Wide Area Computation

Luca Cardelli

Microsoft Research

Abstract. The last decades have seen the emergence of the *sea of objects* paradigm for structuring complex distributed systems on workstations and local area networks. In this approach, applications and system services are composed of and communicate among themselves through reliable and transparently accessible object interfaces, leading to the interaction of hundred or thousands of unstructured objects.

This approach has lead to major progress in software composability and reliability. Unfortunately, it is based on a number of assumptions that do not hold on wide area networks. There, access to resources is intrinsically unreliable (because of failure, congestion, voluntary disconnected operation, etc.) and not transparent (because of variations in latency and bandwidth, hardware and software mobility, and the presence of firewalls). These characteristics are so radically different from the current computational norm that they amount to a new model of computation.

We discuss the challenges of computation on wide area networks. Our approach reflects the intuition that, to function satisfactorily on a wide area network, the *sea of objects* must be partitioned and made hierarchical, internally mobile, and secure. This paper is an abridged version of [3].

1 Introduction

The Internet and the World-Wide-Web provide a computational infrastructure that spans the planet. It is appealing to imagine writing programs that exploit this global infrastructure. Unfortunately, the Web violates many familiar assumptions about the behavior of distributed systems, and demands novel and specialized programming techniques. In particular, three phenomena that remain largely hidden in local area network architectures become readily observable on the Web:

- ***(A) Virtual locations***. Because of the presence of potential attackers, barriers are erected between mutually distrustful administrative domains. Therefore, a program must be aware of where it is, and of how to move or communicate between different domains. The existence of separate administrative domains induces a notion of virtual locations and of virtual distance between locations.
- ***(B) Physical locations***. On a planet-size structure, the speed of light becomes tangible. For example, a procedure call to the antipodes requires at least 1/10 of a second, independently of future improvements in networking technology. This absolute lower bound to latency induces a notion of physical locations and physical distance between locations.

- ***(C) Bandwidth fluctuations***. A global network is susceptible to unpredictable congestion and partitioning, which result in fluctuations or temporary interruptions of bandwidth. Moreover, mobile devices may perceive bandwidth changes as a consequence of physical movement. Programs need to be able to observe and react to these fluctuations.

These features may interact among themselves. For example, bandwidth fluctuations may be related to physical location because of different patterns of day and night network utilization, and to virtual location because of authentication and encryption across domain boundaries. Virtual and physical locations are often related, but need not coincide.

In addition, another phenomenon becomes unobservable on the Web:

- ***(D) Failures***. On the Web, there is no practical upper bound to communication delays. In particular, failures become indistinguishable from long delays, and thus undetectable. Failure recovery becomes indistinguishable from intermittent connectivity. Furthermore, delays (and, implicitly, failures) are frequent and unpredictable.

These four phenomena determine the set of *observables* of the Web: the events or states that can be in principle detected. Observables, in turn, influence the basic building blocks of computation. In moving from local area networks to wide area networks, the set of observables changes, and so does the computational model, the programming constructs, and the kind of programs one can write. The question of how to "program the Web" reduces to the question of how to program with the new set of observables provided by the Web.

At least one general technique has emerged to cope with the observables characteristic of a wide area network such as the Web. *Mobile computation* is the notion that running programs need not be forever tied to a single network node. Mobile computation can deal in original ways with the phenomena described above:

- ***(A) Virtual locations***. Given adequate trust mechanisms, mobile computations can cross barriers and move between virtual locations. Barriers are designed to impede access, but when code is allowed to cross them, it can access local resources without the impediment of the barriers.
- ***(B) Physical locations***. Mobile computations can move between physical locations, turning remote calls into local calls, and thus avoiding the latency limit.
- ***(C) Bandwidth fluctuations***. Mobile computations can react to bandwidth fluctuations, either by moving to a better-placed location, or by transferring code that establishes a customized protocol over a connection.
- ***(D) Failures***. Mobile computations can run away from anticipated failures, and can move around presumed failures.

Mobile computation is also strongly related to recent hardware advances, since computations move implicitly when carried on portable devices. In this sense, we cannot avoid

the issues raised by mobile computation: more than an avant-garde software technique, it is an existing hardware reality.

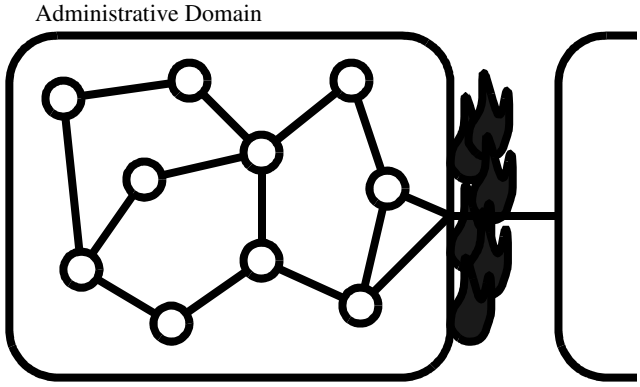
2 Three Mental Images

We begin by comparing and contrasting three *mental images*; that is, three abstracted views of distributed computation. From the differences between these mental images we derive the need for new approaches to global computation.

2.1 Local Area Networks

The first mental image corresponds to the now standard, and quickly becoming obsolete, model of computation over local area networks.

When workstations and PCs started replacing mainframes, local networks were invented to connect autonomous computers for the purpose of resource sharing. A typical local area network consists of a collection of computers of about the same power (within a couple of hardware generations) and of network links of about the same bandwidth and latency. This environment is not always completely uniform: specialized machines may operate as servers or as engineering workstations, and specialized subnetworks may offer special services. Still, by and large, the structure of a LAN can be depicted as the uniform network of nodes (computers) and links (connections) in Mental Image 1:



Mental Image 1: Local Area Network

A main property of such a network is its predictability. Communication delays are bounded, and processor response times can be estimated. Therefore, link and process failures can be detected by time-outs and by “pinging” nodes.

Another important property of local area networks is that they are usually well-administered and, in recent times, protected against attack. Network administrators have the task of keeping the network running and protect it against infiltration. In the picture, the bound-

ary line represents an *administrative domain*, and the flames represent the protection provided by a *firewall*. Protection is necessary because local area networks are rarely completely disconnected: they usually have slower links to the outside world, which are however enough to make administrators nervous about infiltration.

The architecture of local area networks is very different from the older, highly centralized, mainframe architecture. This difference, and the difficulties implied by it, resulted in the emergence of novel distributed computing techniques, such as remote-procedure-call, client-server architecture, and distributed object-oriented programming. The combined aim and effect of these techniques is to make the programming and application environment stable and uniform (as in mainframes). In particular, the network topology is carefully hidden so that any two computers can be considered as lying one logical step apart. Moreover, computers can be considered immobile; for example, they usually preserve their network address when physically moved.

Even in this relatively static environment, the notion of mobility has gradually acquired prominence, in a variety of forms. Control mobility, found in RPC (Remote Procedure Call) and RMI (Remote Method Invocation) mechanisms, is the notion that a thread of control moves (in principle) from one machine to another and back. Data mobility is achieved in RPC/RMI by linearizing, transporting, and reconstructing data across machines. Link mobility is the ability to transmit the end-points of network channels, or remote object proxies. Object mobility is the ability to move objects between different servers, for example for load balancing purposes. Finally, in Remote Execution, a computations can be shipped for execution to a server (this is an early version of code mobility, proposed as an extension of RPC [13]).

In recent years, distributed computing has been endowed with greater mobility properties and easier network programming. Techniques such as Object Request Brokers have emerged to abstract over the location of objects providing certain services. Code mobility has emerged in Tcl and other scripting languages to control network applications. Agent mobility has been pioneered in Telescript [14], aimed towards a uniform (although wide area) network of services. Closure mobility (the mobility of active and connected entities) has been investigated in Obliq [4].

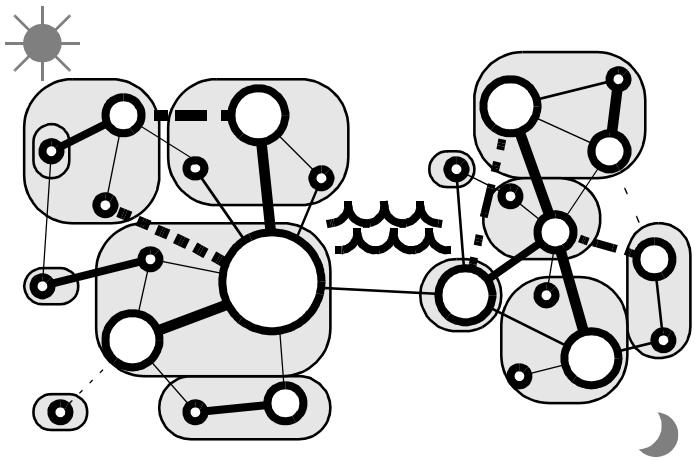
In due time, local area network techniques would have smoothly and gradually evolved towards deployment on wide area networks, e.g. as was explicitly attempted by the CORBA effort. But, suddenly, a particular wide area network came along that radically changed the fundamental assumptions of distributed computing and its pace of progress: the Web.

2.2 Wide Area Networks

Global computing evolved over the span of a few decades in the form of the Internet. But it was not until the emergence of the Web that the peculiar characteristics of the Internet were exposed in a way that anybody could verify with just a few mouse clicks. For clarity and simplicity we will refer to the Web as the primary global information infrastructure, although it was certainly not the first one.

We should remember that the notions of a global address space and of a global file system have been popular at times as extensions of the mainframe architecture to wide area networks. The first obvious feature of the Web is that, although it forms a global computational resource, it is nothing like a global mainframe, nor an extension of it. The Web does not support a global (updatable) file system and, although it supports a global addressing mechanism, it does not guarantee the integrity of addressing. The Web has no single reliable component, but it also has no single failure point; it is definitely not the centralized all-powerful mainframe of 1950's science fiction novels that could be shut off by attacking its single "brain".

The fact that the Web is not a mainframe is not a big concern; we have already successfully tackled distributed computing based on LANs. More distressing is the fact that the Web does not behave like a LAN either. Many proposals have emerged along the lines of extending LAN concepts to a global environment; that is, in turning the Internet into a distributed address space, or a distributed file system. However, since the global environment does not have the stability properties of a LAN, this can be achieved only by introducing redundancy (for reliability), replication (for quality of service), and scalability (for management) at many different levels. Things might have evolved in this direction, but this is not the way the Web came to be. The Web is, almost by definition, unreliable, unpredictable, and unmanageable as a whole, and was not designed with LAN-like guarantees of service.



Mental Image 2: Wide Area Network (for example, the Web)

Therefore, the main problem with the Web is that it is not just a big LAN, otherwise, modulo issues of scale, we would already know how to deal with it. There are several ways in which the Web is not a big LAN, and we will describe them shortly. But the fundamental reason is that, unlike a LAN, the Web is not centrally administered. Instead, it is a dynamic

collection of countless independent administrative domains, all widely different and mutually distrustful. This is represented in Mental Image 2.

In that picture, computers differ greatly in power and availability, while network links differ greatly in capacity and reliability. Large physical distances have visible effects, and so do time zones. The architecture of a wide area network is yet again fundamentally different from that of a local area network. Most prominently, the network topology is dynamic and non-trivial. Computers become intrinsically mobile: they require different addressing when physically moved across administrative boundaries. Techniques based on mobility become more important and sometimes essential. For example, mobile Java applets provided the first disciplined mechanism for running code able to (and allowed to) systematically penetrate other people's firewalls. Countless projects have emerged in the last few years with the aim of supporting mobile computation over wide areas, and are beginning to be consolidated.

At this point, our architectural goal might be to devise techniques for managing computation over an unreliable collection of far-flung computers. However, this is not yet the full picture. Not only are network links and nodes widely dispersed and unreliable; they are not even liable to stay put, as we discuss next.

2.3 Mobile Computing

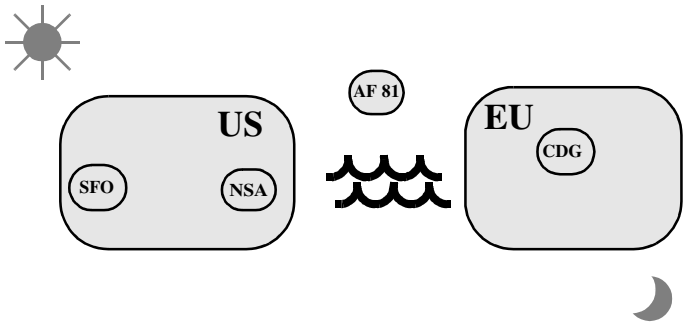
A different global computing paradigm has been evolving independently of the Web. Instead of connecting together all the LANs in the world, another way of extending the reach of a LAN is to move individual computers and other gadgets from one LAN to another, dynamically.

We discussed in the Introduction how the main characteristics of the Web point towards mobile comput*ation*. However, that is meant as mobile computation over a fixed (although possibly flaky) network. A more interesting picture emerges when the very components of the network can move about. This is the field of mobile comput*ing*. Today, laptops and personal organizers routinely move about; in the future entire networks will go mobile (as in IBM's Personal Area Network). Existing examples of this kind of mobility include: a smart card entering a network computer slot; an active badge entering a room; a wireless PDA or laptop entering a building; a mobile phone entering a phone cell.

We could draw a picture similar to Mental Image 1, but with mobile devices moving within the confines of a single LAN. This notion of a dynamic LAN is a fairly minor extension of the basic LAN concepts, and presents few conceptual problems (wireless LANs are already common). A much more interesting picture emerges when we think of mobile gadgets over a WAN, because administrative boundaries and multiple access pathways then interact in complex ways, as anybody who travels with a laptop knows all too well.

Mental Image 3 focuses on two domains: the United States and the European Union, each enclosed by a political boundary that regulates the movement of people and computers. Within a political boundary, private companies and public agencies may further regulate the flow of people and devices across their doors. Over the Atlantic we see a third

domain, representing Air France flight 81 travelling from San Francisco to Paris. AF81 is a very active mobile computational environment: it is full of people working with their laptops and possibly connecting to the Internet through airphones. (Not to mention the hundreds of computers that control the airplane and let it communicate with a varying stream of ground stations.)



Mental Image 3: Mobile Computing

Abstracting a bit from people and computation devices, we see here a hierarchy of boundaries that enforce controls and require permissions for crossing. Passports are required to cross political boundaries, tickets are required for airplanes, and special clearances are required to enter (and exit!) agencies such as the NSA. Sometimes, whole mobile boundaries cross in and out of other boundaries and similarly need permissions, as the mobile environment of AF81 needs permission to enter an airspace. On the other hand, once an entity has been allowed across a boundary, it is fairly free to roam within the confines of the boundary, until another boundary needs to be crossed.

2.4 General Mobility

We have described two different notions of mobility. The first, *mobile computation*, has to do with virtual mobility (mobile software). The second, *mobile computing*, has to do with physical mobility (mobile hardware). These two fields are today almost disconnected, the first dominated by a software community, and the second dominated by a hardware community. However, the borders between virtual and physical mobility are fuzzy, and eventually we will have to treat all kinds of mobility in a uniform way. Here are two examples where the different forms of mobility interact.

The first example is one of virtual mobility achieved by physical means. Consider a software agent in a laptop. The agent can move by propagating over the network, but can also move by being physically transported with the laptop from one location to another. In the first case, the agent may have to undergo security checks (e.g., bytecode verification) when it crosses administrative domains. In the second case the agent may have to undergo security checks (e.g., virus detection) when the laptop is physically allowed inside a new administrative domain. Do we need two completely separate security infrastructures for

these two cases, or can we somehow find a common principle? A plausible security policy for a given domain would be that a physical barrier (a building door) should provide the same security guarantees as a virtual barrier (a firewall).

The second example is one of physical mobility achieved by virtual means. Software exists that allows remote control of a computer, by bringing the screen of a remote computer on a local screen. The providers of such software may claim that this is just as good as moving the computer physically, e.g. to access its local data. Moreover, if the remote computer has a network connection, this is also equivalent to "stringing wire" from the remote location, since the remote network is now locally accessible. For example, using remote control over a phone line to connect from home to work where a high-bandwidth Internet connection is available, is almost as good as having a high-bandwidth Internet connection brought into the home.

The other side of the coin of being mobile is of becoming disconnected or intermittently connected. Even barring flaky networks, intermittent connectivity can be caused by physical movement, for example when a wireless user moves into some form of Faraday cage. More interestingly, intermittent connectivity may be caused by virtual movement, for example when an agent moves in and out of an administrative domain that does not allow communication. Neither case is really a failure of the infrastructure; in both cases, lack of connectivity may in fact be a desirable security feature. Therefore, we have to assume that intermittent connectivity, caused equivalently by physical or virtual means, is an essential feature of mobility.

In the future we should be prepared to see increased interactions between virtual and physical mobility, and we should develop frameworks where we can discuss and manipulate these interactions.

2.5 Barriers and Action-at-a-Distance

The unifying difficulty in both mobile computing and mobile computation is the proliferation of barriers, and the problems involved in crossing them. This central difficulty implies that we must regard barriers as fundamental features of our computational models. This seems contrary to the usual trend.

Access barriers have arisen many times in the history of computing, and one of the main tasks of computer science has been to "abstract them away", often by the proverbial additional level of indirection. For example, physical memory boundaries are circumvented by virtual memory; address space boundaries are circumvented by network proxies; firewall boundaries are circumvented by secure tunnels and agent sandboxing. Unfortunately, when barriers are not purely technological it is not possible to completely abstract them away. The crossing of administrative barriers must be performed by bureaucratic operations, such as exhibiting equipment removal passes and export licences.

Therefore, administrative barriers constitute a fundamental change to the way we compute. Let's review some historical scenarios that, because of barriers, have now become unrealizable computing utopias.

In the early days of the Internet, any computer could talk to any other computer by knowing its IP number. We can now forget about flat IP addressing and transparent routing: routers and firewalls effectively hide certain IP addresses from view and make them unreachable by direct means.

In the early days of programming languages, people envisioned a universal address space in which all programs would live and share data, possibly with world-wide garbage-collection, and possibly with strong typing to guarantee the integrity of pointers. We can now forget about universal addressing: although pointers are allowed across machines on a LAN (by network proxies), they are generally disallowed across firewalls. Similarly, we can forget about transparent distributed object systems: some network objects will be kept well hidden within certain domains, and reaching them will require effort.

In the early days of mobile agents, people envisioned agents moving freely across the network on behalf of their owners. We can now forget about this kind of free-roaming. If sites do not trust agents they will not allow them in. If agents do not trust sites to execute them fairly, they will not want to visit them.

In general, we can forget about the notion of *action-at-a-distance computing*: the idea that resources are available transparently at any time, no matter how far away. Instead, we have to get used to the notion that movement and communication are step-by-step activities, and that they are visibly so: the multiple steps involved cannot be hidden, collapsed, or rendered atomic.

The action-at-a-distance paradigm is still prevalent within LANs, and this is another reason why LANs are different from WANs, where such an assumption cannot hold.

2.6 Why a WAN is not a big LAN

We have already discussed in the Introduction how a WAN exhibits a different set of observables than a LAN. But could one emulate a LAN on top of a WAN, restoring a more familiar set of observables, and therefore a more familiar set of programming techniques? If this were possible, we could then go on and program the Internet just like we now program a LAN.

To turn a WAN into a LAN we would have to hide the new observables that a WAN introduces, and we would have to reveal the observables that a WAN hides. These tasks ranges from difficult, to intolerable, to impossible. Referring to the classification in the Introduction, we would have to achieve the following.

(A) Hiding virtual locations. We would have to devise a security infrastructure that makes navigation across multiple administrative domains painless and transparent (when legitimate). Although a great deal of cryptographic technology is available, there might be impossibility results lurking in some corners. For example, it is far from clear whether one can in principle guarantee the integrity of mobile computations against hostile or unfair servers [12]. (This can be solved on a LAN by having all computers under physical supervision.)

(B) Hiding physical locations. One cannot "hide" the speed of light; techniques such as caching and replication may help, but they cannot fool processes that attempt to perform long-distance real-time control and interaction. In principle, one could make all delays uniform, so that programs would not behave differently in different places. Ultimately this can be achieved only by slowing down the entire infrastructure, by embedding the maximal propagation delay in all communications. (This would be about 1/10 of a second on the surface, but would grow dramatically as the Web is extended to satellite communication, orbital stations, and further away.)

(C) Hiding bandwidth fluctuations. It is possible to introduce service guarantees in the networking infrastructure, and therefore eliminate bandwidth fluctuations, or reduce them below certain thresholds. However, in overload situations this has the only effect of turning network congestion into access failures, which brings us to the next point.

(D) Revealing failures. We would have to make failures as observable as on a LAN. This is where we run into fundamental trouble. A basic result in distributed systems states that we cannot achieve distributed consensus (such as agreeing on which nodes have failed) in a system consisting of a collection of asynchronous processes [10]. The Web is such a system: we can make no assumption about the relative speed of processors (they may be overloaded, or temporarily disconnected), about the speed of communication (the network may be congested or partitioned), about the order of arrival of messages, or even about the number of processes involved in a computation. In these circumstances, it is impossible to detect the failure of processors or of network nodes or links: any consensus algorithm can be delayed indefinitely. The common partial solutions for this unsolvable problem are to dictate some degree of synchrony and failure detection. These solutions work well on a LAN, but they seem unlikely to apply to WANs simply because individual users may arbitrarily decide to turn off their processors without warning, or take them into unreachable places. Other partial solutions involve multiple-round broadcast-based probabilistic algorithms [2] which might be expensive on a WAN in terms of communication load, and would be subject to light-speed delays. Moreover, it is difficult to talk about the failure of processors that are invisible because they are hidden behind firewalls, and yet take part in computations. Therefore, it seems unlikely that techniques developed to deal with asynchrony in operating systems and LANs can be successfully applied to a WAN such as the Web in full generality. The Web is an inherently asynchronous system, and the impossibility result of [10] applies with full force.

In summary: task (A) may be unsolvable for mobile code; in any case, a non-zero amount of bureaucracy will always be required; task (B) is only solvable (in full) by introducing unacceptable delays; task (C) can be solved in a way that reduces it to (D); task (D) is unsolvable in principle, and probabilistic solutions run into tasks (A) and (B).

2.7 WAN Postulates

We summarize this section by a collection of postulates that capture the main properties of the reality we are interested in modeling:

- *Separate locations exist.*
- *Different locations have different properties, hence both people and programs will want to move between them.*
- *Barriers to mobility will be erected to preserve the properties of certain locations.*
- *Some people and some programs will still need to cross those barriers.*

The point of these postulates is to stress that mobility and barrier crossing are inevitable requirements of our current and future computing infrastructure.

The observables that are characteristic of wide area networks have the following implications:

- Distinct virtual locations are observed because of the existence of distinct administrative domains, which are produced by the inevitable existence of attackers. Distinct virtual locations preclude the unfettered execution of actions across domains, and require a security model.
- Distinct physical locations are observed, over large distances, because of the inevitable latency limit given by the speed of light. Distinct physical locations preclude instantaneous action at a distance, and require a mobility model.
- Bandwidth fluctuations (including hidden failures) are observed because of the inevitable exercise of free will by network users, both in terms of communication and movement. Bandwidth fluctuations preclude reliance on response time, and require an asynchronous communication model.

3 Modeling Wide Area Computation

Section 2 was dedicated to showing that the reality of mobile computation over a WAN does not fall into familiar categories. Therefore, we need to invent a new model that can help us in understanding and eventually in taking advantage of this reality.

3.1 Barriers

We believe that the most fundamental new notion is that of barriers; this is the most prominent aspect of post-LAN computing environments.

Many of the basic features of WANs have to do with barriers: ***Locality*** (the existence of different virtual or physical locations, and the notion of being in the same or different locations) is induced by a topology of barriers. ***Mobility*** is barrier crossing. ***Security*** has to do with the ability or inability to cross barriers. ***Communication*** is partitioned by barriers: local communication happens within barriers, while long-distance communication is a combination of local communication and movement across barriers. ***Action at a distance*** (immediate interaction across many barriers) is forbidden.

We have chose barriers as the most important feature of an abstract model of computation for wide area networks, the Ambient Calculus [7], which we briefly outline.

3.2 Ambients

The current literature on wide area network languages can be broadly classified into agent-based languages (e.g., Telescript [14]), and place-based languages (e.g., Linda [8]). An ambient is a generalization of both notions. Like an agent, an ambient can move across places (also represented by ambients) where it can interact with other agents. Like a place, an ambient supports local undirected communication, and can receive messages (also represented by ambients) from other places. Ambients can be arbitrarily nested, generalizing the limited place-agent-data nesting of most agent languages, and the nesting of places allowed in some Linda dialects.

Briefly, an *ambient* is a place that is delimited by a boundary and where multi-threaded computation happens. Each ambient has a *name*, a collection of local *processes*, and a collection of *subambients*. Ambients can move in and out of other ambients, subject to *capabilities* that are associated with ambient names. Ambient names are unforgeable, this fact being the most basic security property.

In further detail, an ambient has the following main characteristics.

- An ambient is a *bounded* place where computation happens.

  If we want to move computations easily we must be able to determine what parts should move. A boundary determines what is inside and what is outside an ambient, and therefore determines what moves. A boundary implies some flexible addressing scheme that can denote entities across the boundary; examples are symbolic links, URLs (Uniform Resource Locators) and Remote Procedure Call proxies. Flexible addressing is what enables, or at least facilitates, mobility. It is also, of course, a cause of problems when the addressing links are “broken”.

- Ambients can be nested within other ambients, forming a tree structure.

  As we discussed, administrative domains are (often) organized hierarchically. Mobility is represented as navigation across a hierarchy of ambients. For example, if we want to move a running application from work to home, the application must be removed from an enclosing (work) ambient and inserted in a different enclosing (home) ambient.

- Each ambient has a collection of local running processes.

  A local process of an ambient is one that is contained in the ambient but not in any of its subambients. These “top level” local processes have direct control of the ambient, and in particular they can instruct the ambient to move. In contrast, the local processes of a subambient have no direct control on the parent ambient: this helps guaranteeing the integrity of the parent.

- Each ambient moves as a whole with all its subcomponents.

  The activity of a single local process may, by causing movement of its parent, influence the location, and therefore the activity, of other local processes and subambients. For example, if we move a laptop and reconnect it to a different network, then

all the threads, address spaces, and file systems within it move accordingly and automatically, and have to cope with their new surrounding. Agent mobility is a special case of ambient mobility, since agents are usually single-threaded. Ambients, like agents, automatically carry with them a collection of private data as they move.

- Each ambient has a name.

  The name of an ambient is used to control access (entry, exit, communication, etc.). In a realistic situation the true name of an ambient would be guarded very closely, and only specific capabilities based on the name would be handed out.

3.3 Ideas for Wide Area Languages

Ambients represent our understanding of the fundamental properties of mobile computation over wide area networks. Our final goal, though, is to program the Internet in some convenient high-level language. Therefore, we aim to find programming constructs that are semantically compatible with the ambient principles, and consequently with wide area networks.

These compatibility requirements include (A) *WAN-soundness*: a wide area network language cannot adopt primitives that entail action-at-a-distance, continued connectivity, global consensus, or security bypasses, and (B) *WAN-completeness*: a wide area network language must be able to express the behavior of web surfers and of mobile agents and users, and of any other entities that routinely roam those networks.

More specifically, we believe the following are necessary ingredients of wide area languages.

- ***Naming***. Names are symbolic ways of referring to entities across a barrier. Names are detached from their corresponding entities; one may possess a name without having immediate access to any entity of that name. To enable mobility and disconnected operation, all entities across a barrier should be denoted by names, not by "hard" pointers.
- ***Migration***. Active hardware and software components should be able to migrate. Migration of certain active hardware components is possible today, but the ability to automatically disconnect and reconnect those components to surrounding (possibly multiple) networks is not currently available. Migration of active software components is even harder, typically for lack of system facilities for migrating live individual threads and groups of threads.
- ***Dynamic connectivity***. A wide area network cannot be started or stopped all at once. Therefore, it is necessary to dynamically connect components. This is contrary to the current prominence in programming languages of static binding, static module composition, and static linking. The ambient calculus provides an example of a novel mixture of ordinary static scoping of names (which enables typechecking) with dynamic binding of operations to names (which enables dynamic linking).

- ***Communication***. Communication on wide area networks must in general be asynchronous. However, local communication (within or even across a single barrier) can usefully be synchronous. Moreover, in the presence of mobility, it is necessary to have some level of synchronization between communication and movement operations. This remains an interesting design area for mobile languages.
- ***Security***. Security abstractions should be provided at the programming-language level, that is, above the fundamental cryptographic primitives. Programmers need to operate with reliable high-level abstractions, otherwise subtle security loopholes can creep in. We believe that barriers are one such high-level security abstraction, which can be supported by programming constructs that can be mechanically analyzed (e.g., via type systems [6]).

Summary

The ambient semantics naturally suggests unusual programming constructs that are well-suited for wide area computation. The combination of mobility, security, communication, and dynamic binding issues has not been widely explored yet at the language-design level, and certainly not within a unifying semantic paradigm. We hope our unifying foundation will facilitate the design of such new languages.

3.4 Wide Area Challenge: A Conference Reviewing System

We conclude with the outline of an ambitious wide area application. The application described here does not fit well with simple-minded Web-based technology because of the complex flow of active code and stateful information between different sites, and because of an essential requirement for disconnected operation. The application fits well within the agent paradigm, but also involves the traversal of multiple administrative domains, and has security and confidentiality requirements.

This is meant both as an example of an application that could be programmed in a wide area language, and as a challenge for any such language to demonstrate its usability. We hope that a language based on ambients or similar notions would cope well with this kind of situation.

- *Description of the problem.* The problem consists in managing a virtual program committee meeting for a conference. The basic architecture was suggested to me by comments by Richard Connors, as well as by my own experience with organizing program committee meetings and with using Web-based reviewing software developed for ECOOP and other conferences.

In the following scenario, the first occurrence of each of the principals involved is shown in boldface.

- *Announcement.* A **conference** is announced, and an electronic **submission form**, signed by the **conference chair**, is publicized.

- *Submission.* Each **author** fetches the submission form, checks the signature of the conference chair, and activates the form. Once activated, the form actively guides most of the reviewing process. Each author fills an instance of the form and attaches a **paper**. The form

checks that none of the required fields are left blank, electronically signs the paper with a signature key provided by the author, encrypts the attached paper, and finds its way to the **program chair**. The program chair collects the submissions forms, and gives them a decryption key so that they can decrypt the attached papers and verify the signatures of the authors. (All following communications are signed and encrypted; we omit most of these details from now on.)

- Assignment. The program chair then assigns the submissions to the **committee members**, by instructing each submission form to generate a **review forms** for each assigned member. The review forms incorporate the paper (this time signed by the program chair) and find their way to the appropriate committee members.

- Review. Each committee member is a **reviewer**, and may decide to review the paper directly, or to send it to another reviewer. The review form keeps tracks of the chain of reviewers so that it can find its way back when either completed or refused, and so that each reviewer can check the work of the subreviewers. Eventually a review is filled. The form performs various consistency checks, such as verifying that the assigned scores are in range and that no required fields are left blank. Then it finds its way back to the program chair.

- Report generation. Once the review forms reach the program chair, they become **report forms**. The various report forms for each paper merge with each other incrementally to form a single report form that accumulates the scores and the reviews. The program chair monitors the report form for each paper. If the reviews are in agreement, the program chair declares the form an **accepted paper report form**, or a **rejected paper review form**.

- Conflict resolution. If the reports are in disagreement, the program chair declares the form an **unresolved review form**. An unresolved review form circulates between the reviewers and the program chair, accumulating further comments, until the program chair declares the paper accepted or rejected.

- Notification. The report form for an accepted or rejected paper finds its way back to the author (minus the confidential comments), with appropriate congratulations or regrets.

- Final versions. Once it reaches the author, an accepted paper report form spawns a **final submission form**. In due time, the author attaches to it the final version of the paper and signs the copyright release notice. The completed final submissions form finds its way back to the program chair.

- Proceedings. The final submission forms, upon reaching the program chair, merge themselves into the **proceedings**. The program chair checks that all the final versions have arrived, sorts them into a conference schedule, attaches a preface, and lets the proceedings find their way to the conference chair.

- Publication. The conference chair files the copyright release forms, signs the proceedings, and posts them to public sites.

In summary, in this example, interactions between various parts of the system happen over a wide area network. The people involved may be physically moving during or between interaction. As they move, they may transport without warning active parts of the system. At other times, active parts of the system move by their own initiative and must find a route to the appropriate principals wherever they are.

4 Conclusions

The global computational infrastructure has evolved in fundamental ways beyond standard notions of sequential, concurrent, and distributed computational models. The notion of *ambients* captures the structure and properties of wide area networks, of mobile computing, and of mobile computation. The ambient calculus [7] formalizes these notions simply and powerfully. It supports reasoning about mobility and security, and has an intuitive graphical presentation in terms of a folder calculus [3]. On this foundation, we can envision new programming methodologies, libraries and languages for wide area computation.

5 Acknowledgments

Andrew D. Gordon is a coauthor of several related papers.

References

[1] Bharat, K. and L. Cardelli: **Migratory applications**, *Proc. of the ACM Symposium on User Interface Software and Technology '95*. 133-142. 1995.

[2] Bracha, G. and S. Toueg, **Asynchronous consensus and broadcast protocols**. *J.ACM* **32**(4), 824-840. 1985.

[3] Cardelli, L., **Abstractions for Mobile Computation**, in *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, Jan Vitek and Christian Jensen (Eds.). Springer. 1999. (To appear.)

[4] Cardelli, L., **A language with distributed scope**. *Computing Systems,* **8**(1), 27-59. MIT Press. 1995.

[5] Cardelli, L. and R. Davies. **Service combinators for web computing.** *Proc. of the First Usenix Conference on Domain Specific Languages, Santa Barbara*. 1997.

[6] Cardelli, L., G. Ghelli, and A.D. Gordon, **Mobility Types for Mobile Ambients**, *Proc. ICALP'99*.

[7] Cardelli, L. and A.D. Gordon, **Mobile ambients**, in *Foundations of Software Science and Computational Structures*, Maurice Nivat (Ed.), LNCS 1378, Springer, 140-155. 1998.

[8] Carriero, N. and D. Gelernter, **Linda in Context**. *Communications of the ACM*, **32**(4), 444-458. 1989.

[9] Chandra, T.D., S.Toueg, **Unreliable failure detectors for asynchronous systems.** *ACM Symposium on Principles of Distributed Computing*, 325-340. 1991.

[10] Fischer, M.J., N.A. Lynch, and M.S. Paterson, **Impossibility of distributed consensus with one faulty process.** *J.ACM* **32**(2), 374-382. 1985.

[11] Milner, R., J. Parrow and D. Walker, **A calculus of mobile processes, Parts 1-2**. *Information and Computation*, **100**(1), 1-77. 1992

[12] Sander, A. and C. F. Tschudin, **Towards mobile cryptography**, *ICSI technical report 97-049*, November 1997. *Proc. IEEE Symposium on Security and Privacy*, Spring 1998.

[13] Stamos, J.W. and D.K. Gifford, **Remote evaluation**. *ACM Transactions on Programming Languages and Systems* **12**(4), 537-565. 1990.

[14] White, J.E., **Mobile agents**. In *Software Agents*, J. Bradshaw, ed. The MIT Press. 1996.