive assembly is the key to accomplishing next generation applications. Assembling nanoscale wires and molecules into computers is the key to low-power low-cost high-density computers. Assembly of sensing and actuating components with computing elements is essential for the low-cost computation and communication needed to realize ubiquitous computing. Moving up the hierarchy is the challenge of assembling many processing elements into a unified distributed computing system. The means of organizing billion-processor networks into a coherent whole is a challenge containing many challenges. Reducing the cost and increasing the reliability of software is essentially a challenge of assembling components. Producing the high-fidelity sensor networks required for many ubiquitous computing tasks is a challenge in assembling information from many low-resolution sensors into a high-resolution picture. Computer science has produced, or at least understands, a powerful set of tools and stand-alone components. It now faces the task of assembling them together into coherent ensembles. Here, I briefly outline some of the assembly challenges facing computer science, from basic computing technology to a grand challenge application: smart matter.

At the base of any computer science application is the computing element itself. For the last twenty years the computing substrate of choice has been CMOS. The cost per compute element has dropped as manufacturers have learned to assemble ever more components on a single chip. With the advent of MEMS, this assembly process has begun to include sensor and actuator technology as well. The beauty of this process has been that components (e.g., wires and transistors) have been manufactured and assembled into a device in a "single" step. However, that "single" step is now more than 200 processing steps—each of which requires a fine degree of precision. The challenge is to find a new method of assembling components into devices; a method that requires less precision, a method that works with elements at the nanoscale, a method that is inexpensive, and yet robust.

Researchers have already demonstrated computing components that are smaller than ten nanometers. The goal is to assemble these components into a useful computing device. Many believe that we should be able to create inexpensive, low-power computing devices with densities of 10^{12} logic-gates/cm^2. Proposed manufacturing techniques range from top-down (e.g., photolithography) highly organized techniques to bottom-up random placement. The former require high-precision at manufacturing time which currently appears to be too expensive and too dependent on almost defect-free results. The latter requires time consuming discovery processes to bring order out of the random placement of devices. A middle road is to construct, bottom-up, simple grid-like structures; a wire pitch of 10nm will meet the density specified above. To make this technology viable requires a very inexpensive assembly technology—one that relies on self-

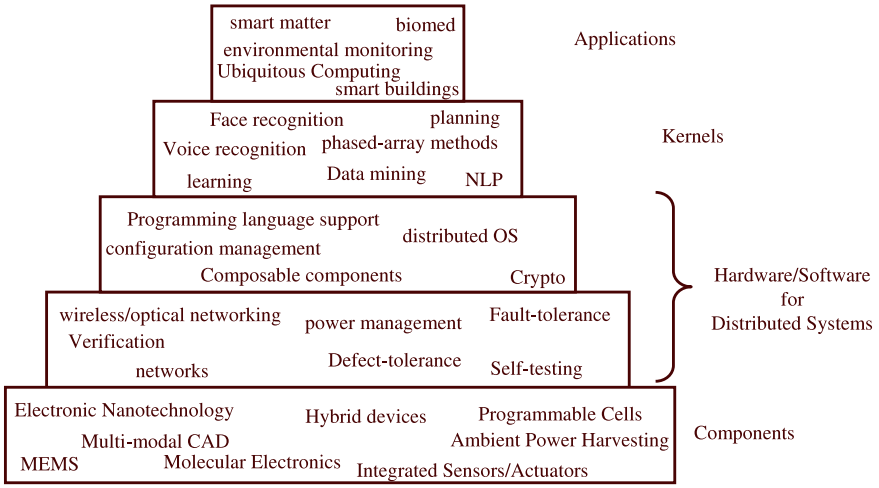

Figure 1: *Strategic Computing Pyramid similar to that of Figure 1 from the 1983 DARPA report Strategic Computing.*

assembly. A technology that is almost oblivious to the individual placement of a component. The manufactured device must be defect-tolerant and highly homogeneous; for example, it will be a structure like a field-programmable gate array (FPGA). Thus, the information necessary to make a computing device will be added after the manufacturing of the substrate. In other words, differentiation is required to create the desired functionality from a basically homogeneous crystal-like structure. There are many systems challenges inherent in such a manufacturing technology. These include defect-tolerant architectures, fault-tolerant and low-power circuit techniques, and, scalable CAD/compilation tools.

In Figure 1 I show a hierarchy of challenges/tasks in the systems area. At the base of this hierarchy are new computing technologies, e.g., molecular electronics. Many other base technologies must be developed to seamlessly integrate computing into our lives. For example, integrating sensors and computing devices is required to lower size, cost, and power while increasing reliability. Such systems will need to harvest power from the environment (from light, vibration, etc.) and then use it judiciously when computing and, most importantly, communicating. For example, to reduce cost and power requirements one might combine many low-resolution microphones into a high-resolution directional microphone through phased-array techniques. Another example is in programmable cells. In this technology computing devices are incorporated into biological cells, allowing the cells to express behaviors under our control. The usefulness of programmable cells is contingent on our ability to assemble complex and differentiated behavior from a large collection of cells, all with the same simple and uniform programs.

The need to create an aperiodic complex system from homogeneous and possibly defective parts is also present one layer up the hierarchy. Imagine that we can create inexpensive, dense, low-power computing devices. This will lead to computers embedded into the very fabric of our existence—in the paint on our walls, in the clothes we wear. These massively distributed sensor/computing networks will be composed of trillions of computing devices. The most likely way to inexpensively build such a network will be to construct it essentially at random. In other words, there will be several basic computing devices mixed together and then spread out. These components will have to self-assemble into a network of cooperating sensing and processing elements.

The challenge here lies in evoking organized behavior from a large collection of self-assembled processors. For example, how do we obtain global power management when each processor is only sensing its local conditions? How do we obtain low-latency high-bandwidth communication from a collection of possibly faulty routing nodes that were placed at random? How do we ensure that the system is working as a whole when we can only probe some of its components? These questions are unified by asking how we can create a system where each part may have to perform a different task when we can only afford to give each component the same program? This problem is similar to the biological problem of creating an organism that consists of many different kinds of cells all from a single cell.

If history holds any lessons for us, then no matter how difficult the above challenges appear, they will pale in comparison to the difficulty of developing inexpensive robust software. Software development, even for uniprocessors, is still more a matter of art than science. We need to develop the technology that allows software components to be easily assembled into working systems—working systems which themselves are composed of millions of components. An essential part of this will be the development of composable components. A composable component is a software module that reliably performs its specified task—independent of invocation order, parallelism, and other components in the system. Such components must have well specified interfaces. Interfaces that are flexible enough to allow a programmer to easily reason about how the component will act, but will not prevent a compiler from optimizing it for a particular use. Orchestrating single programs across massively distributed systems and then multiple programs on the system will require new approaches to operating system design. Systems such as J2EE and ".net" are much too fragile and heavyweight to support massively distributed systems with plug-n-play capabilities, especially for low-power sensor networks. Finally, managing systems with trillions of processing elements requires new approaches to configuration management and fault-tolerance.

Above the layer of the operating system and programming language support are the kernels needed for intelligent applications, particularly ubiquitous computing. Kernels such as data mining, planning, natural language processing, etc. will assemble useful coherent information from many sources.

Ubiquitous computing projects investigate some of the possibilities that arise when computing, sensing, and communicating are essentially free. What if we extend this to allow actuation that is also almost free? The result is *smart matter*. Smart matter is a collection of potentially homogeneous components that can come together in engineered ways. For example, imagine a basic component that is a cube, 25microns on a side. Each cube contains some processing along with a set of sensors, power, and communication. Furthermore, the cubes can "glue" themselves to other cubes under program control. This glue may be in the form of electric or magnetic fields or more permanent (programmable) covalent bonds. The cubes are small, inexpensive, and light enough that billions of them can be floating around in a small container ($< 3\text{m}^3$). A designer can broadcast a design to the ensemble and diffusion will be enough for the cubes to coalesce into a product with a significant amount of integrated computation. This is loosely analogous to the differentiation of a zygote, in that, the zygote grows in a polar medium which allows cells to express different genes depending on where they are geographically.

Meeting the challenge of smart matter will require us to meet all the challenges of assembly and differentiation. Whether it is assembling computer systems out of simple homogeneous molecules or software systems out of simple composable components.

BIO:

Dr. Seth Copen Goldstein received his Ph.D. in Computer Science at the University of California at Berkeley in 1997. In 1994 he completed his M.S. in Computer Science at the same institution. His undergraduate work was undertaken at Princeton University in the Department of Electrical Engineering and Computer Science.

Currently Dr. Goldstein is an Assistant Professor in the School of Computer Science at Carnegie Mellon University. He is also co-director of the center for interdisciplinary nanotechnology research. Before attending UC Berkeley, Seth was CEO and founder of Complete Computer Corporation which developed and marketed object-oriented programming tools.

Seth is currently pursuing research in the areas of electronic nanotechnology and reconfigurable computing.

URL: www.cs.cmu.edu/~seth
EMAIL: seth@cs.cmu.edu
PHONE: 412-268-3828