A New Framework for Computer Science and Engineering

Paul S. Rosenbloom
University of Southern California

To realize the compelling future of computing studies, researchers and analysts must rethink the traditional way of partitioning and structuring computer science and engineering. A framework that USC is already partially implementing combines an explicitly interdisciplinary academic focus with a systems-oriented perspective.

Traditionally, computing studies has two partitions—science and engineering—that are separated by a line roughly at the computer architecture level. Above the line is computer science and below is computer engineering. Each of these two disciplines then breaks into subspecialties, such as the decomposition of computer science into theory, systems, and AI.

Such an approach succeeds in dividing the field into affinity groups, but it tends to isolate its respective parts and the systems that come from them. This isolation, in turn, makes it difficult to incorporate either an integrative systems-oriented perspective or the interdisciplinary work that is rapidly becoming critical to computing studies' future.[1] A more effective organization for computer science and engineering (CS&E)—and one that helps point the way toward its promising future—requires an intrinsically interdisciplinary framework that combines academic and systems-oriented computing perspectives.

My colleagues and I have been developing such a framework, which is already changing key segments of CS&E at the University of Southern California (USC). The framework reaggregates computer science and computer engineering and then repartitions the resulting single field into analysis and synthesis components.[2-4] *Analysis,* the more academic of the two, focuses on the nature of computing and the relationships between computer science and other major scientific domains. The *synthesis* component builds up computer engineering as a more pragmatic systems-oriented activity through a hierarchy of related layers, each of which covers important interdisciplinary issues.

The framework is based on the notion that science is foremost about dissecting and understanding, and engineering is mostly about envisioning and building. True, scientists also envision and build—integrated models, for example—but such activities are directed mostly at dissecting and understanding something. Likewise engineers also dissect and understand things, but primarily to envision and build.

Extending these notions to computing studies, computer science is concerned with dissecting and understanding the nature of computing and its relationship to the other sciences, while computer engineering is concerned

0018-9162/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

Table 1. Decomposition of binary computer science into subareas by domain and type of relationship.

	C + P	C + L	C + S	C + C
Implementation (/): Technology from the physical sciences (P), life sciences (L), or social sciences (S) is used to implement computation (C) or computation is used to implement (possibly a model or function of) an aspect of one of the domains				
C/*	C/P: Silicon and quantum computing	C/L: Biological and neural computing	C/S: Wizard of Oz	C/C: Languages, compilers, operating systems, emulation
***/C**	P/C: Modeling and simulation, data/information systems	L/C: Artificial life, biomimetics, systems biology	S/C: Artificial intelligence	
Interaction (•): A symmetric relationship in which two domains interact as peers				
C•* and *•C	C•P and P•C: Sensors, effectors, robots, peripherals	C•L and L•C: Biosensors	C•S and S•C: Human-computer interaction, authorization	C•C: Networking, security, parallel computing, grids
Embedding ([]): Some fragment of one domain is embedded in another				
C[*]	C[P]: Analog computing	C[L]: Autonomic systems	C[S]: Immersion	C[C]: Embedded monitoring and testing
***[C]**	P[C]: Embedded computing	L[C]: Cyborgs	S[C]: Cognitive prostheses	

with the more pragmatic aspects of envisioning and building real computer systems. Our framework is a product of these two threads.

COMPUTER SCIENCE: AN INTERDISCIPLINARY ANALYSIS

The precomputer age saw the development of three great scientific domains: The *physical sciences* (P), which focus on nonliving matter; the *life sciences* (L), which focus on living matter; and the *social sciences* (S), which focus on humans and their societies. Researchers investigated each domain in its own right, but eventually the overlapping areas also became critical targets. Much of the current excitement flows around disciplines that represent overlaps between P, S, and L, such as molecular biology (P + L), cognitive neuroscience (S + L), and human ecology (S + P).

After the invention of computers, a fourth great scientific domain opened up: *computer science* (C), which focuses on computing. At C's core is the essence of computing—computational theory, algorithms, computer architecture, and so on. This is C in isolation, or what can be called *unary* computer science.

More broadly, however, the new domain concerns not just C in isolation, but its relationships with P, L, and S and with itself. *Binary* computer science takes a step in this direction by combining C with one other domain (P, L, S, or C). Much of the intriguing work in computing over the past few decades has been binary; for example, quantum computing implements computing via physical phenomena (P + C), and AI implements the functionality of human cognition via computing (S + C).

As Table 1 shows, it is possible to decompose binary computer science along two dimensions: the domain that combines with computing and the type of relationship between the two domains (earlier 2D decompositions of computer science can be found elsewhere[4,5]). What makes all this computer science is clearly C's essential role. However, the focus is not on C in isolation, but on C in the context of interdisciplinary relationships with P, L, S, and itself. The future of computer science is in these relationships.

Implementation

An *implementation* relationship (/) holds between C and P, S, L, or C if one domain uses technology from the other to implement computation or uses computation to implement (possibly a model or function of) aspects of one of the other domains. Silicon computing, for example, involves the physical implementation of computing on silicon, C/P. Inverting this relationship yields modeling and simulation—that is, the computational implementation of models of physical phenomena, P/C. Neural computing involves implementing computation in a neural substrate model, C/L. Artificial life inverts this by modeling a living organism on a computer, L/C.

In Wizard of Oz experiments, a human implements computer functionality—in essence, emulating a computer, C/S. In contrast, AI emulates a human mind on a computer, S/C. Compilers implement computation—specified in a high-level language—in a form that is closer to what the hardware provides, C/C.

Implementation is the traditional relationship among domains in a hierarchical analysis of science, as seen, for example, in the interface between physics and chemistry in physical chemistry and in the relationship between chemistry and biology in biochemistry.

Interaction

An *interaction* relationship (•) holds between C and P, S, L, or C if they interact as peers. Table 1 has only one row for interaction because it is a symmetric relationship. Interaction between computation and the physical world, C•P, focuses on input

and output devices in their various forms—whether they are mundane devices such as keyboards and printers or more exotic ones such as robotic sensors and effectors. Interaction between computation and the biological world, C•L, concerns sensors for biological substances and processes and the effectors that can intervene in them.

Interaction between computation and people, C•S, concerns primarily the traditional discipline of human-computer interaction, but it also includes issues such as the authorization processes that entitle people to access computational resources. Interaction among computers, C•C, concerns how they communicate among themselves (networking) and how groups of communicating computers work together (parallel/distributed computing and grids).

Embedding

Although *embedding* is somewhat like interaction, in an embedding relationship one domain wholly surrounds the other to the point that the embedded component effectively becomes a part of the embedding domain rather than maintaining its own identity in a peer relationship.

An embedding relationship ([]) holds if some fragment of one domain is embedded within another. For example, analog computing involves embedding a physical process within a computation to directly perform part of the computation, C[P]. Embedded computing, in contrast, submerges computation within the physical world, P[C].

Autonomic systems embed lifelike processes within computing, C[L], while cyborgs are living organisms that have computing embedded within them, L[C]. Immersion submerges a human in a computational environment, C[S], while cognitive prostheses embed cognitive aids within a human, S[C]. Finally, embedded monitoring embeds computing within computing, C[C].

Beyond binary computer science

Things rapidly get more complex if the number of distinct domains or relationships between domains increase, leading to *ternary*, *quaternary*, and even higher orders of computer science. At some point, counting loses its purpose, however, and it makes more sense just to look at the actual relationships.

Clearly, much of computer science's future is embodied in these higher-order relationships, and all this work is highly interdisciplinary. In ubiquitous computing, for example, people interact with a world that embeds computing, S•P[C].

In intelligent robots, emulated human thought interacts with the physical world, (S/C)•P or P[S/C]•P. Embedded biomimetic systems attempt to replace the faulty components of living bodies with computationally implemented models of the biological components, L[L/C]. Virtual humans attempt to implement the functionality of the human body and mind, L[S]/C or (L[S]•P)/C.

Virtual organizations comprise teams of people and intelligent agents working toward common goals.

Virtual organizations comprise teams of people and intelligent agents working toward common goals, $S^m \bullet (S/C)^n$ (where m and n are exponents representing arbitrary numbers of people and agents interacting). Complex virtual worlds simulate not only the physical objects and processes within some world, but also its social members and behaviors, (P•S)/C, or even its living organisms and computational infrastructure, (P•S•L•C)/C.

The exploding size and complexity of higher-order possibilities make it difficult to cover this space systematically. An intriguing open question is how to use this space of possibilities. Can it aid in generating useful ideas for new CS disciplines, or will it serve primarily as a post hoc analysis and organizational tool?

An equally intriguing question is the implication for the academic CS&E enterprise. The field can still be viewed according to the traditional decomposition into theory, systems, AI, and hardware by noting that C is predominantly theory, C + C is predominantly systems, C + S is predominantly AI, and C + P is predominantly hardware.

Similarly, Peter Denning recently proposed understanding much of computing in terms of "five windows of computing mechanics,"[6] which map onto this perspective as

- *computation*—what can be computed; limits of computing, C;
- *communication*—sending messages from one point to another, C•C;
- *coordination*—multiple entities cooperating toward a single result, $C^m \bullet S^n$;
- *automation*—performing cognitive tasks by computer, S/C, P[S/C]; and
- *recollection*—storing and retrieving information, C+C.

Both the traditional approach and Denning's cover much of unary and binary CS&E, with the notable exception of relationships with the life sciences, C + L, but sustaining them organizationally becomes increasingly difficult as the field focuses more of its energies on higher-order interactions.

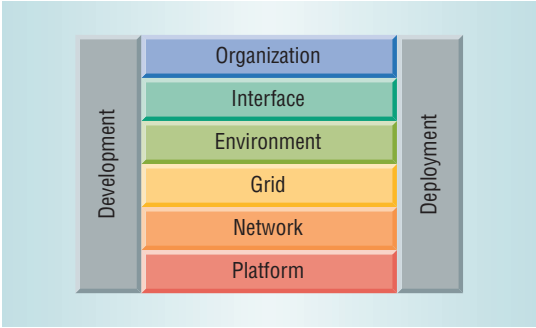

Figure 1. Computer engineering hierarchy. The layers emulate the structure of computer systems, with each layer contributing critical capabilities to the layers above it. The pillars on either side support all layers with key development activities, such as tool creation, and deployment characteristics, such as security.

An alternative organizational approach is to restructure the academic CS&E enterprise explicitly around these relationships. The field as a whole would then cover all the relationships, but the organizations within it would likely concentrate on only a few relationships at a time. The CS department at USC, for example, recently committed to focusing on four themes in realizing computing's future:

- *computation*—computational modeling (including simulation and optimization) and new and emerging computational models, especially in neural and genomic computing, P/C, C/L;
- *interaction*—the coordinated investigation of distributed information technology, C•C;
- *autonomy*—intelligent agents embodied in both hardware (robots) and software, S/C, P[S/C]; and
- *immersion*—the natural and effective interactions among people (both real and virtual), computation, and the world through improved embeddings of each within the others, C[*], *[C].

The CS department has reorganized its faculty, research areas, hiring process, and course assignments around these themes. The responsibility for junior faculty recruiting is now divided across four committees, one for each of the four themes. Both courses and faculty are partitioned by theme, with the associated curriculum and teaching responsibilities for each theme assigned to that theme's faculty group.

COMPUTER ENGINEERING: A SYSTEMS-ORIENTED SYNTHESIS

Given that computer engineering is about envisioning and building computer systems, its structure should emulate the structure of computer systems, rather than the kind of decomposition that computer science follows.

The hierarchy in Figure 1 is based on the notion that computer systems are built from technology layers, with each layer capturing a coherent technology thrust that builds on the layers below it and contributes critical capabilities to the layers above it.

Computer engineering's future is in the challenges these layers present and the ways in which the layers build on each other to enable the construction of real systems.

Hierarchy layers

At the bottom of the hierarchy is the *platform* layer, which provides the computational hardware. (In the grand scheme of things, lower layers do exist, starting with the microsystems layer directly under the platform layer and proceeding down to fundamental physics, but these are beyond the scope of this article.)

The main challenge at this layer is providing abundant resources appropriately embedded within the environments in which the system is to be used. Physically, the platforms could be computers, personal digital assistants (PDAs), phones, sensor nodes, equipment, watches, robots, satellites, or cars.

Research issues at the platform layer include breaking the von Neumann bottleneck to generate petaflops of performance, reducing size and power usage, combining computational and noncomputational hardware (as in sensor nodes that combine sensing, computing, power, and radios to monitor physical environments), embedding computing into challenging environments (such as outer space and the human body), and developing mobile and reconfigurable robotic platforms.

The *network* layer provides connectivity among multiple platforms. The main challenge at this layer is universal connectivity: providing 24/7 connectivity among any number of different types of platforms regardless of location.

Research issues at the network layer include coping with the constraints that nontraditional embedding and communication environments impose, end-to-end performance, security, and the dynamic creation and management of large-scale networks.

The *grid* layer converts networks of platforms into shared-resource pools. The main challenge is in providing services that yield uniform access across the full set of networked resources despite their distribution across geographical and organizational boundaries.

Grid-layer research issues include cross-domain authorization and security, resource discovery and

allocation, data movement and interoperability, workflow management, and metadata description.

The grid layer supports the development of *environments* that embody the data, information, knowledge, and models that a domain comprises as well as the processes that operate on these environments to generate additional content and insights, such as calculation, fusion, reasoning, and simulation.

The main challenge at the environment layer is how to organize resources over entire domains of interest, such as a science or engineering discipline, a complex multiscale system such as the human body, or a geographic region such as a city. Research issues include how to acquire, organize, and extract implicit conclusions from large bodies of heterogeneous content and how to relate the content to the real-world system it represents (for example, using sensors that provide real-time links between the two).

Users access environments through *interfaces*. The main challenge at the interface layer is providing a low-overhead way for people to effectively interact with the content at the environment layer. Research issues include bidirectional communication of information and instruction, adaptation of interactions to needs and situations, use of human-human interaction modalities—such as natural language, speech, gesture, sketches, and facial expression—for human-environment interaction, visualization techniques for very large data sets, and mixed-initiative, multimodal dialogue.

On top of the interface layer is the *organization* layer, where the main challenge is supporting groups of people—and goal-oriented systems such as agents and robots—in working toward common goals. Research issues include dynamically creating such organizations; enabling them to span geographical, organizational, and political boundaries; providing coordination and collaboration assistance; and converting loose communities into effective organizations.

An integrated application scenario

Beyond the issues involved in developing individual layers is the challenge of combining them into fully functional systems. Consider the development of a system for responding to unexpected events in urban environments, such as large-scale earthquakes or high-impact terrorist attacks (www.isi.edu/crue/research/report.html).

Starting at the top of the hierarchy in Figure 1, at the organization layer, response to unexpected events requires the dynamic assembly of an effective response team from the available participants—first responders from local, state, and federal agencies; nongovernmental organizations such as the Red Cross; community groups; the media; and private citizens—who might not have worked together previously. This virtual organization must be able to access information through interfaces that are quick and easy to learn and that facilitate rather than interfere with getting the job done in time-critical situations.

A virtual organization must be able to access information through interfaces that facilitate rather than interfere with getting the job done in time-critical situations.

Through these interfaces, the participants need access to the environment layer, where sits all the relevant information about the metropolitan area of concern—including buildings, roads, utilities, hospitals, schools, and traffic patterns—and the dynamic data about the unfolding event, preferably based on real-time sensor data and reports. Participants should also be able to integrate this information as needed to answer questions and simulate hypothetical outcomes to aid decision making.

A grid layer is required to support this virtual city environment's data and computational requirements, which itself must be supported by a network that connects all participants and resources, and makes the resources available even if the event has unexpectedly damaged the infrastructure.

Finally, all this bottoms out in a set of computational platforms that the response team can use, such as desktop computers, laptops, tablets, PDAs, mobile phones, and supercomputers.

Combining these solutions provides what some have called a cyberinfrastructure, in this case for responding to unexpected events.

Development and deployment

Although this response system example illustrates the comprehensiveness of the Figure 1 hierarchy in terms of application functionality, it omits any recognition of the range of tools required to develop such a system or the range of system characteristics that successful deployment requires. The two pillars on either side of the hierarchy provide this support—similar to the design and computer practices components that Denning proposed to complement the five windows of computing mechanics.[6]

Developing computer systems requires the support of languages and abstractions for expressing large, complex systems; compilers and debuggers for these languages; and design and construction aids such as CAD systems, programming environments, learning and optimization algorithms, and knowledge-acquisition tools. Computer system

deployment requires dealing with issues such as security and privacy, autonomy and robustness, scalability, performance, dependability, usability, and maintainability.

Development and deployment are pillars in Figure 1 because these activities are critical across all the hierarchy's layers. Deploying secure systems, for example, requires attending to issues not just at the network layer, but at all layers. Security is needed at the platform layer, where the designer must isolate processes/threads from each other. At the grid layer, the system must share services without compromising security. At the environment layer, issues of privacy come to the fore. At the interface layer, user authorization is critical. Finally, at the organization layer, trust issues arise, and protection from both intruders and insiders is vital.

At the USC Viterbi School of Engineering's Information Sciences Institute, a strong focus on building working prototype systems goes hand-in-hand with fundamental research, making it an excellent test bed for the structure in Figure 1. Indeed, the structure is proving particularly valuable in strategic planning. By laying out the scope of what complete systems require, the structure aids in identifying gaps in key technology coverage (such as a relative weakness in research on development tools). By laying out a main challenge at each layer and delineating how the layers interrelate, we are better able to develop larger-scale technology thrusts across projects and subareas. For example, we are taking a strongly cross-layer—and thus cross-project and cross-subarea—perspective in looking into the concept of a virtual city.

Rather than partitioning CS&E at the top level into hardware and software arenas, our idea is to partition the combined discipline into analysis and synthesis components and to use an inherently interdisciplinary framework to examine each. The framework's interdisciplinary perspective is already impacting key segments of the CS&E enterprise at USC. Ideally, the work there will also help point the way toward a compelling future for CS&E as a whole. ■

References

1. J. Hartmanis and H. Lin, eds., *Computing the Future: A Broader Agenda for Computer Science and Engineering*, Nat'l Academy Press, 1992.
2. S. Amarel, "Computer Science: A Conceptual Framework for Curriculum Planning," *Comm. ACM*, June 1971, pp. 391-401.
3. B. Arden, ed., *What Can Be Automated? The Computer Science and Engineering Research Study*, MIT Press, 1980.
4. P. Denning et al., "Computing as a Discipline," *Comm. ACM*, Jan. 1989, pp. 9-23.
5. P. Denning, "Computer Science: The Discipline," *Encyclopedia of Computer Science*, A. Ralston and D. Hemmendinger, eds., Nature Publishing Group, 2000, pp. 405-419.
6. P. Denning, "Great Principles of Computing," *Comm. ACM*, Nov. 2003, pp. 15-20.

Paul S. Rosenbloom *is a professor of computer science at the University of Southern California and the associate director of the USC Viterbi School of Engineering's Information Sciences Institute. His current focus is on conceptualizing, instigating, and coordinating interdisciplinary efforts across computer science and its interactions with other disciplines and society. Rosenbloom received a PhD in computer science from Carnegie Mellon University. Contact him at rosenbloom@isi.edu.*