CHAPTER 3

The Scope of NDC

Since December 1969, when the ARPANET project created the first modern packet-switched network—the genesis of today's Internet—the challenge and promise of NDC has resulted in an explosion of investment, research, and software development. Ensuing efforts encompass nearly all aspects of computer science today.

The scope of NDC is quite impressive. No other single aspect of computer science research and development quite compares with the myriad problem spaces enjoined when computers communicate, swap data, and share processing responsibilities. This chapter presents an overview of some of the many relevant areas of NDC research and development today.

Each of these areas is a moving target, in that while progress is being made in each area and rapid improvement may sometimes be achieved, a complete examination or solution in any of these areas is not likely in the short term. In fact, to the extent that each represents a community of autonomous agents, general fitscape attributes apply. Other categories of NDC will emerge over time, as new technologies converge and evolve and as innovative technology adoption patterns continue to manifest themselves in consumer-driven economic fitscapes worldwide.

The categories here, which I call fitscapes, reflect many of the topical areas of the IEEE Computer Society's Distributed Systems Online journal, which itself is a constantly changing resource that tracks the branching processes so evident in the exploration of NDC today.[1] The categories are derived as well from other fitscapes: ongoing activities of the W3C, for example, and traditional areas of development that may be closely related but are nevertheless subtly different (for example, grid computing versus massively parallel computing).

Table 3.1 lists (in alphabetical order) the general areas of NDC R&D that are explored in this chapter, representing a sampling of the general categories of NDC R&D. Each category is ripe with potential, rich in detail and nuance, and deserving of extensive examination well beyond the scope of this volume.

Table 3.1 Some NDC research and development areas

cluster concepts	distributed storage	peer-to-peer
collaborative computing	grid computing	pervasive computing
dependable systems	languages	real time and embedded
distributed agents	massively parallel systems	security
distributed algorithms	middleware	Semantic Web
distributed databases	mobile and wireless	spaces computing
distributed filesystems	network protocols	ubiquitous computing
distributed media	operating systems	Web Services

This list does not represent all R&D fitscapes within NDC. One obvious area not covered here is that of systems and network management, a pioneering technology in areas like distributed agents and network protocols. Since the management of systems and networks is encompassed in a number of the 24 categories cited in this chapter, it's left out of the overviews in this chapter. Also not considered here are many of the rapidly evolving edge-related aspects of NDC, including user interfaces. As some of the areas considered in this chapter converge over time, that consolidation will give impetus to the NDC edge-computing fitscapes. The fitscapes presented here represent those that are either historically ripe, currently in vogue, or of sufficient industry interest to attract levels of investment beyond one or two research institutions.

There are clear connections between and among these areas—relationships that are presumably recognized by many of the participants engaged in the exploration of each. For example, NDC security is germane to all other categories of NDC development. Distributed agents share ground and influence with at least pervasive and ubiquitous computing and the Semantic Web, and probably others as well. A random clump of these fitscapes chaotically thrown against the wall, with relationships mapped, might appear as in Figure 3.1.

The purpose of this map is not to present a canonical listing of relations and dependencies; indeed, only the most obvious ones are noted here. Every aspect of NDC is directly related in some way to almost every other aspect of NDC, thus making the map moot as an exercise, so don't bother trying to follow the relationships or memorize the dependencies. The point of the map is to illustrate the level of complexity inherent in our efforts to simply articulate the relationships among the areas of research in NDC, never mind the more profound complexities inherent within each.

Imagine the complexities you encounter in keeping track of all the influences and discoveries if, for example, your role is one of conceptualizing distributed agents. How can your work, dependent on NDC-related work in—at least—middleware, security, distributed databases, and possibly operating systems, proceed concurrently with work in those areas? How can you, with many other potential advances also dependent on you, confidently progress? Clearly, not everything can proceed in lock-step. By the same token, it may not be obvious, nor is it reasonable, to stage an ordered process whereby advances in subdisciplines of NDC research can proceed. A complex fitscape governs each subdiscipline from which a broader, much more complex, fitscape of overall NDC R&D emerges.

It may be reasonable, however, to estimate possible dependencies that one category of exploration might have on another over time. Work in

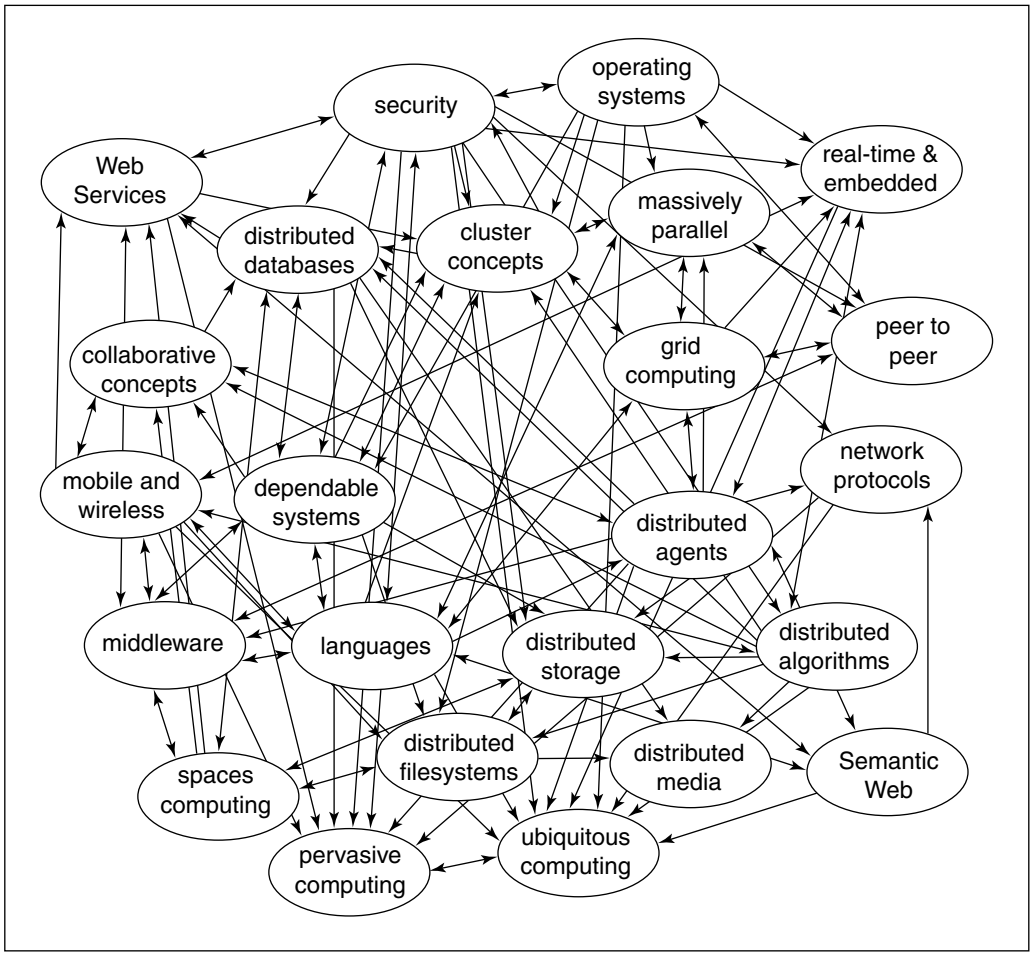

Figure 3.1 NDC R&D fitscapes

some areas will certainly mature more quickly than others, driven by levels of investment, which in turn are driven by technology adoption patterns, and governed by the complexity of the computer sciences issues that must be solved.

The chart in Figure 3.2 offers conjecture with respect to a maturity order of the 24 categories, evolving such as to provide basic solutions upon which NDC developers can build. The *y* axis, *degree of decoupling,* captures component decomposition as well as the movement of intelligence closer to the "edge" of networks; the impact of Moore's law over time, upon which various categories of NDC development will build, is also implied here and will accommodate an even greater decoupling.

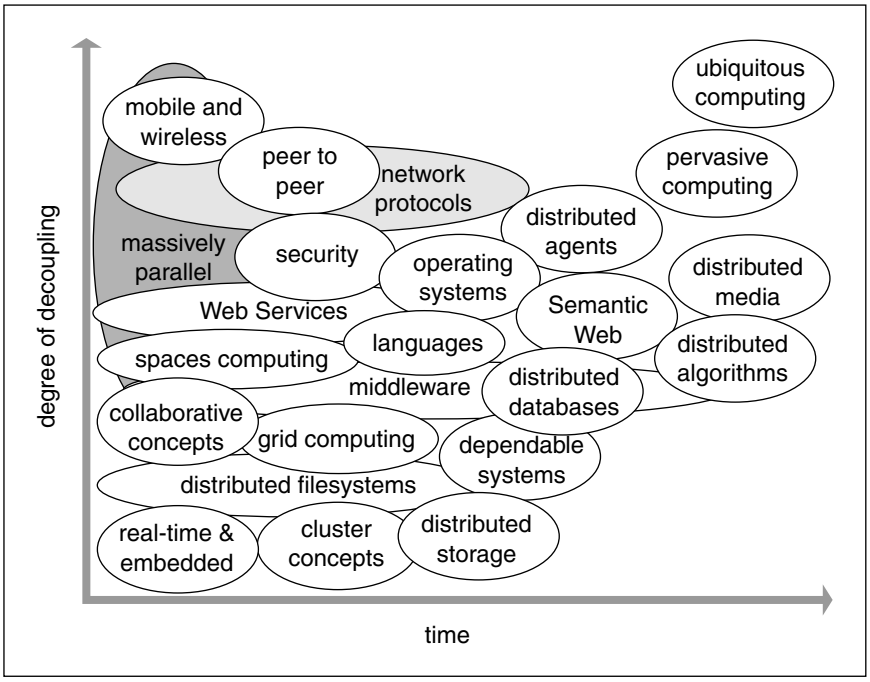

Figure 3.2 Evolution of NDC over time (pro forma)

Given an accelerating rate of innovation in the major technology trends cited earlier (some of which are themselves fitscapes of NDC), it is a given that any estimates of future developments, relationships, or dependencies among these should be viewed as speculative. The odds are perhaps not as high as those against walking into the Atlantis Hotel and Casino in Reno, Nevada, placing a $3 bet in the MegaBucks machine, pulling the handle, and winning the jackpot in one try. But futures in NDC are nevertheless speculative.

I will say more about the Atlantis Casino later in the context of real-world implementation, which, as you may know, is not always by design. Nor is implementation research, strictly speaking by design; implementations are less erudite and much dirtier than speculation, research, or theory would have us believe. But for now, an overview of many of the current NDC R&D fitscapes is in order.

Ubiquitous Computing

In a well-connected network, we can begin with any node and theoretically find our way to each of the others by following links along the way.

Begin with an end in mind, and you'll find lofty ideals touted by an inspired fitscape. The teleological vector of technology,[2] at least that which is heir to Turing's mind child, is ubiquitous computing. Pervasive computing would make information available everywhere; ubiquitous computing would *require* information everywhere. There is a subtle but certain difference, one that will provide NDC challenges for years beyond the near-future, pervasive-computing world that we might soon imagine.

Buildings need to be smart, down to the rivet. Electrical systems need to be smart, down to the light bulb. Monetary systems need to be smart, down to the penny. And all those systems and more need to be connected and available down to the network if the ultimate in ephemeralization is ever to be approached. Are continuing productivity increases necessary? Goff's axiom may apply here as well. What do we call economies that do not grow? The essence of economic growth is increasing productivity. Unless we are prepared both to forgo economic organizational assumptions altogether and begin anew, as it were, with other approaches (which may be even more painful to consider than economic stagnation) and to decline given current assumptions, we cannot turn away from the path of ephemeralization. There is no other direction, therefore, than eagerly toward ubiquitous computing.

Many authors do not distinguish between "pervasive" and "ubiquitous" when it comes to computing visions; even Mark Weiser used the terms synonymously. But I think it's important to be cognizant of the differences and argue that we will enjoy the fruits of one even as we continue to pursue the other. Indeed, we are beginning to see early signs of pervasive computing today. Any city in which I can easily find an "information field," in which dynamic network connections can be enjoined via a mobile computing device, is one in which pervasive computing potential has emerged. Arguably, any place where an I-mode phone can function is a place of pervasive computing. But until all possible computing applications are explored and every niche for network intelligence fully exploited, ubiquitous computing will remain the unseen terminal of a teleological vector.

Once computers disappear and dynamic, ad hoc ensembles of software swarm about like beneficent organisms to serve our every whim, utilizing resources with previously unimaginable efficiencies, a miraculous invisible network may then emerge which is as unfathomable to our early 21st-century minds as a wireless Internet-connected device would have been to a pre-Copernican vision. The Network Age is the age of magic. NDC developers, by virtue of the myriad fitscapes in which we all play,

are the magicians of this new age. Ubiquitous computing is our shared Nirvana—whether we realize it or not.

Web Services

> There is nothing either good or bad but thinking makes it so.
> —William Shakespeare, *Hamlet*

From the sublime to the ridiculous, the first two NDC fitscapes reveal an interesting yin/yang: the transcendent hope of ubiquitous computing versus the gritty reality of Web Services. Arg![3] Reaching Nirvana through the church of Web Services may be a long journey indeed.

I am not a fervent evangelist of the vanilla Web Services approach to NDC. My own biases are drawn to the more *organic* approaches, embodied in spaces computing, mixed with the sensibilities of real-time and embedded computing. In my view, the advent of SOAP was not the brightest day in world of Internet standards. Alas, my personal views are not germane to a reasoned discussion of NDC in a larger sense, so I'll attempt to not color this commentary accordingly—at least not just yet.

Web Services are based on several XML-derived concepts, designed to facilitate standardized, portable exchange of "component" data in NDC enviroments. In the fall of 2001, Microsoft and IBM jointly announced the Global XML Web Services Architecture, which includes XML, SOAP, and UDDI.[4]

- XML: eXtensible Markup Language
- SOAP: Simple Object Access Protocol
- UDDI: Universal Directory and Discovery Interface
- WSDL: Web Services Description Language

The Global XML Web Services architecture would also define specification principles that, in tandem with aspects of the Semantic Web, would give one day give rise to an NDC framework that would be

- Modular—Composable modules that can be combined as needed to deliver end-to-end capabilities (new modular elements enjoined into ensembles as needs arise);
- General purpose—Designed to meet a wide range of XML Web Services scenarios, ranging from business-to-business and enter-

prise application integration solutions to peer-to-peer applications and business-to-consumer services;

- Federated—Fully distributed, designed to support XML Web Services that cross organizational and trust boundaries, requiring no centralized servers or administrative functions;
- Standards based—Based on protocols that are submitted to "appropriate standards bodies."

Component reuse, one holy grail of software development, may potentially be realized if both the Web Services and Semantic Web visions are viable and embraced. Indeed, promises from Microsoft and IBM in the Web Services arena would clearly indicate that component composability is not only within reach, it is imminent. The promise of component reuse and composability is to software what COTS VLSI chips were to hardware; once enabled by NDC, the joint visions of the Semantic Web and Web Services promise a composable NDC architecture. The operative word is *promise.* Composability is nontrivial in any environment that involves distributed systems, and is especially so in NDC. Much like the clarinet player, a component needs more than sheet music and a functioning instrument if a viable orchestration is to be achieved.

The Semantic Web

> The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents, roaming from page to page can readily carry out sophisticated tasks for users.
> —Tim Berners-Lee, et al.

Homo sapiens is a culture-creating species. Our social expressions serve to reduce our needs, individually and institutionally. A hierarchy of needs, identified by Abraham Maslow,[5] is as applicable to institutions as it is to individuals, as a benchmark of cultural provisioning. Language, traditions, economics, and social and political arrangements, all reflect need-reduction epistemologies, which run the gamut from physical survival at the bottom of the pyramid to self-actualization at the summit. Each successive step in the virtuous climb toward a shared peak involves creation of a shared meaning, which itself constitutes an abstract fitscape: culture and meaning.

According to clinical psychologist Paul Watzlawick, the question of whether there is order in our shared reality has one of three possible answers:

1. There is no order, in which case reality is tantamount to confusion and chaos, and life is a psychotic nightmare.
2. We relieve our existential state of *disinformation* by inventing an order, forgetting that we have invented it and experiencing it as something "out there" that we call reality.
3. There is an order. It is the creation of some higher Being on whom we depend but who Himself is quite independent of us. *Communication* with this Being, therefore, becomes man's most important goal.[6]

Either there is order or there is not. If there is not, we perceptively impose order. If there is, we discern it. But in either case, the point Watzlawick makes should be clear with respect to semantics: meaning too is either something we create by perception or something we similarly discern. Regardless, there is a cultural imperative to articulate meaning in such a manner as to facilitate communication.

Although human communication mechanisms are highly redundant and equally ambiguous, the paradoxical nature of communication itself, the handmaiden of culture, never seems to prevent heroic attempts to engineer to its essence. The Semantic Web is perhaps the most ambitious of such efforts to date.[7]

Semantic Web activities, chartered by the W3C, proffer and articulate the following general layers of software, which would ostensibly lead to a meaningful realization of the vision:

- XML—eXtensible Markup Language
- RDF—Resource Description Framework
- Ontologies—Formal definitions of relationship
- Agent—An agent language providing for exchange of proofs (i.e., trust)

In specifying data that can self-describe, XML provides data portability. It was proposed in 1996 by John Bosak of Sun Microsystems and accepted for standardization by the W3C, partially in response to proprietary extensions to HTML that resulted from the now-historic browser conflict of that era.[8] As a language describing language for portability, XML is misnamed. The language would more accurately be described by its ability to

provide a basis for the creation of portable metadata—stories about the stories that data would tell.

RDF builds upon XML, providing a framework for representing that metadata. RDF does not require XML, per se. The specification for RDF is not bound to XML, though XML can (and likely will) be used for implementation of the RDF model. The ultimate goal of RDF is to enable the automation of activities (such as discovery) that are germane to dynamic, ad hoc assemblies of data and behavior.

A standardized metadata framework is not yet enough. Once we can portably express meaning, meaning must be meaningfully expressed as ontologies. Ontologies are collections of information that provide an organizational basis for expression.

The term *ontology,* borrowed from philosophy and referring to a theory regarding the nature of existence, implies a study and classification of that which exists. AI research uses the word to mean a collection of formally defined relations among terms, the most common being a set of inference rules and a taxonomy therein. (A taxonomy specifies classes of objects and relations among them; for example, an address may contain a street, a street number, a city, a zone, a nation, and so on.)

Once data can be represented in a standard manner and organized to provide a basis for common semantic agreement, we can realize the potential of NDC by dynamically knitting together meaningful collections of information, thereby yielding levels of services heretofore only imagined. Widespread, standardized, distributed agents can become viable. Knowledge management, edge-to-edge services, integrated information pools—with the realization of the Semantic Web, the foundation of ubiquitous computing itself is theoretically in place.

Spaces Computing

> A Mirror World is some huge institution's moving, true-to-life mirror image trapped inside a computer—where you can see and grasp it whole. The thick, dense, busy sub-world that encompasses you is *also,* now, an object in your hands. A brand new equilibrium is born.
>
> —David Gelernter

Biologists are much more interested in microbes than in microscopes. Computers are tools at best and annoyances otherwise; for too long, software developers have dwelt on the discipline-specific minutiae while ignoring the big-picture implications that NDC finally forces before our eyes, demanding resolution. If the computer industry itself is to be more than just a passing fad, outside-the-box visions must be heeded. Spaces computing is one of those visions.

David Gelernter of Yale University published *Mirror Worlds* in 1991, well before the general public had heard of the Internet, let alone acquired an email address.[9] In the Mirror World, a software framework could contain all elements of reality we would deign to measure, track, transport, or number; reality could be reflected in software in real time—as could all the relationships among myriad mirrored images. This vision was manifested in the implementation Gelernter pioneered, a system called "Linda," which tendered *tuple-spaces,* a simple persistent storage that transcended nodes and networks, the mirror in which levels of reality could begin to reflect.

With *Mirror Worlds,* Gelernter was hailed as one of the most brilliant computer scientists of the modern era—so celebrated, in fact, that he was victimized by one David Kaczynski, aka the Unabomber, in June of '93, toward the end of the neo-Luddite's anonymous reign of terror. Gelernter was consumed by a lengthy recovery the next several years, the period which saw URLs emerge from obscurity to emblems of the then nascent dotcom mania. Gelernter's vision, however, needed no therapy.

A Sun Implementation

Let's look at JavaSpaces. Sun Microsystems demonstrated JavaSpaces in the spring of 1998 at the JavaOne conference, the annual worldwide gathering of Java devotees in San Francisco's Moscone Center. JavaSpaces was the first commercially viable instantiation of Gelernter's seminal vision, a Mirror World framework, which was enabled by the Write Once, Run Anywhere promise of the Java platform. Built upon Jini network technology protocols, JavaSpaces was demonstrated at the conference by the 14,000 Java Rings given away to conference attendees.

The rings, as shown in Figure 3.3, featured a small embedded processor[10] that ran the smallest of Java virtual machines, a Java SmartCard-specified device that defines a WORA engine for credit-card sized devices.

Figure 3.3 The Java Ring, JavaOne 1998

Once the ring was registered in a central database, each ring bearer's name and coffee preference were stored in their own ring as persistent data that could be accessed upon subsequent connections to the serial readers that were interfaces for such devices. Demonstrations were then made available at coffee-dispensing stations around the event facilities. Espresso, cappuccino, or regular coffee? Decaffeinated? Tea, perhaps? The stored preference on each ring determined the beverage of choice.

But another demonstration of Java Ring technology proved to be even more interesting.

What if all those rings could be used to solve a large problem? What if 14,000 asynchronous, independent, intermittently connected CPUs could be harnessed to serve the needs of one problem? Would that, conceptually, hold value? What other kinds of applications might also be so served? Conference attendees were encouraged to periodically take a few moments, connect their rings briefly to the network, and allow the ring CPU to be used to compute a small part of a large problem.

The fractal mathematics necessary to compute the location and color of each pixel in the 64K image, as shown in Figure 3.4, that emerged over the course of the four days of the event was hosted on those rings. Over a conference period, the image slowly but certainly filled in.

Computing on rings was terribly cool. Equally cool and almost unheralded, however, was JavaSpaces—the "man behind the curtain," the Mirror World framework that allowed all those compute transactions to easily and seamlessly occur.

Since that event, spaces computing has slowly emerged in the Java platform. Jini network technology has found some application beyond the early (mistakenly) device-specific marketing spin it suffered.

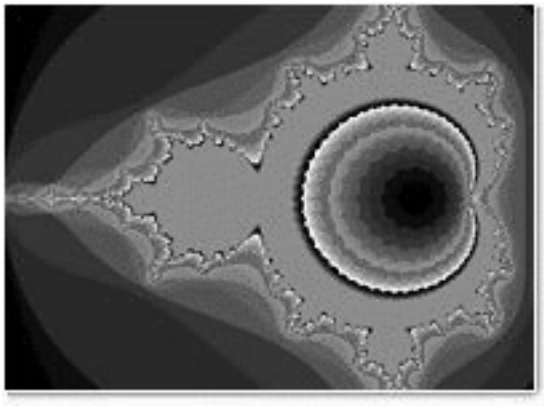

Figure 3.4 JavaRing/JavaSpaces problem, JavaOne 1998

Commercial Potential

Some purveyors of spaces computing frameworks beyond Gelernter's work and JavaSpaces have recently begun to emerge. GigaSpaces, an Israeli software venture, announced a commercially hardened JavaSpaces in 2001 that promises a vital framework for solving myriad NDC problems today.[11] Complementary to, if not competitive with, other approaches like Web Services, spaces computing may yet capture a critical mass thanks to its simplicity, an attribute which may become more dear in an increasingly complex NDC world.

Peer-to-Peer Computing

All the assumptions and institutions that brought us here are being fundamentally challenged by the work we do in computer science. The network metaphor is laden with both promise and foreboding, depending on the context of deployment and the justification of purpose. The very idea of intellectual property and its inherent value (as codified by existing laws and protections) is disrupted in an era in which all information of all kinds can easily be duplicated at near zero costs and transmitted at light speed to any number of receivers.

Napster served as an early warning system insofar as the institutional cognitive dissonance endemic in NDC can harbor.[12] Even as it pioneered the vast potential that can be gleaned in Internet-wide peer-to-peer (p2p)

deployment, it gave p2p a bad name. By utilizing simple, "organic" networking concepts, Napster illustrated both the potential and the unease p2p engenders.

The concepts behind p2p are relatively simple and amazingly effective. One way to understand how p2p works is to view it from the perspective of the node that may need to engage a p2p network.

To begin, the uninitiated node must have some knowledge of at least one peer. The joining of a node to a network has traditionally been a function of hands-on system administration; even with advances like Dynamic Host Configuration Protocol (DHCP), an administrator must be involved at some juncture at least to ensure that any new node has a view of a DHCP server.[13] In most p2p implementations, the address of another peer is required information.

If a node can locate and communicate with a peer node, it is possible that the peer node may have further addresses, as well as descriptions of services that might be available from other nodes. So the newly initiated peer can then discover the existence of other peer nodes and realize the ability to communicate with them simply by knowing of the first peer node. The ability to discover other peers is exponentially derived from knowing the location of the first peer node, and a cascading series of discoveries is thus enabled.

This method of "learning" does not depend on access to a central respository of knowledge about the network. As the World Wide Web grows in scope and degree of information distribution, it may very well be that the more organic approaches to NDC software architecture, like p2p, will be the only ones that can adequately navigate the complex relationships that are emerging.

Collaborative Computing

> The main idea is to regard a program as a communication to human beings rather than as a set of instructions to a computer.
>
> —Donald E. Knuth

It is the collaborative computing fitscape of NDC that produces most implementations of *groupware,* a concept that implies groups of people working together on shared projects. Types of groupware include collaborative drawing and writing tools, frameworks for scientific collaboration, shared

applications, video communications tools, Web-based conferencing, workflow and workflow management tools, the emerging field of knowledge management, and even email. Since groupware, by definition, involves groups of human being, it might follow that academic disciplines with more of a humanities focus might find a haven in research in this area, and such is the case.

Humans beings have the disturbing habit of being human. Our best efforts to systematically impose rational ontologies on human activities fall short, even as we find the tried-and-true scientific method occasionally lacking when it comes to understanding a reality that is inherently subjective, complex, and squishy. Most endeavors to date, for example, in the vein of artificial intelligence, have led to dead ends. Perhaps we are coming to realize that technologies serve us better when we strive to foster human intelligence, rather than replace it. Some collaborative computing research seems to reflect such postcyborg sensibilities, at least in the area of knowledge management (KM).

What is KM? Alas, there is no universal definition. It's probably most useful to think of KM in the broadest context. From a collaborative computing perspective, KM is the process through which organizations generate value from their intellectual and knowledge-based assets—humans. "Best practices" approaches fall within the KM sphere. Much in the way of persistent conversation (like email and instant messaging), if mined properly, has the potential to provide knowledge value to the firm. But there are no solid rules when it comes to KM, except perhaps for one: people are key.

There is no knowledge without human beings. Technology, for example, is not knowledge. It is knowledge incarnate and perhaps a means for collating knowledge. But it's the taxonomy-defying masses that constitute the "mine" from whence knowledge must be extracted. As such, NDC developers who focus on collaborative computing would be well served to learn as much about human beings as possible—through literature, history, religion, economics, biology, sociology and psychology—since it is only through cross-disciplinary activities that the most valuable resource any company may boast can be fully exploited.

Collaborative computing would also be a fruitful pursuit for scientists eager to share data. Biotech researchers, for example, are well aware of the potential of technologies like XML, which, when properly extended, can facilitate collaborative efforts specific to research disciplines. The Interoperable Informatics Infrastructure Consortium,[14] an ad hoc organization whose stated mission is to facilitate and enable data exchange and knowl-

edge management across the entire life science community, is just one example of a discipline-specific instantiation of collaborative computing, which itself is utilizing technologies for other collaborative computing efforts (as embodied by the W3C).

The broad NDC spectrum of collaborative computing is cross-disciplinary by nature; it is likely through R&D efforts in this broad category that humanizing influences will be felt by computer science at large.

Dependable Systems

Fault-tolerant DC was an active research field during the last two decades of the 20th century and continues to be in the current era. Once the domain of mainframe systems, dependability in NDC systems is a natural result of global competitive pressures. *Dependability* in any system can be defined as the ability of the system to ensure that it (and the services it may deliver) can be relied upon within certain measurable parameters, the definition of which depends on the context of deployment. Generic concepts such as reliability, availability, scalability (RAS), and security define dependable NDC systems characteristics. Measures such as mean time between failures (MTBF) traditionally evaluate the reliability for such systems.

As global dependence on NDC continues to increase, the probability of crises rooted in network and system failures also increases. While the consequences of these failures are often petty inconvenience (my pager stopped working), the probability that key application failure could give rise to large economic perturbations or even loss of life also increases. As more NDC applications become the norm, failures too become more distributed. Dependable systems must engender trust from many perspectives if NDC is to continue enriching human activities without introducing equally large measures of risk.

Dependable NDC systems require dependable hardware, which is beyond the scope of this book. A bigger part of the equation, however, is NDC software. A brief discussion of software dependability is germane at this juncture.

Below are two examples of many fault-tolerant software approaches that are applicable to NDC application development—techniques which, when used with other well-engineered development processes and components, will serve to provide more dependable NDC systems software going forward.

Checkpoint-Restart Technique

While discussions of dependable NDC software date back to the earliest experiences with networked computing,[15] a growing body of research in this category parallels the growth of the Internet over the same period. An excellent summary of the state of software fault tolerance status relevant to this era was published in 2000 by Wilfredo Torres-Pomales from the NASA Langley Research Center in Hampton, Virginia.[16] Torres-Pomales cited a number of general approaches to software fault tolerance, many of which are applicable to NDC, including Single-Version Software Fault Tolerance techniques (that is, redundancy applied to a single version of a piece of software, designed to detect and recover from faults). The most common example of this approach cited by Torres-Pomales is the checkpoint-and-restart mechanism pictured in Figure 3.5.[17]

Most software faults (after development has been completed) are unanticipated and usually depend on state. Faults of this type often behave similarly to spurious hardware faults in that they may appear, do their damage, and then disappear leaving no vapor trail. In such cases, restarting the module is often the best strategy for successful completion of its task, one that has several advantages and is general enough to be used at multiple levels in an NDC system or environment. A restart can be dynamic or static, depending on context: a static restart brings the module to a predetermined state; a dynamic one may use dynamically created check-

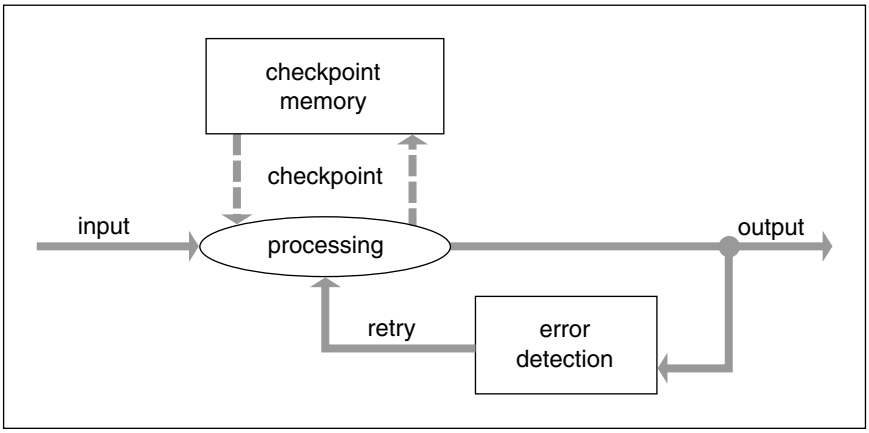

Figure 3.5 Single-version, checkpoint-restart technique

points at fixed intervals or at certain key points during execution. All this depends on error detection, of course, which also has several applicable techniques that can be used.

Recovery-Blocks Technique

Multiversion software fault tolerance techniques are, as the name implies, based on the use of two or more variants of a piece of software (executed either in sequence or in parallel), the assumption being that components built differently (by different designers using different approaches, tools, and so on) will fail differently. So if one version fails with a given input, an alternative version should provide appropriate output.

One example Torres-Pomales cites is a "Recovery Blocks" technique, which shares some attributes with Byzantine agreements discussed later.[18] The Recovery-Blocks technique combines the basics of checkpoint and restart with multiple versions of a given component; if an error is detected during processing in one variant, a different version executes. As shown in Figure 3.6, a checkpoint is created before execution, and error detection in a given module can occur at various checkpoints along the way, rather than through an output-only test.

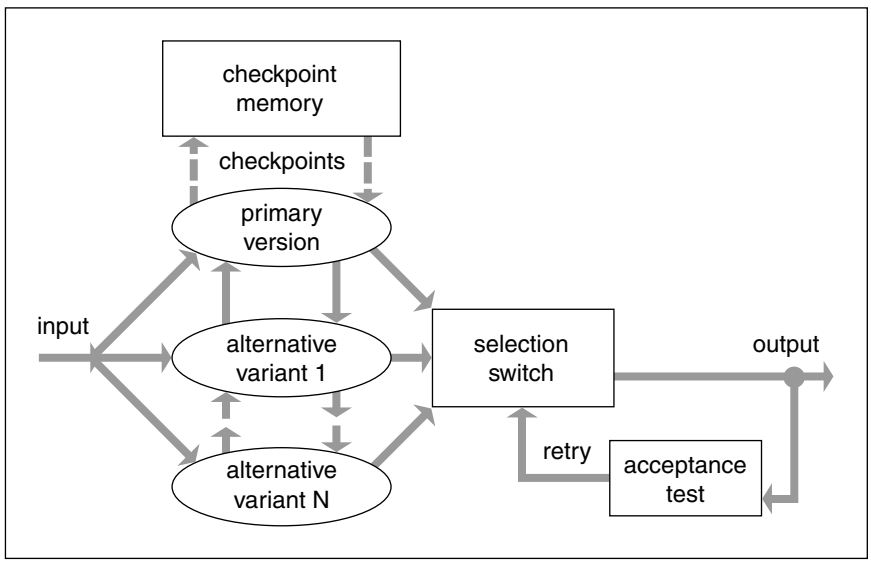

Figure 3.6 Recovery Blocks technique

Although most of the time the primary version will execute successfully, the Recovery-Blocks technique allows alternative variants to process in parallel (perhaps to lesser accuracy, depending on CPU resources available) in order to ensure overall performance if such are the requirements of the application.

Security

Security, like every other fitscape of NDC, can hardly be discussed in isolation. But our sciences are narrow fields of study if measured progress is to be made. Security is based in mathematics but enabled by engineering. and in the context of NDC, any discussion of security is ripe with paradox. In more practical terms, security is mired in encryption, but encryption is not nearly enough to provide reasonable assurances for NDC. Information can be hidden by encryption methods, but encryption doesn't solve other, more basic, issues of trust, including data origin, access control, and privacy.

Juxtaposing engineering and pure mathematics is one way to examine NDC security. Another, more paradoxical, dichotomy is that of privacy versus data transparency, which would yield measurable availability characteristics if done well. The IETF has offered the following definition of security in RFC 2828, intended to define Internet needs:[19]

1. The measures taken to protect a system
2. The condition of a system that results from the establishment and maintenance of measures to protect the system
3. The condition of system resources being free from unauthorized access and from unauthorized or accidental change, destruction, or loss

Security is a proper subset of trust. Trust, however, implies not only security as defined by the IETF but also protections against conditions that are not a function of unauthorized access or even accidental damage. Trust implies a correctness of function, including communications, which might define fault-free computing.

Maintaining security within a single node is simple compared to security issues in NDC environments; most breaches of the security in solitary nodes are due to poor engineering, which can theoretically be addressed. Assuring security in NDC environments in which all hardware components are under the physical control of one owner is also relatively

simple; many systems are designed to run well in such environments, a good example being Sun Microsystem's NFS, which supports secure sharing of data while also providing sufficient data transparency to users. Next in difficulty is the environment in which the endpoints are controlled by the owner but the networks are public; virtual private networks (VPNs) strive to solve NDC security matters in those cases.

Alas, most modern NDC applications cannot be as constrained as these models if they are to ultimately fulfill the promise of ubiquitous computing. Typically, users have their own nodes—perhaps many of them—connections between which are increasingly intermittent, utilizing the random communications fields that mobile and wireless computing make possible. Furthermore, intelligence is migrating to every conceivable niche; Moore's law implies not only more capable traditional systems but also much smaller technology applications, potentially as disposable as the envelope that delivered your last credit card bill. When processing capabilities are pervasive, no assumptions can be made with respect to their nature. Users will store secrets on computer devices as lacking as today's smartcards, which means NDC security requires radically innovative solutions going forward.

In the context of NDC, security weighs in at many levels. It can potentially be "baked in" at a fundamental protocol level, it can be layered in at higher protocol levels, it can be application specific or even network specific. Clearly, however, it cannot be ignored.

Languages

How many spoken languages has our species uttered? Although some 6,000 remain today, despite the language pruning effects of global competitive pressures of the last century, some experts assert that all but 250 to 600 languages will become extinct by the end of the current century.[20] While it might be more efficient from an economic perspective if all humans spoke one and only one language, how immeasurable would the loss be?

Imagine our world today without the influence of Plato, Aristotle, or Archimedes. Imagine Western legal systems without Exodus or the New Testament. Though their languages are officially extinct, civilization today is ineffably indebted to the written words of ancestral giants, many of whose tongues would otherwise be silent in the modern world. Language conveys thought beyond the essence of message-passing. It pro-

vides the foundation for worldview, which cannot be expressed with words but rather between them. Alas, the loss of human language diversity is accelerating—a consequence of our shrinking planet.

Computer languages may be subject to Darwinian selection mechanisms, but once compiled, the binary code will run as long as a processor exists that can execute the target instruction set. In a metaphorical sense the language is still in use, although not "spoken." This too presents a problem for our collective well-being. While spoken languages, if not used by a large enough body of humans, disappear, computer languages can go on for years after the last line of code is compiled. As such, the maintenance of aging code can become problematic, especially when the larger fitscape rewards developers who may be fluent in whatever language happens to be popular at the moment.

An example of this exposure was the much-hyped Y2K problem, which never materalized because so many organizations made considerable effort to ensure that applications were Y2K-aware. But the lessons of COBOL (which is currently still in use, albeit not the language of choice for any number of new NDC development projects) and language paradigm persistence should be clear: the code we write today may last a lot longer than we anticipate. As such, our choice of language is as important as any other project choice we may make.

In the 1970s to 1990s, many computer programs were written that used only a two-character field to contain the year of any particular date to save memory and storage. This practice was based on the assumption that the data and the code written to manipulate that data would not survive past the turn of the century. Unfortunately, developers who made this assumption were wrong. When the calendar turned from 1999 to 2000, applications designed to use only two characters to represent the year were suddenly confused because not only was data storage involved in the shorthand but algorithms were as well. Previously sound code would suddenly break—or so the Y2K story went. But a cascading set of system crashes did not occur. Indeed, the calendar turned without much notice from the IT community. Perhaps due to the investments made by large organizations to correct the problem in advance or perhaps due to the overestimates of Y2K-injured code, the problem in retrospect appears to have been a nonproblem.

NDC efforts have brought computer languages into the forefront of research as well as coolness. For example, the Java programming language arguably bootstrapped a new generation of Internet applications. With the Java specification, the essence of C++ without the shortcomings was married with byte code and the promise of Write One, Run Anywhere, and new models for applications that would span the public networks could be imagined. The Java platform was a solid step in the direction of a network-aware platform. The fact that Microsoft's C# (pronounced "C sharp") platform is a close syntactic and runtime mirror of Java platform designs attests to the visionary appeal that brought Gosling's invention such dramatic acclaim.[21]

But just as nothing stands alone in the Network Age, nothing stands still. The Java platform has evolved considerably since "dancing Duke" first graced a Web page.[22] And since languages are the first line of expressive capability, it follows that computer languages in a general sense should continue to evolve.

As with all the other categories of NDC, language evolution represents its own complex fitscape. Containing layers within layers, a computer language ultimately relies on an underlying theory of computing in order to provide a usable tool set from which NDC developers can choose to implement algorithms of choice. Examples of language R&D that NDC developers should at least be cognizant of going forward include the following:

- **Java/C#**—Object oriented, virtual machine interpreted languages
  The Java platform started evolving from the moment it was announced. From its modest beginning came a standard version (J2SE), an enterprise-aware version (J2EE), a version targeting smaller and mobile devices (J2ME), a community process to enable standardization (the Java Community Process), and much more. The dream of Write Once, Run Anywhere, once a holy grail of computer labs worldwide, unfolded thanks to a C++ type of syntax, a virtual machine, baked-in security considerations, and the collective imaginations of millions of developers around the world. Java (and Microsoft's C#.NET) is one aspect of NDC languages that continues to evolve because of research and fitscape development pressures.
- **XML** (eXtensible Markup Language)—A language for high-level language creation

With XML came the ability to create HTML-like higher-level languages to serve specific needs. Portable data, supported by the portable behavior of the Java platform, proved a compelling vision. But XML provides only the basis for creation of metadata-centric approaches to data standardization. The hard part is the creation of domain-specific extensions; such efforts require communities of agreement, not unlike the JCP. An analogy would be a new spoken language; if I decided to invent a new language, it would provide no benefit to anyone unless others understood and spoke the same tongue. Now imagine that this new language must be created and agreed to by a committee of individuals who will ostensibly use it to compete with each other. That's the inherent problem XML faces.

- **The Fox Project**—A strongly typed intermediate language with proof-carrying code
  This language layer was pioneered by Carnegie-Mellon University. Funded by the Defense Advanced Research Projects Agency (DARPA) of the U.S. government, the Fox Project's goal is to develop language support for building safe, highly composable, reliable systems. For the goal to be reached, a return to the mathematical basis for programming languages is deemed to be a certain requirement. While the current emphasis has been on applications for ensemble composition in embedded systems, the Fox Project is a comprehensive program of research that is applying theoretical foundations of programming languages to development tools and techniques for systems in general. Interesting features of the Fox Project include the following:
  - Typed intermediate languages, which extend the benefits of type safety enjoyed by higher-level source languages to the intermediate and target languages of a compiler
  - Certifying compilers, which provide a foundation for trust-free code dissemination, by which code can be shared in an untrusted environment without sacrifice of safety
  - Proof-carrying code, a technique by which a host computer can automatically verify that code provided by an untrusted agent is safe to execute
- **π-calculus**—A theoretical computing model for mobile code
  Robin Milner of the University of Cambridge is one of the early advocates of this theoretical model. The π-calculus recognizes

> that communication is a fundamental component of theoretical computing models, which differ from other models of communicating behavior primarily in their recognition and treatment of mobility: The movement of pieces of data inside a program is treated exactly the same as the transfer of a message (which can be an entire program) across the Internet. The π-calculus differs from other models in its capacity to simply classify behavioral equivalence among entities, as well as in its patterns of interactive behavior. It holds that previous theory (that is, classical automata theory), upon which most of today's parsers are built, does not appear to be correct when an automaton's actions consist of reactions between it and another automaton. In other words, a fundamental network-awareness needs to be part and parcel of the computational theories upon which our languages (and hence our systems) should be built.

Each of these NDC language developments represents a different layer of computer language implementation, just as each represents the evolving nature of languages in the current period.

These examples of language evolution in NDC have three things in common:

1. Research on all began in earnest in or after 1995 (year zero of the Network Age).
2. All introduce a heretofore absent network-awareness into computer languages.
3. All reexamine communication approaches between distributed computing nodes.

Pervasive Computing

I don't want to carry my laptop around with me any longer. It doesn't matter that it's lighter or more powerful than the first boat-anchor DOS system I used to carry on flights in the late 1980s. It doesn't matter that the battery allows me to work several hours at a time. The fact is, I don't need the laptop; I need access to my data, an application interface that allows me to manage that data, and a connection to the Internet.

Pervasive computing is the idea that information should follow me and become available to me where and when I need it; securely, efficiently, completely, and timely. Very often the term "pervasive computing" is used synonymously with "ubiquitous computing," but there is a very real dif-

ference between computing and network resources that *can be available* anywhere and similar resources that *are present* everywhere. Hence, the different categories.

The pervasive computing category represents adoption patterns as much as it does enabling technologies. For the information that I require to be available where and when I need it, the work in many fields must culminate in fruitful solutions. In fact, we are beginning to realize the potential of pervasive computing, at least in western economies. For example, I was able to access my Sun-proprietary email account from anywhere in the world (given a reasonable Internet connection) since 1996, when I first used my Sun-issued DES Gold "enigma" card, which provided dynamic password access to a Sun employee portal designed specifically to enable mobile workplace interaction. My laptop also featured a virtual private network (VPN) capability that allowed an even finer granularity of access to proprietary information through the Sun wide area network (SWAN), as if I were sitting at a traditional workstation behind a traditional firewall.

Internet cafes began in 1995 with the advent of the browser (coincident with Netscape's initial public stock offering), when what had once been the domain of geeks (the Internet) became lodged in a general public zeitgeist. (This was due, I'm sure, to Netscape's dramatic capital market success; arguably, Netscape's coming-out party was the day the dotcom debacle began.) Despite the public misreading of the potential of the Internet during the late 1990s, the Internet continues to grow, Internet access continues to become more pervasive, and the era of pervasive computing continues to promise increasing productivity and material ephemeralization. The computer science challenges in this fitscape are almost entirely contained by other aspects of NDC, with technology adoption as the central issue.

Cluster Concepts

NDC progress in the fitscape of cluster computing, as well as the use of cluster systems for scientific and commercial applications, involves participants (researchers, developers, and users) from academia, industry, laboratories, and commerce. Advances and trends in this area include but are not limited to the following:

- System modeling and architecture
- Hardware systems and networks
- Single-system images

- System software and tools
- Programming models, environments, and languages
- Algorithms and applications
- Performance modeling and evaluation
- Resource management and scheduling
- High availability
- Scalable clustered systems
- System management and administration

Strictly speaking, clustered systems have not been traditionally thought of as networked distributed systems. Although network datacom can be a key part of the design of any cluster, what we normally think of as networks (or the middleware typically found in an arbitrary network) has not been traditionally involved. This is, however, changing. Because of competitive pressures and the need to incorporate COTS technologies into more and more areas of technology, some clusters too are starting to utilize protocols traditionally found only in less synchronization-demanding applications, even as COTS-level operating system technologies are entering the fray. Beowulf systems, for example, are high-performance clusters constructed from commodity hardware running open source operating system infrastructures, connected by private internal networks running open network protocols.[23]

Key to the notion of clustered computing is the idea of a single-system image. A cluster is generally a group of nodes that are coupled in such a way as to present the image of a single node to an application that runs on or interacts with the cluster. Clusters are designed to either scale or fail over smoothly; when one subnode in the cluster fails, another is there to take its place, with the application (including its users) hardly noticing. Hybrid clusters, those with sufficient subnodes and intelligence to fail over and scale well, are also useful.

Early clusters were more or less proprietary systems; tightly coupled CPUs that shared a common bus for memory and storage to reasonably share data at very high speeds, not very different from multiple-CPU systems. Advances in datacom capabilities have made it possible for clusters to leave the shared-bus environment, existing now across buildings and even across campus. Indeed, a cross-town cluster is not unthinkable, depending on the quality and speed of the dedicated datacom connection.

While the constraints of light speed may make it difficult for computer engineers to build a global cluster today, dedicated fiber-optic capabilities

may one day make such an arrangement imaginable. A global cluster using the public Internet for datacom, however, is not so easy to imagine, given the inherent indeterminacy of the network and the state of Internet protocols today, regardless of increasing bandwidth. But there may yet be clever approaches to overcoming even those intrinsic limits.

Distributed Agents

An *agent* can be many things. The broadest accepted definition is that of an entity that acts on, or has the power or authority to act on, behalf of another. An agent can be thought of as a means by which something is accomplished or is caused.

In any agent-based model, a human being (or even a non-human agent) may delegate some authority to the agent, which may be "intelligent," mobile, or both. Given the false start that AI appears to have suffered, it may not be realistic to speak of intelligent agents in NDC. We can say, however, that an intelligent agent may be able to make rule-based inferences and conduct probabilistic decision analyses or learning on behalf of its client.

In the context of NDC, a *mobile agent* may move between nodes to accomplish assigned tasks; this vision is of particular interest to mobile users and mobile communications. Such a view naturally raises many questions with respect to security and trust-based models, not to mention assumptions regarding code viability on a given node. Recall Deutsch's Eight Fallacies; many of those fallacies must either be ignored or addressed by substantial work at an industrywide level if a standard agent-based model for NDC is to become reality. And given the presumed relationship between the concept of distributed agents and so many other fitscapes of NDC, it is difficult to imagine a scenario that proffers the ultimate achievment of ubiquitous computing (which I consider to be the teleological vector of NDC, if one exists) without such a standard framework.

Not all nodes must sing the same song—quite the contrary. But "agentness" must be well formalized if a semblance of the intelligence needed to traverse an arbitrary network is ever to be realized.

Within NDC, agents and the characteristics of distributed agents were first considered in the problem space of systems and network management. The standardization of distributed agents is now being explored in the context of the Semantic Web as well as the DARPA Agent Markup Language (DAML).

Distributed Algorithms

> The notion of an algorithm is basic to all of computer programming, so we should begin with a careful analysis of this concept.
> —Donald Knuth

A distributed system is a collection of individual computing components that can communicate. This basic definition covers processing organizations ranging from a VLSI chip to the broadest set of cooperative ensembles that might one day be available as a result of resource convergence over the widest of global public networks. The heart and soul of all such systems is the abstraction expressed by the algorithm.

Algorithms are the step-by-step definitions of computational and communications flow. From a research perspective, algorithms are the research domain of theoretical computer science practitioners, as well as practical references for real-world application development. The study of algorithms has proven to be a successful endeavor in solitary computing node arrangements, providing a basis for understanding problems of practical importance as well as affording a framework for articulating intrinsic limitations (such as computability).

But NDC is inherently different from local computing. Indeed, Deutsch's Eight Fallacies would teach that not only are computing models never uniform across the network, but communication ambiguities also provide indeterminate failure attributes that cannot easily be masked. NDC introduces interesting complexity measures mired in time and space variables that are simply not visible to local computing models. And just as there are greater challenges from greater complexities, the plethora of approaches to NDC serves to further complicate the NDC fitscape, making real-world architectures even more difficult to manage.

Three main architectural models, generally differing in degree of synchrony and decoupling, are algorithmically identifiable in NDC today:

1. A synchronous message passing model. RPC-like models include RMI (Jini network technology), JAX-RPC, and earlier synchronous message passing systems.
2. An asynchronous message-passing model. Java Messaging Service (JMS), which provides a basis for Java platform implementations of Web Services, is representative of this approach.

3. An asynchronous shared-memory model. The more tightly coupled approaches to NDC like grid computing, cluster concepts, and spaces computing can all benefit from distributed computing algorithmic advances gleaned from this model.

Algorithmically, systems can be considered to be asynchronous if there is no fixed upper bound on how long it takes for a message to be delivered or for elapsed time of processing between steps. Email, for example, typically takes only a few seconds to arrive but may also take several days, depending on network temperament. As such, asynchronous models cannot, by definition, provide reliable temporal guarantees. On the other hand, the reliability (also understood as uncertainty reduction) of asynchronous models may be enhanced with other benefits like message persistency. Goff's axiom applies here.

In addition to matters of synchrony, network topology is also a domain of distributed algorithms; determining shapes and boundaries of dynamic networks to effectively navigate the growing rivers of datacom is a matter of much interest. The commercial popularity of the Internet has given rise to a substantial increase in research and development in distributed algorithms, as it has so many other aspects of NDC.

Distributed Databases

Distributed databases are sets of databases that are stored on multiple nodes but appear to applications as a single database. Through this mechanism, an application may concurrently access and modify data in several databases on a network. Typically, each database in the meta-database is controlled by a local node but cooperates to maintain the consistency of the distributed database and to provide the application-level illusion that a single database is involved.

Database vendors have had interesting challenges to face for a number of years. Companies like Oracle, for example, have added important value to businesses over the past two decades by balancing mission-critical data demands against the often less-than-reliable COTS technologies that their customers are forced to deploy because of increasing competitive pressures.

The closer a database engine operation is to the hardware, the easier it is to provide reasonable assurances of data viability; by the same token, the closer the database engine operation is to the hardware, the less reliant it can and must be on interfaces provided by the host operating system. To

easily play with COTS technologies, therefore, database vendors face a catch-22, which becomes even more acute as more NDC COTS technologies are enjoined by databases . . . yet another salute to the flag of paradox emblematic of the Network Age.

Because of inherent constraints in NDC, the meaningful data-integrity guarantees that database vendors must deliver become difficult beyond a tightly configured cluster. These challenges must be overcome if a truly distributed database is to go beyond the illusion of the metadatabase.

This is not to say that the larger, enterprisewide, geographically dispersed metadatabase is not tractable; Oracle offers an impressive array of products that today enable even the largest enterprise manager to glean up-to-date data from the globally distributed firm. The data itself, however, is maintained in many databases, but ideally, only one database. Realistically, as we imagine an age of ubiquitous computing, many databases will likely still need to be knitted together in some fashion. The NDC exploration areas of pervasive computing and distributed agents are both related to problems encountered when distributed databases are considered, as are dependable computing, peer-to-peer computing, security, and others.

Distributed Filesystems

In regard to data access, the balancing of guarantees of data availability with security and efficient resource utilization is the product of the distributed filesystems fitscape of NDC. The first viable distributed filesystem was the Network File System (NFS) from Sun Microsystems. NFS enabled distributed storage, distributed media, cluster concepts, collaborative computing, and more. It was used on Sun's UNIX-based workstations exclusively at first (circa 1983) but became a standard file sharing mechanism for many operating systems with the public release of NFS 2.0 in 1985. NFS version 3 came along around 1994; major revisions are currently being implemented to allow better performance across the Internet, ultimately turning NFS into a true WAN filesystem.

NFS is not so much a filesystem as it is a collection of protocols that collectively give rise to a client-perspective distributed filesystem. Key to NFS is the concept of a remote file service that is managed by a remote server. Clients need not be aware of a file's location, but rather are given an interface similar to what a local filesystem might offer; the interface offers various file operations that the server is responsible for implementing. This approach can be viewed as a *remote access model,* as contrasted with an *upload/download model,* as shown in the Figure 3.7.[24]

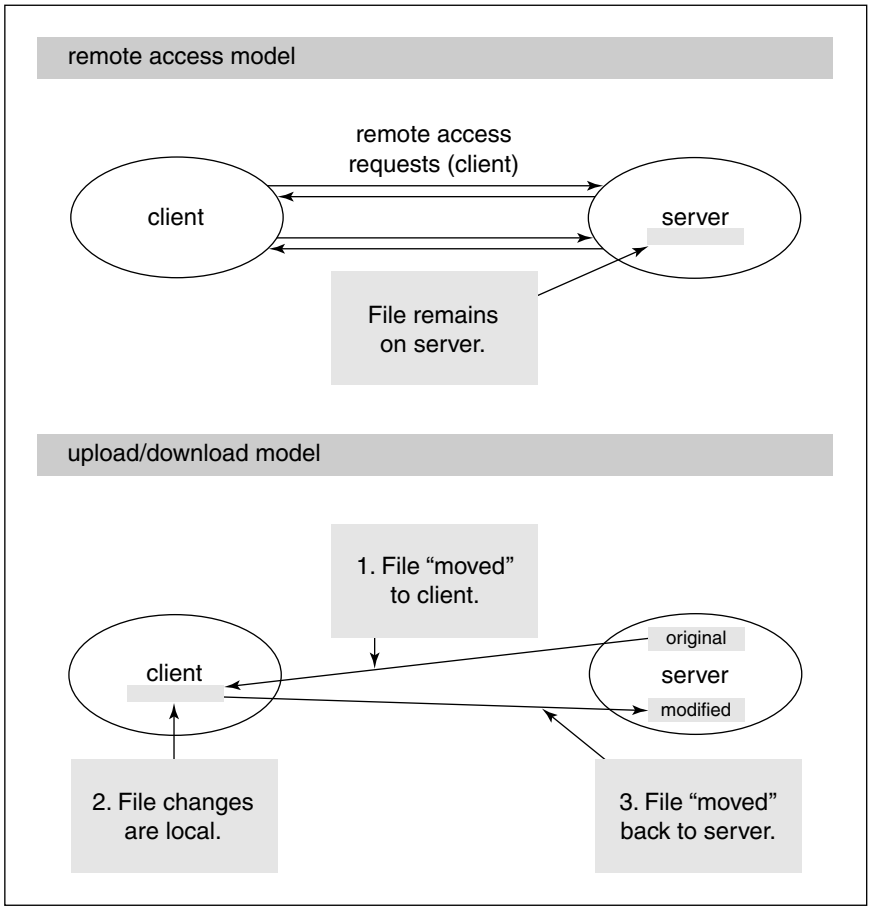

Figure 3.7 General distributed filesystem models

A rudimentary example of an upload/download model is an Internet FTP service when used by a client to obtain, modify, and store data on a remote server.

NFS is implemented on top of RPC. Filesystem interfaces have, for the most part, been abstracted from operating system interfaces to traditional local filesystems to transparently offer distributed filesystem capabilities. A virtual filesystem interface (VFS) has been a standard feature in UNIX and UNIX derivatives since the mid-1980s.

Basically, all requests for file access, whether they be remote or local, go through the operating system's VFS interface. This allows applications to treat all files uniformly, which is of great benefit to NDC application

developers. Nevertheless, it is also important to remember the inherent local/remote differences; the ambiguities and uncertainties of remote computing must be considered at the interface level if uniform filesystems access is to be provided.

When it comes to distributed filesystems, two general approaches should be considered. Either multiple nodes access a filesystem managed by a single node, or the data in a filesystem is distributed across several nodes.

Other notable research in NDC distributed filesystems includes the Farsite work of Microsoft (discussed in the operating systems category below), which provides for replication of files, to increase data availability across arbitrary networks, and the Extensible File System, proffered in the early 90s from research at SunLabs, that explored a stacking filesystem, wherein one filesystem can be stacked on top of an existing one, thereby allowing the sharing of the same underlying data in a coherent manner.[25]

Distributed Media

Distributed media (or multimedia) is one of those catch-all fitscapes of NDC research projects and emerging technologies. Products related to user-friendly multimedia creation, presentation, search, communication, and presentation within distributed environments come from this fitscape area.

Today's euphoria over the World Wide Web does not do justice to the true potential of the Internet. Given Moore's law and Gilder's law (which together give meaning to Metcalfe's law), it is inevitable that the Internet (or more likely Internet2) will support distributed interactivity based on processes that require increasingly larger amounts of data for visualization in real time. If there is a StarTrek-like holodeck in our future, distributed media will be one of the areas responsible for firing it up. Even without such ambitious dreams, this fitscape will enable a new generation of media-savvy users that are prepared for, and in fact demand, such developments.

Consider my 12 year-old niece Tyrell, who lives in an Idaho town so small that all grades in public school—K through 12—share physical facilities. Yet her school is not lacking in resources. One of the courses it offered last year was television production. Using low-cost production gear, made possible by rapid developments in digital media, her class studied hands-on processes and techniques that just a few years ago were limited to only the best-funded schools and professional production studios. Her

entire class had the opportunity to leave behind the technology inhibitions and shortcomings that might have been all but inevitable for an earlier generation as geographically remote as hers. It's this catch-all NDC fitscape, facilitated and enabled by the economic implications of the Nth Law metatrends, that will be the enabler of the most prolific generation of computer scientists and users the world has seen. Telepresence will be as natural as cell phones to my niece and her peers, given their early exposure to distributed media, the tools that create it, and the networks that carry it.

Distributed Storage

The Nth Laws also give rise to an exponentially growing amount of data itself. Vast pools of data require ever-increasing means of storage; Moore's law applies as well to storage capabilities as it does to CPU power. As such, the kinds of storage available today not only feature several orders of magnitude greater capability; they also ensure that legacy storage becomes increasingly outdated. Nonuniform tiers of storage very quickly emerge in any enterprise that has had IT dependencies for more than five years.

The distributed storage category of NDC takes into account the nonuniform tiers of storage available in the modern enterprise, which is itself a fitscape that continues to evolve. In general, the cost of storage is inversely related to the access speed of the storage medium in question. If this were not so, it would probably make sense to have hundreds of terabytes of RAM available for each node and dispense with other storage media altogether. Until an all-optical network and holographic memory systems replace the mountains of disk drives, magnetic tape readers, and removable storage devices currently in vogue, distributed storage will be preoccupied with the costs, access speeds, and reliability of persistent data.

The concept of the storage area network (SAN) was pioneered in the mid-90s, along with so many of the other areas of NDC, stemming, no doubt, from the more commercially uniform emergence of the network metaphor.

The acquisition of computer resources as well as the management and storage of data was an enterprise-centric concern before the 1980s. As such, capital budgets of organizations were impacted by all computer resources, that is, no IT decision was made without careful examination by managers of capital budgets for organizations. With the advent of the first IBM PC in 1981,[26] computer resource purchase decisions were no longer constrained by the more watchful eyes of capital budget managers. An

item that can be expensed in a given year[27] can evade detection, making it much less visible from a corporate budget perspective.

As the PC started to become something "aspirational" in firms,[28] more and more PCs started showing up on desks without specific IT decisions to goad their purchase—flying below the capital budget radar, as it were. The dilemma that firms then had to face was that of corporate data access and backup. Since the PC operating environment quickly became dominated by Microsoft and since Microsoft chose not to bundle TCP/IP with their operating systems until 1995, a data-backup gap ensued that provided a serious problem for corporations that were becoming increasingly dependent upon data stored on compute islands not designed to be connected to the corporate IT infrastructure. The same problem also gave rise to opportunities for vendors to help solve the problem.

Proprietary networking protocols like Banyan Vines and those from Novell provided the basis for networked-PC data management in the era immediately preceding the Network Age. It is from those roots that companies like Veritas, Network Appliances, and EMC produced the basis for what is today evolving into sophisticated data management applications that span the multiple storage tiers and multiple network infrastructures that have grown from those islands of PCs. Storage consolidation is a vital to any firm that is concerned with total cost of ownership. The SAN approach features "full fabric SAN infrastructures," which are highly available and support large-scale consolidation, covering wide area networks, load balancing, and ease of management.

Modern firms require global data management strategies. The need for storage, access, security, and availability of data increases as the volume of data increases, which makes the distributed storage fitscape one of NDC's more interesting fitscapes in terms of increasing opportunity and investment.

Grid Computing

The grid-computing fitscape aims to contribute to the development and advancement of technologies that enable standard, universal access to computing power and resources, all of which are used in a manner similar to electrical power. A computational grid is conceptually similar to an electric power grid. Grid computing envisions the coupling of geographically distributed resources to offer consistent, inexpensive, location-agnostic access to a wide variety of resources, thereby enabling the aggregation

and sharing of such things as supercomputers, computer clusters, SANs, distributed databases, embedded networks, instruments, nodes on an arbitrary network, and, ultimately, people. Solutions for large-scale, CPU-bound, and data-intensive NDC applications are naturals for a grid-computing approach. Once specific to particular applications, grid computing for general-purpose commercial use was first made widely available by Sun Microsystems.

Sun Microsystems was the first large computer vendor to make grid computing available for general-purpose commercial use.* On July 24, 2000, Sun announced the acquisition of Gridware, Inc., a privately owned commercial vendor of advanced computing resource management software that originated in Regensburg, Germany. Gridware developed resource management software, which was used primarily in compute-intensive, technical computing environments, such as electronic design automation. Its products were deployed to optimize utilization of workstations, servers, and dedicated compute farms, an area of strategic interest to Sun.

Thanks to the acquisition, Sun released a general-purpose grid-computing production that allows any organization to reap the benefits of such an approach, for example, the following benefits:

- Specialized agents on each machine to identify and deliver the compute resources as needed
- A GUI and command-line interface for user job submission and control
- A queuing system to manage priorities and to assign jobs to available resources

*Gridware, www.sun.com/gridware/

Clear resource utilization benefits can be achieved with products like Gridware. If an organization can better aggregate the compute power of existing servers and desktop nodes, a highly scalable clusterlike resource (which can include thousands of processors) is the result. Many organizations have made heavy investments in compute resources; many of those nodes remain idle much of the time. Estimates of electricity consumed by Internet-connected nodes in the United States range from 2 percent to 8 percent,[29] which may not seem substantial, but any waste of processor

capability is a waste of resources. The grid computing fitscape of NDC will help address better resource utilization in the aggregate. Interestingly, the Gridware product from Sun also reflects the trend toward ephemeralization with respect to software product cost; the basic engine is free.

Massively Parallel Systems

In the early 1990s, parallel processing systems started to emerge from the shadow of the vector processing supercomputers which they would complement and arguably replace.[30] Parallel processing began with dozens of ordinary processors that could be connected in such a way as to allow simultaneous calculations of different units of data from some larger problem.

Thinking Machines Corporation was founded in 1983 with the intention of providing compute resources to support the always-nascent artificial intelligence field. Business concerns led TMC to adjust, ultimately becoming the market leader for massively parallel systems around 1990. TMC's Connection Machine 5 (CM-5) was the first large-scale, massively parallel system, which utilized a single-instruction stream multiple-data stream (SIMD) architecture, essential for parallel processing.[31] The CM-5 was an all-Sun Microsystems play, featuring a Sun 2000 SMP front-end compile server and a bevy of Sun workstations using SPARC microprocessors working under the Solaris operating environment.

TMC is no more. While a leader in massively parallel systems for a short time, its business model was evidently lacking. But the concepts and history of massively parallel systems echo into the Network Age, with grid computing, languages, and distributed algorithms all beneficiaries of early massively parallel systems (MPS).

Computing approaches like that of SETI@home are similar in many respects to MPS systems. The SETI (Search for Extra-Terrestrial Intelligence) project began in 1984 with a single mainframe system to analyze the data harvested from radio antennae around the world. The ability of one system, regardless of its power, to adequately analyze data across a wide range frequencies and a wide swath of pattern probabilities is very limited. But a project like SETI is easily victim to budget cuts during political cycles when such matters make for visible policy adjustments; no funds were available to buy the needed processing capabilities to adequately do the job. Enter SETI@home in 1995. With the growing deployment of home PCs, which consume electricity whether their CPUs

are busy or not, the advent of the commercial Internet and the browser gave rise to a resource ripe for harvest. The SETI@home project capitalized on this opportunity. By distributing a free screen saver that was actually an application that performed CPU-intensive calculations on discrete units of data gathered from myriad antennae, the project utilized the capabilities of a indeterminately large number of otherwise wasted cycles in a worldwide NDC massively parallel system. Other efforts with similar needs will inevitably follow.

Middleware

In the context of NDC, the middleware fitscape eases the task of designing, programming, and managing distributed applications by providing simple, consistent, integrated distributed programming environments. Middleware is essentially a distributed software layer, or "platform," which attempts to abstract complexities and even mask many of Deutsch's Eight Fallacies from developers.

Platforms

Different middleware platforms support different programming models. One popular model is object-based middleware, in which applications are structured into (potentially distributed) objects that interact by location-transparent method invocation. Examples of this type of middleware are Common Object Request Broker Architecture (CORBA), from the Object Management Group, and Microsoft's Distributed COM (DCOM).[32] Both platforms offer an interface definition language (IDL) that abstracts the fact that objects can be implemented in a variety of suitable programming languages. Both also offer an object request broker, which is responsible for transparently directing method invocations to the appropriate target object, as well as a set of services including naming, time, transactions, and replication, which further enhance the distributed programming environment. A word of caution here, with reference to *A Note on Distributed Computing:* There are inherent differences between local and distributed objects that may not be so easy to ignore. While some work can be accomplished with models like DCOM and CORBA, other NDC applications designs may not so readily lend themselves to this type of approach. (Some of the limitations of DCOM and CORBA from a middleware perspective are discussed later.)

Other Paradigms

Not all middleware is object based. At least two other popular paradigms exist, as well as many derivatives, which are the subject of research and investigation. Event-based middleware and message-oriented middleware are both in use today. Both employ a "single shot" communications approach rather than the request-reply approach found in object-based middleware. Event-based middleware is particularly suited to the construction of noncentralized NDC applications that must monitor and react to changes in their environment. Examples include process control and Internet information channels such as stock tracking, sports score tickers, and the like. Event-based middleware proponents claim this paradigm has potentially better scaling properties than object-based middleware for such applications, which may be a reasonable claim given the less communications-intensive approach. Message-oriented middleware, on the other hand, is biased toward applications in which messages need to be queued and persistently stored. Workflow and messaging applications are examples.

Interesting areas of investigation include charming approaches like "mChaRM: Reflective Middleware with a Global View of Communications," which is a multichannel reification model being pioneered by Walter Cazzola of the University of Genova.[33] In computer science, *reification* (from the Latin *re,* "thing," to regard or treat an abstraction as if it had concrete or material existence) refers to the action by which information is transferred (read or copied) from an internal mechanism of a system into the domain within which it may be utilized as a processing entity. *Reflection* is the action by which information is transferred from the domain of such a system to its internals. The Java Core Reflection API, for example, "provides a small, type-safe, and secure API that supports introspection about the classes and objects in the current Java virtual machine. If permitted by security policy, the API can be used to

- Construct new class instances and new arrays
- Access and modify fields of objects and classes
- Invoke methods on objects and classes
- Access and modify elements of arrays[34]

Walter Cazzola's mChaRM as shown in Figure 3.8 is a reflective model that allows the system to reify multipoint communications instead of objects. Each method invocation can be reified into a multichannel, which can perform metacomputations about it before real activation need occur. In this way, classic communication semantics can be enriched with new

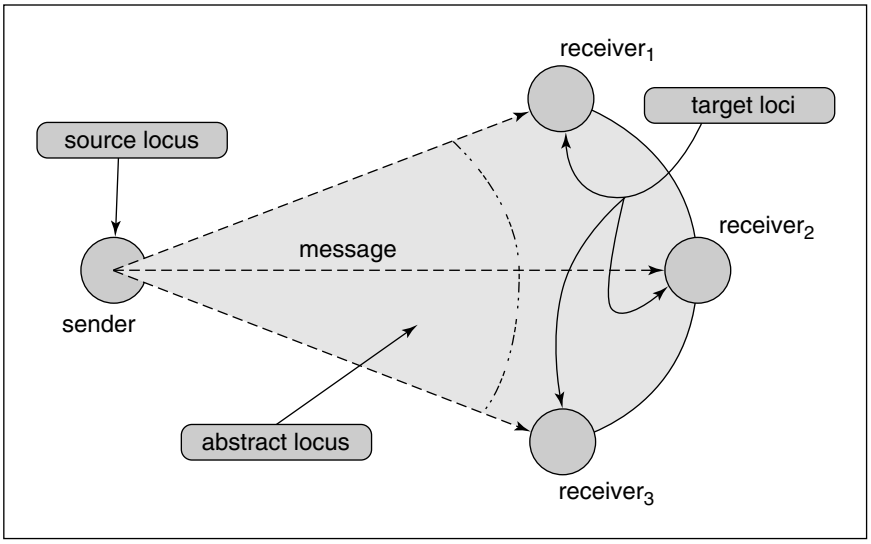

Figure 3.8 General distributed filesystem models: mChaRM, multi-channel reification model (dsonline.computer.org/middleware/articles/dsonline-mcharm.html)

behaviors and properties, and theoretically the flow of communications in an ensemble could be modeled before the ensemble is assembled, which could potentially be very powerful indeed.

Middleware has been responsible for much of the contention, confusion, investment, and progress of NDC. It is also key to the future if progress toward global ubiquitous computing is ever to be realized.

Mobile and Wireless Computing

The mobile and wireless category of NDC is another broad field that encompasses technologies ranging from radio and television broadcasting to pagers, mobile phones, Bluetooth-enabled PDAs, and satellite communications. Even as the newest next-next-generation mobile phone hits the shelf, standards and protocols for that phone are being updated or even discarded. The fitscape is moving rapidly, as is the rapidly expanding wireless world that would serve wireless LANs.

Standards and Protocols

One driver is the acceptance of the WiFi standard. With Microsoft vocally committed to the standard, and companies like Sun Microsystems actively

engaged in disseminating products that enable developers to bring applications to the space, wireless local networking for computers and other devices is spreading rapidly in many cities around the world.

Most wireless communications discussions split technologies into two general types: local and wide area. In time, these discussions will segment even further, as shown in Figure 3.9, ultimately reflecting a broader IEEE 802 vision:

- Personal (wireless personal area network [WPAN]) local (within my clustered campus [LAN])
- Wide (across town—metropolitan area network [MAN])
- Very wide (across planet [WAN])

The problems, opportunities, transmission frequencies, and data rates shift as we traverse the various ranges of wireless datacom.

Radio frequency identification (RFID) is a wireless protocol sponsored by AIM, the global trade association for the Automatic Identification and Data Capture (AIDC) industry.[35] AIDC technology started tracking and access applications during the 1980s, providing for noncontact, low-bandwidth datacom. Effective in manufacturing and other hostile environments where barcode labels are not appropriate, RFID is established in a wide range of markets such as livestock identification and

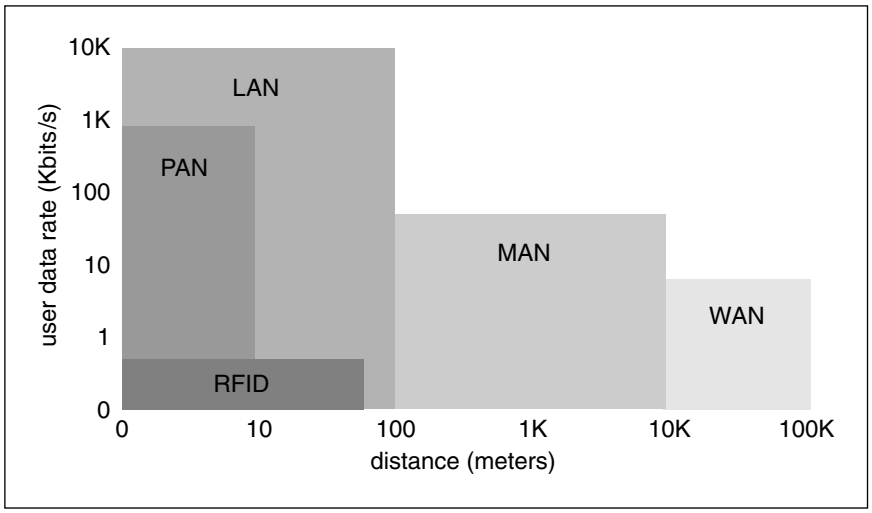

Figure 3.9 Taxonomy of wireless datacom (grouper.ieee.org/groups/802/)

automated vehicle identification systems. RFID is found in "smart labels," which feature thin programmable stick-ons with a very small microchip and antenna to transmit product information. Tracking moving objects is a key feature of RFID. Incorporating RFID solutions into myriad ad hoc personal services will be inexpensive and easy as wireless LAN/MAN solutions provide integratable information fields.

Personal Devices

Personal devices in years to come will include a bevy of connectables, including Internet-enabled eye-glasses with retinal projection systems, which will be linked with voice-activated input devices or hand-sized keyboards (innovative keyboard layouts will be necessary, perhaps a Nintendo-style input device, complementing voice-activated input devices). Such devices will also link with the simple wireless devices common today, such as automobile remotes, garage door openers, 900 MHz cordless phones, and wireless 802.11 networks. All of these devices operate within a short distance, typically just a few meters. Intelligent, integratable wearables that can sense "information fields" that will traverse the range of wireless datacom capabilities will become standardized, very likely around these and other protocols like Bluetooth.

Wide-area wireless devices operate over a much greater area, although many are similar to the more local wireless devices, which then touch global land-based support networks. Most mobile phones fall into this category. Other wider-area (somewhere between LAN and WAN) providers, such as Wave Wireless,[36] are pioneering stationary, broadband-wireless solutions in 18+ GHz frequencies, using proprietary protocols. Solutions for remote broadband (short of satellite, which can be problematic if symmetric datacom services are required), these kinds of wireless solutions can provide point-to-point broad-band (11 Mbps and beyond) covering distances up to 40 kilometers.

Applications

But there's much more to wireless and mobile computing than datacom protocols and frequencies. Indeed, the preceding discussion might have been better suited to the network protocols category in any event. Application requirements are clearly different for mobile devices than for other general areas of NDC, so discussion of wireless and mobile platforms is also germane. Take the Java platform, for example.

The Java platform leads in wireless with the Java 2 Platform Micro Edition (J2ME), specifically designed for small or mobile devices. J2ME technology addresses a large number of intelligent consumer devices, ranging from pagers to set-top boxes. To address wireless and mobile needs, J2ME requires the mobile information device profile (MIDP), a set of Java APIs which when combined with the connected limited device configuration (CLDC) provides a complete J2ME application runtime environment. Targeting all types of mobile information devices, which generally have memory restrictions as well as tiny user displays and limited battery life, the MIDP/CLDC specifications address issues such as user interface, persistence storage, networking, and application model. J2ME through MIDP/CLDC provides a standard runtime environment that allows new applications and services to be dynamically deployed on the growing array of small, mobile devices, effectively enabling such devices to become intermittent nodes on the Internet.

Even As I Write . . .

The wireless fitscape is evolving at an incredible rate. The mobile and wireless category of NDC will have changed substantially by the time the ink is dry on these pages. Developments here also affect and are affected by a number of other fitscapes in the broad scope of NDC.

Network Protocols

Network protocols define the sets of rules governing communication between nodes on a network. Timing, format, sequencing, error control, routing, reliability, and security are all considerations that network protocols either have or will encompass. To cope with burgeoning network complexities, protocols are broken down into steps or layers, each with its own set of rules for operation and data organization. Each layer, in essence, is its own protocol.

Modern datacom is founded on the principles of Shannon's information theory, which consists of a set of elegant mathematical models for describing stochastic processes and ergodic systems made up of such processes. These models give rise to a probabilistic approach to data encoding and transmission characteristics. With Shannon, the ability to encode and effectively transmit binary codes in an environment that is always prone to noise (transmission errors) is made tractable. The stack approach to datacom is enabled at the lowest level by Shannon. The physical transmission

of bits, which are transcribed and framed at the Data Link layer (OSI 7-layer model), relies almost completely on Shannon's information theory. Modern data transmission protocol inventions like DSL, 802.11, Bluetooth, are all founded on Shannon's work.[37]

There are a lot of datacom protocols. Novell offered a proprietary stack called internetworking packet exchange/sequenced packet exchange (IPX/SPX) for use with their NetWare operating system, one of the early players in the PC-networking space. Another is NetBIOS (network basic input/output system) from IBM, a now-standard interface that permits PC applications to communicate directly with other PCs at the transport layer. Which brings us to Transmission Control Protocol/Internet Protocol (TCP/IP), the standard that enables the internet. Loosely based on the ISO OSI 7-layer model,[38] TCP/IP provides basic networking capabilities for computing nodes that provide and consume Internet resources.

Myriad protocols building on top of TCP/IP give rise to other interesting applications. The now-indispensable browser relies on HyperText Transfer Protocol (HTTP), which operates above the transport layer, as do File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and Simple Network Management Protocol (SNMP). With the advent of Web Services, additional network protocols are being considered to enable a new generation of networked applications. Baked-in security and reliability, for example, are both required at the protocol level if the Web Services vision is to be realized. Indeed, additional protocols that build on top of the transport layer are becoming quite common and are likely being utilized as competitive differentiators in a business sense as much as they are standardized communication frameworks in a more secular NDC sense.

In NDC, communications protocols are of utmost importance. Indeed, many of the topics covered later in this volume contain discussion of the protocols germane to them. Suffice it to state here that network protocols are as abundant and complex as are the broad categories of NDC itself—that is, evolutionary forces will likely trim this fitscape too, given sufficient time, consumption, and investment.

Operating Systems

Operating systems have been around since ENIAC, the first modern computer system, and still provide the basis for software development and deployment. My first computer science love was the mysterious and enabling world of operating systems, where software must ultimately meet hard-

ware. Operating systems are responsible for a variety of tasks for a given node, including the following:

- Managing and scheduling resources (CPU, disk, printer, memory, and so on)
- Scheduling tasks, and jobs
- Controlling I/O
- Handling error recovery
- Providing security
- Supplying a basis for user commands

With the proliferation of NDC, operating systems too are subject to change resulting from competitive pressures and network-derived innovations. Once, researchers were content with sensible implementations of a simple IPO model, connecting to the network only as an afterthought. But today, research in operating systems reflects the assumption that any node of interest will be connected to other nodes and share data and processing responsibilities with them. Given this worldview, operating system metamorphosis is inevitable. It should be noted that some operating system designers have long recognized connection to a network as a given. For example, the first system ever sold by Sun Microsystems came with a TCP/IP stack integrated into the system.

Several notable approaches recognize new opportunities for Internet-aware operating systems. One example is the Odyssey project at Carnegie-Mellon University being pioneered by Mahadev Satyanarayanan, a computer science professor who envisions application-aware operating systems, which can adapt as needed depending upon any number of context-dependent variables (e.g., available bandwidth, network load, battery power in a mobile device).[39]

Imagine you're in the back seat of a taxicab enjoying a full-motion color video conversation with your mother on the latest and greatest cell-phone cum Internet portal, via a high-bandwidth wireless MAN, when your cab stops at a light in the shadow of a tall building, temporarily degrading the signal. In this scenario, the operating system provided by the Odyssey vision notifies the (presumably smart) color video application that signal degradation has occurred; instead of the 2 Mbps stream you had so far enjoyed, you are temporarily restricted

to 200 Kbps. The application adjusts from a 10 frame/sec color rendering to something more sustainable, perhaps 2 frame/sec black and white. But your conversation, complete with video, continues.

Another example of operating system research is Microsoft's Farsite (Federated, Available, and Reliable Storage for an Incompletely Trusted Environment) project, which promises a more distributed approach to data storage, fault tolerance, self-tuning, and serverless computing that resembles peer-to-peer sensibilities implemented at the operating system level.[40] Microsoft promises implementations as soon as 2006.

IBM, too, is advertising a self-healing, highly scalable approach they're calling "Blue Gene," which is principally founded on their emerging Autonomic Computing initiative, a program that would imbed operating systems with notions like "self-optimization," as well as self-healing (not unlike the autonomic nervous systems in living organisms).

In 1998, Sun Microsystems announced Jini network technology, which brought such notions as self-healing networks to the forefront of the computer industry. The fact that both Microsoft and IBM too now recognize that requirement is testament to the efficacy of that vision. NDC requires ever more "organic" approaches to software development and network deployment as the level of complexity increases. It is only natural that operating systems too begin to reflect that need.

Real-Time and Embedded Systems

Just as the Internet has grown at a phenomenal rate since its inception, so too have processors that are not connected to an open network, per se, but that provide useful functionality nonetheless. All but the simplest of electrical devices feature an embedded processor today. Why? Because embedding intelligence in a device provides value to consumers. Televisions, coffee pots, telephones, microwave ovens, automobiles, traffic lights, home-climate systems, room air conditioners, CD players, radios, refrigerators, dishwashers, washers and dryers . . . there seems no end to the devices and products that include intelligent processors to meet emerging consumer expectations. And just as the Internet brings great value by connected computing nodes, embedded networks (EmNets) bring great value to real-time and embedded processors. Indeed, the coupling of EmNets and the Internet will provide interesting opportunities in myriad directions over the next few years.

EmNets, especially those with hard real-time requirements,[41] demand tighter specifications and particular sensibilities, typically not required in NDC development. But with the growing integration of real-time and embedded systems intended to leverage and share general-purpose platforms, all developers face design and architectural challenges in managing these newly shared resources. As embedded systems are applied to diverse and potentially adverse environments, appropriate protection of performance characteristics must be ensured. All this has led to increased need for design methodologies for system and application software, operating-system support mechanisms, and resource and application control techniques for these novel forms of embedded systems.

When NDC meets these newly emerging real-time and embedded systems, areas of interest can include such diverse topics as these:

- OS support for mixed response requirements
- Real-time applications in COTS operating systems (e.g., Linux)
- Real-time software components
- Novel kernel-level mechanisms
- Open architectures for resource control
- Embedded control applications
- Secure real-time systems
- Middleware support
- Java implementation and applications
- Power-aware resource management
- QoS-aware application design
- System modeling and analysis

As real-time and embedded networks are knitted together and emerge as services available from the World Wide Web, the sensibilities of each general category (for example, time-bound versus time-agnostic) will influence and impact the other. Further, NDC developers will increasingly become more time aware, even if hard real-time requirements do not immediately constrain network application development.

Commentary

The end of this unordered list of NDC R&D fitscapes marks the beginning of the journey for software developers. There is no avoiding the complexities we've unleashed if we would move forward.

In a little over two decades, NDC R&D, coupled with the immutable laws of supply and demand and seasoned by an early 21st-century shock wave of realization that Moore's law really was an accurate, practical forecasting model rather than science fiction, has given rise to a interleaved group of fitscapes that is beginning to rival some of nature's own more interesting efforts. Not unlike the universe it would help us to understand, the sphere of computer science is enjoying an inflationary era, as the rate of innovation continues to accelerate, following its own exponential vector—somewhere between Moore and Gilder, applauded by Metcalfe.

In this era of ever-changing change, it may be wise to consider Stuart Kauffman's observation that the rate of innovation cannot exceed the ability of the fitscape to adequately test the novelty without risking systemic collapse. If we move too fast, we may experience spectacular collapse, the magnitude of which we cannot even grasp, let alone predict.

If any conclusions can be drawn from an overview of NDC and the relationships among its 24 fitscapes, they can be summarized as follows:

1. Complexity will continue to increase, absent radically different approaches.
2. Formal methods are enjoying a renaissance while standardization efforts both ignore and embrace formal methods.
3. Increasing global competition and COTS technologies will prune the complex undergrowth of NDC R&D options.
4. Technology adoption will continue to increase only if perceived value is obvious to the markets (and hence consumers) which that technology would serve.
5. No one individual, standardization effort, organization, or concern can adequately prestate NDC configuration space. Indeed, as Kauffman asserts, it is very likely impossible.
6. The kaleidoscope of NDC categories are all related; convergence among categories of NDC R&D is inevitable.

Notes

1. dsonline.computer.org
2. The term *teleology,* from philosophy, denotes the study of ultimate aim or purpose in nature. With respect to technology, we can use the term to mean the ultimate aim of technology insofar as it can be discerned.

3. Arg!: a technical term from early email protocol; an emotive signifying frustration.
4. xml.coverpages.org/MS-GlobalXMLWebServicesArchitecture.html
5. Abraham H. Maslow, *Toward a Psychology of Being* (Princeton, NJ: van Nostrand, 1968).
6. Paul Watzlawick, Munchhausen's Pigtail or Psychotherapy and Reality (New York: Norton, 1990), p. 125.
7. Tim Berners-Lee, James Hendler, and Ora Lassila, "The Semantic Web," *Scientific American* (May 2001).
8. The basis for the U.S. government's suit against Microsoft.
9. David Gelernter, *Mirror Worlds* (New York: Oxford University Press, 1991).
10. Dallas Semiconductor.
11. www.gigaspaces.com
12. For a history of Napster, see Joseph Menn, *All the Rave: The Rise and Fall of Shawn Fanning's Napster* (New York: Crown Business, 2003).
13. Jini network technology provides an interesting alternative to this traditional constraint, which is useful to consider as the idea of self-healing becomes more widely held in networks as well as within nodes.
14. i3c.org
15. From "Brief Timeline of the Internet" (http://www.webopedia.com/quick_ref/timeline.asp) "October 1, 1969: Second node installed at Stanford Research Institute; connected to a SDS 940 computer. The first ARPANet message sent: 'lo.' Trying to spell log-in, but the system crashed!"
16. Wilfredo Torres-Pomales, "Software Fault Tolerance: A Tutorial," NASA/TM-2000-210615, citeseer.nj.nec.com/torres-pomales00software.html
17. Dhiraj K. Pradhan, *Fault-Tolerant Computer System Design* (Upper Saddle River, NJ: Prentice Hall, 1996).
18. Brian Randell and Jie Xu, "The Evolution of the Recovery Block Concept," in *Software Fault Tolerance,* Michael R. Lyu, ed. (New York: Wiley, 1995).
19. www.ietf.org/rfc/rfc2828.txt

20. Janet Raloff, "Languishing Languages: Cultures at Risk," Science News Online (February 25, 1995), http://www.sciencenews.org/sn_edpik/aa_1.htm
21. The Java platform (which includes language specification and run-time virtual machine specification—hence, the Java "platform"—was invented by James Gosling of Sun Microsystems and announced by Sun in 1995.
22. Duke, the Java mascot, emerged in the summer of 1992, when the Green Team—the pioneers at Sun who created the Java programming language—built a working demo of an interactive, hand-held home-entertainment device called the *7 ("Star 7"). The *7 featured Duke, an animated character who served as an agent for the user and who could interact with multiple objects on screen.
23. www.beowulf.org
24. Andrew S. Tanenbaum and Maarten van Steen, *Distributed Systems, Principles and Paradigms* (Upper Saddle River, NJ: Prentice Hall, 2002), p. 577.
25. research.sun.com/techrep/1993/smli_tr_93-18.pdf
26. The IBM PC wasn't the first personal computer, per se. But when IBM introduced the 5150 in August 1981, the PC era began in earnest.
27. Generally, tax laws treat capital purchases and expenses differently. Capital purchases are items that must be depreciated over time, whereas expenses are generally deductible from income in the same year as expenditure. Capital purchases usually have a higher minimum amount associated with the purchase, for example, "all items under $5000 can be expensed."
28. It can be argued that many PCs purchased during the 1980s were aspirational as opposed to functional. If I am a first-line manager, for example, and the manager in the cube next to mine has a PC, which is becoming something cool to have, then I too must have a PC.
29. usinfo.state.gov/topical/global/ecom/01020603.htm
30. Vector processors perform CPU-intensive calculations analogous to an assembly line. A central processor doles out the first unit of data, the second processor performs a calculation and hands the task to the next processor, and so on. Vector processing is well suited for

problems that feature well-organized, parallelizable datasets, like calculation of weather patterns. For years all supercomputing was synonymous with vector processors.

31. Multiple-instruction stream, multiple-data stream (MIMD) machines feature processors that function in an independent or asynchronous manner. SIMD architectures are more tightly coupled from a memory perspective and offer superior ability to manipulate vectors, offset by a disadvantageous approach to managing memory exchange.
32. www.omg.org, www.microsoft.com/com/tech/DCOM.asp
33. www.disi.unige.it/person/CazzolaW/, Walter Cazzola, Univ. of Genova, Italy, Dept. of Informatics and Computer Science.
34. Java Core Reflection API Specification, java.sun.com/products/jdk/1.1/docs/guide/reflection/spec/java-reflection.doc.html
35. www.aimglobal.org/technologies/rfid/
36. wavewireless.com
37. C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal* 27 (July and October 1948): 379–423; 623–56.
38. www.webopedia.com/quick-ref/OSI.Layers.asp
39. www.2.cs.edu/~odyssey/
40. research.microsoft.com/farsite
41. Realtime applications can be classified as either hard or soft realtime. Hard realtime applications require a response to events within a predetermined amount of time for the application to function correctly. If a hard realtime application fails to meet specified deadlines, the application is considered to have failed. Soft realtime applications, however, do not necessarily fail if a deadline is missed. An example of a soft realtime application is an airline reservation system where temporal delays do not necessarily constitute failure, although a "reasonable" temporal component is implied.