Report Number: WUCS-99-31

1999-01-01

Software Engineering for Mobility: A Roadmap

Gruia-Catalin Roman, Gian Pietro Picco, and Amy L. Murphy

The term distributed computing conjures the image of a fixed network structure whose nodes support the execution of processes that communicate with each other via messages traveling along links. Peer-to-peer communication is feasible but client-server relationships dominate. More recently, servers have been augmented with brokerage capabilities to facilitate discovery of available services. Stability is the ideal mode of operation; changes are relatively slow; even in the case of failure, nodes and links are expected eventually to come back up. By contrast, mobility represents a total meltdown of all the stability assumptions (explicit or implicit) associated with distributed computing. The... **Read complete abstract on page 2.**

Software Engineering for Mobility: A Roadmap

Gruia-Catalin Roman, Gian Pietro Picco, and Amy L. Murphy

Complete Abstract:

The term distributed computing conjures the image of a fixed network structure whose nodes support the execution of processes that communicate with each other via messages traveling along links. Peer-to-peer communication is feasible but client-server relationships dominate. More recently, servers have been augmented with brokerage capabilities to facilitate discovery of available services. Stability is the ideal mode of operation; changes are relatively slow; even in the case of failure, nodes and links are expected eventually to come back up. By contrast, mobility represents a total meltdown of all the stability assumptions (explicit or implicit) associated with distributed computing. The network structure is no longer fixed, nodes may come and go, processes may move along nodes, and even programs (the code executed by processes) may evolve and change structure. The challenges and opportunities associated with this computational melee form the main subject of this paper. We seek to sort out this chaotic form of computing by focusing our attention on the formulation of a simple framework for vieweing mobility, on precise definition of terms, and on research issues mobility poses for the software engineering community.

Software Engineering for Mobility: A Roadmap

Gruia-Catalin Roman
Gian Pietro Picco
Amy L. Murphy

WUCS-99-31

April 28, 2000

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Software Engineering for Mobility: A Roadmap

Gruia-Catalin Roman[1], Gian Pietro Picco[2], Amy L. Murphy[1]

[1]Department of Computer Science
Washington University in St. Louis
Campus Box 1045, One Brookings Drive
St. Louis, MO 63130-4899, USA
{roman,alm}@cs.wustl.edu

[2]Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
picco@elet.polimi.it

Abstract
The term distributed computing conjures the image of a fixed network structure whose nodes support the execution of processes that communicate with each other via messages traveling along links. Peer-to-peer communication is feasible but client-server relationships dominate. More recently, servers have been augmented with brokerage capabilities to facilitate discovery of available services. Stability is the ideal mode of operation; changes are relatively slow; even in the case of failure, nodes and links are expected eventually to come back up. By contrast, mobility represents a total meltdown of all the stability assumptions (explicit or implicit) associated with distributed computing. The network structure is no longer fixed, nodes may come and go, processes may move among nodes, and even programs (the code executed by processes) may evolve and change structure. The challenges and opportunities associated with this computational melee form the main subject of this paper. We seek to sort out this chaotic form of computing by focusing our attention on the formulation of a simple framework for viewing mobility, on precise definition of terms, and on research issues mobility poses for the software engineering community.

1 Introduction

Software engineering focuses on the study of software artifacts, people as (imperfect) producers of software, and processes as (approximate) guarantors of software quality. Yet, the best way to start understanding mobility is to leave behind this software-centric perspective and to ask a broader question: What is happening today with computing in the society at large? It is not at all difficult to see that computing is drifting away from computers. We have been trained to place software at the center of the computing field and at the heart of the systems we build. In reality, a strong centrifugal force is driving computing (along with its hardware and software structures) closer and closer towards the surrounding fabric of the society, its infrastructures, its enterprises, and its people. Increasingly, computing is being taken for granted in the same manner that we ignore electricity, the complexities of television broadcast technology, and the presence of motors throughout automobiles today—they are there, somewhere, but we no longer think of them. Commodity computers are likely to end up close to our bodies and in our clothing. Smart sensors will be found in elements of building structures or car parts. The distinction between electronic and traditional enterprises is already blurred. A dynamic society promotes and assimilates likewise computing structures. A society on the move demands computing structures that are mobile, malleable and available at any location. New kinds of networking architectures are emerging.

In the traditional static network, fixed hosts with statically assigned (IP) addresses exchange messages via the standard Internet infrastructure of fixed routers and switches. At the periphery of this fixed network one can envision base stations (fixed routers with wireless communication capabilities) that control message traffic to and from mobile hosts forming a dynamic fringe. Some of them may have fixed addresses while others may acquire temporary addresses as the need arises. Ultimately, mobile hosts can detach themselves completely from the fixed infrastructure and may evolve independently of it. Such clouds are called *mobile ad hoc networks.* They are opportunistically formed structures that change rapidly in response to the movement of the carriers to which the mobile hosts are attached. For the time being, communication within ad hoc networks is point to point over a physical broadcast medium, the airwaves. However, growth in performance and capabilities (e.g., new protocols) eventually will allow some of the mobile units to start serving as mobile routers for others in the area. Through transitivity, routing in ad hoc networks will expand the connectivity pattern beyond the limits of an immediately accessible region.

Across this highly dynamic physical structure a logical layer still more fluid is emerging. Code mobility removes the static binding between the software components of a distributed application and the network hosts where they are executing. Relocation of such components is then enabled, to achieve a higher degree of flexibility and customizability, or improved bandwidth utilization. More radical views encompass migration of execution units which become regarded as mobile agents autonomously performing tasks without requiring permanent connectivity towards the client. Although the interest in code mobility is being popularized and amplified by the success of the Java language, the relevance of mobile code is not only technological. This form of logical mobility has the potential to completely change the way distributed applications are conceived and deployed, by raising the location of components to the status of a first class design concept.

Sorting out the software engineering implications of this intriguing and perplexing wave of technological changes is a nontrivial task. Without any pretense of being comprehensive in its coverage, this paper identifies important research areas and puts forth a conceptual framework for thinking about mobility. In our view, *space* and *coordination* are the two most critical dimensions to be considered in any systematic treatment of mobility. They provide the basis for drawing the distinction between distributed and mobile computing and for differentiating among various perspectives on mobile computing.

The remainder of the paper is organized as follows. Section 2 explores the notion that variations in the definition of space and in the choice of coordination mechanisms can offer meaningful insights in the way different systems treat mobility. Section 3 is concerned with theoretical aspects of software engineering. They include formal models, which seek to uncover fundamental principles and essential features, and algorithms, which need to be revisited in the context provided by mobility. Section 4 is divided among applications and middleware. Applications are important because it is ultimately the end user that will determine the success and failure of the various technologies. Middleware is the area likely to see the greatest level of research activity as the demand for rapid deployment of dependable new applications increases. Each subsection examines relevant trends and identifies key research areas. Conclusions appear in Section 5.

2 Mobile Computing

A world of abundant, untethered, portable (even wearable), and unobtrusive computers is made possible only by the unique combination of two powerful trends: rapid component miniaturization and the emergence of high-speed wireless communication. As the number of components per unit of space grows (eventually reaching into the hundreds) running wires is no longer feasible. The same is true when components are carried around by human beings from one setting to another or when they reside on moving platforms such as cars and airplanes. Wireless communication (be it radio or infrared) is the only viable link among mobile components which aggregate together to form complex structures mostly due to proximity and shared functional needs or connect seamlessly to the wireline networks as they change location. The communication industry is actively pursuing these opportunities by investing in new wireless technologies (e.g., wireless LAN speeds in excess of 10Mbps), by co-operating in the establishment of interoperability standards (e.g., IEEE 802.11b High Rate standard), and by forming powerful consortia. The IETF Mobile Ad Hoc Networks (Manet) Working Group is considering standardization efforts based on IP technology. The list of consortia includes: 3G.IP which focuses on high bandwidth wireless technology using W-CDMA; Bluetooth which uses frequency hopping and is designed to provide low-cost support for small groups of co-located devices; and others which promise home networks (via a Shared Wireless Access Protocol or SWAP) or Web page delivery to low-bandwidth devices (via a Wireless Application Protocol or WAP).

Of course, wireless communication is not the same thing as delivery of data to a mobile unit. The latter presupposes the ability to find the current location of the unit and to continue to send data as the unit moves from one place to another. Cellular phone systems accomplish this through a combination of broadcasts (to notify the unit about the incoming call) and hand-off protocols (to maintain the connection in the face of movement). In the Internet setting, special protocols, such as Mobile IP, have been designed to enable packet delivery while a mobile unit is away from its home base. The next version of IP (IPv6) is anticipated to provide still better support for transparent packet delivery to mobile units away from their home networks. Efforts are also under way to respond to the special needs of ad hoc networks. Rapidly changing topology renders impractical many well-established routing strategies such as link-state and distance-vector. New variants are being proposed and evaluated. They include Temporally Ordered Routing Algorithm (TORA) [35], Dynamic Source Routing (DSR) [7], and Ad hoc On demand Distance Vector (AODV) [36]. A common feature among all three is their reactive nature, i.e., routing information is built in response to the demand to communicate among specific hosts. The provision of multicast services is still another area receiving much attention in the mobile setting.

Even though one might expect to see the concern with mobility reach into the next layer of the computing host architecture, the operating system, this does not seem to be the case at this time. The communication substrate, in general, continues to be tightly integrated in the fabric of the operating system services without actually affecting its design in any significant way. Actually, the impact of mobility on systems research and development is manifest mostly at the language and middleware levels. Languages are likely to continue to be dominated by the concerns of code mobility. With the emergence of Java and its well-integrated code-on-demand capabilities, both mobile code languages (e.g., Obliq) and mobile agent languages (e.g., Agent Tcl) are losing ground even as mobility in all its forms is growing in significance. Middleware is emerging as one of the most fertile areas of systems research in mobility. This is, in part, because specialized languages have lost favor with practitioners and researchers alike. New languages require too great an investment and entail unacceptable risks while middleware can take advantage of the deployed software infrastructure while providing clean high-level programming abstractions in languages already available today. Middleware hides the protocol layer but makes explicit the key concepts involved in the development of mobile applications, e.g., the management of location data, event notification, quality of service assessment, adaptability, etc. Middleware can be specialized for logical or physical mobility or may combine the two in a single cohesive package. Ultimately, research on protocols, operating systems and languages for mobility will not represent a major concern for the software engineering community. It is the concern with models, algorithms, applications and middleware that will dominate software engineering research on mobility in the decade to come.

From a software engineering perspective, *we view mobile computing to be the study of systems in which computational components may change location.* Suitable locations are points in a space that may be continuous or discrete. At a very coarse level, spaces can be of two types: physical and logical. Physical mobility entails the movement of mobile hosts in a building or even large regions of the earth, i.e., a subset of the physical space we occupy on the planet and beyond. Logical mobility involves mobile units (of code and state) that migrate among hosts. Typically, the hosts are stationary and the mobility space reflects directly the structure of the underlying network. It should be immediately apparent that this simple distinction may be an accurate characterization of past developments but that it is woefully inadequate of current trends, which are likely to allow for mobile units to migrate among both mobile and stationary hosts. This is why a careful characterization of the mobility space and of the patterns of movement that it permits is important.

Prevailing terminology in mobile computing is tied to specific, albeit successful, types of applications/configurations that do not reflect the potential richness of the field. In the case of physical mobility, for instance, we often distinguish between nomadic computing and ad hoc networks. The former is a term that denotes systems consisting of a fixed core network and a fringe of mobile hosts that connect to it via base stations. Here, space may be used to distinguish between the cellular structure of the phone networks and the kind of systems that rely on wireless LANs. Ad hoc networks refers to systems consisting of mobile hosts exclusively. Hosts are connected to each other when they are within communication range. An individual mobile host may or may not act as an ad hoc router for the benefit of its neighbors. A more refined characterization of such systems is needed. Moving along train tracks, interacting within the confines of a single room, and roaming across the sky lead to very different classes of ad hoc networks. It is the definition of space that can provide clean and useful formalizations of such subclasses. In general, as the technology matures and specialized applications emerge one should expect to see mobility spaces with increasingly complex structures. Movement in multidimensional spaces may turn out to be a useful concept, e.g., administrative domains may be viewed as another spatial dimension, orthogonal to the geographic coordinate system.

Once we consider the notion that programs can move through physical and logical spaces, it is only natural to start wondering what happens when they meet. How do they recognize each other as friends or foes? How do they exchange information? How do they construct complex cooperative behaviors? These and other similar questions are all about coordination. Coordination is important in mobile computing because of the need to decouple, both conceptually and pragmatically, the treatment of the individual components from the manner in which they interact with each other. By focusing on coordination one can limit the extent to which components need to be aware of each other, especially when one realizes that they may never know in advance with whom and where interactions will take place. This is because coordination brings about a world view that is centered, metaphorically speaking, on the social dynamics rather than the individual personalities. If much of the research on concurrency looked at components (i.e., processes) from inside out, coordination seeks to view components (i.e., mobile units) from outside in. In one case we ask the question how shared variables are accessed while in the other case we ponder about how variables are shared. Coordination is concerned primarily with the mechanisms (usually supplied by the middleware or the operating system) needed to discover who is around, to exchange information, to synchronize actions, etc. This is why the manner in which mobile components interact with each other becomes an important differentiating feature among systems that support mobility.

Let us consider, for instance, the problem of discovering who else is there. One solution is to provide some sort of known registry that can be interrogated, e.g., a name server. Another approach is to rely on passing around acquaintances, i.e., one becomes aware of the presence of others through the grapevine. Broadcast may be used to announce one's presence, if there is agreement on the broadcast protocol to be used. In all cases, some a priori agreement is needed—often called metalevel communication. In mobile computing, the question becomes how to minimize such common knowledge in order to maximize the open character of the systems we build. Message passing protocols traditionally provided the standard interface among units when wireless communication is involved. Shared variables (or objects) are frequently present in mobile agent systems. Recently, shared tuple spaces, in a variety of forms, are gaining in popularity. The trend towards more abstract coordination mechanisms is a very positive development because it holds the promise for facilitating rapid and dependable development of mobile applications.

Another coordination concern, synchronization, may assume a variety of forms, from the familiar to the exotic. All types of statement synchronization may be encountered in mobile systems. New forms of synchronization involve the notion of location. For instance, a group of robots may start congregating together due to a predilection to migrate towards one another. Similarly, components may seek to move as a group by preserving connectivity, i.e., by staying within communication range. One can also envision diffused forms of synchronization in which aggregate information is used to decide on direction of movement. This would be the case when components may be biased to migrate towards areas of the space where a lot of communication is taking place or where the signal is stronger without actually making contact with any specific component in the range. Of course, coordinated movement can be accomplished by taking advantage of the properties of space and relative speeds. Two hosts moving along parallel trajectories at constant speed will remain equidistant, for instance.

Coordination mechanisms can also be classified as being explicit or implicit. The former happens when one component refers to another for the purpose of sending a message across. The latter form is encountered when the underlying system makes the coordination transparent to the components involved. A change in context or the arrival of an event may be the result of transparent coordination activities perceived by the unit as spontaneous changes in the programming context. One particularly interesting mechanism for effecting a change in context may be the result of code movement. The unit may not be aware that any coordination took place, yet its behavior may be altered dramatically. Variations in the granularity of the code being shipped around deserve to be investigated as well.

Implicit in much of the discussion above is the notion that coordination among mobile hosts or units takes place whenever they are in touch with each other. However, the notion of being in touch entails careful analysis. One may be tempted to assume that components can coordinate with each other when they are co-located. Two agents may arrive on the same server or two hosts may find themselves in the same vicinity (on the same wireless LAN). Unfortunately, such a view ignores considerations having to do with security, administrative domains, or quality of service. A weak radio link with high error rates may need to be ignored; a good connection that cannot deliver the desired bandwidth for a multimedia application may not be of any use either. The basic concept of co-location is right but its definition is complex. There is a need to allow for flexible specification of the conditions under which co-location actually permits coordination to take place.

Since co-location is a binary relation, it is natural to raise the issue of whether the relation is symmetric and/or transitive. For most researchers, communication is two-way because the kinds of protocols we employ today, e.g., the need for acknowledgements and negotiation. Asymmetry, however, is frequently manifest in the bandwidth disparity associated with the forward and reverse communication channels. In a wireless setting, there are many situations in which asymmetric communication is reasonable, e.g., in nomadic systems where differences in transmitter power of the base station and the mobile unit are very pronounced or in ad hoc networks where the battery state may vary greatly among units in the field. Both transitive and non-transitive definitions of co-location make sense as well. For security reasons, mobile hosts interacting with the same base station may not be considered to be co-located. At the same time, networks that support ad hoc routing may be viewed as fully connected clusters of co-located components.

Finally, provisions for distant interactions may alter the picture even further either by annihilating space altogether or by forcing a clear differentiation among local and distant coordination mechanisms. Here again, we have an example in which the degree of novelty and the range of opportunities are most striking when we consider the interplay between coordination and space.

3 Theory research

Theoretical studies tend to focus on essential traits of broad classes of systems. This section explores possible definitions of mobile computing as a field of study and research by considering the range of models and algorithms that are beginning to be explored today. Models are concerned with the formulation of proper abstractions useful in specification and evaluation. They identify fundamental concepts and relationships that provide the elements of discourse for a particular scientific field and the basis for the development of analytical tools. As such, models provide valuable insights into how the field might evolve, what ideas are considered important, and what avenues remain unexplored. Algorithmic research centers on discovering difficult problems that are frequently encountered during design and on formulating and analyzing basic solutions to such problems. Algorithms offer designers trusted solutions to fundamental problems and design strategies that can be readily adapted to new settings. Another important aspect of this kind of research is gaining an understanding of what is and what is not possible and at what cost. As seen in the remainder of this section, research on models and algorithms for mobility is only in its formative stages, opening opportunities for fresh, new ideas.

Models

In this section, we focus our attention on models that entail an explicit notion of space and components that move through it. A component may be either a code fragment that is given the ability to roam the address spaces of a computer network or a physical device moving through the real world. Abstraction often blurs the distinction between logical and physical mobility thus allowing us to formally specify and reason about arbitrary components moving across a broad range of conceivable spaces. By and large, models tend to subsume physical into logical mobility, as the latter exhibits characteristics that have no direct physical counterparts, e.g., the ability to spawn remotely a new mobile unit. As one might expect, models vary greatly in the way they answer questions such as who is allowed to move, where it can go, and how context changes caused by movement are managed. The choice of unit of mobility is central to any model of mobility since it shapes to a large extent the way in which the other two questions are addressed. The treatment of location is indicative of the model's perception of space. The handling of contextual changes reflects the component's perception of the coordination mechanisms that tie components into a system. Ultimately, the assumptions and choices a model makes relative to these particular concerns differentiate it from other models of mobility.

The *unit of mobility* represents the smallest component in the system that is allowed to move. A typical choice is to make the unit of mobility coincide with the unit of execution. This approach fits well a mobile device that moves in physical space as well as a mobile agent that migrates among network hosts. The vast majority of models share this choice, e.g., higher-order extensions of π-calculus [43], Ambients [9], and Mobile UNITY [27], to name only a few. However, the reality of middleware and applications for logical mobility suggests that finer-grained units, weaker than full-fledged execution units, are pervasive in every day practice. Among various design paradigms for code mobility [14], for instance, code on demand is probably the most widely used at this time. In this style of logical mobility, the unit of execution does not actually move. Its behavior is dynamically augmented by foreign code that becomes linked when a particular trigger condition occurs. Evidently, this fine-grained perspective provides a new degree of freedom in describing how a distributed system gets reconfigured by exploiting mobility among its components. The unit of execution is no longer tied to a host and neither are the unit's constituents tied to it. From this perspective, the ability to move a unit of execution as a whole (commonly called a mobile agent) may be regarded as a special case of a more general framework in which single code fragments and/or their corresponding states can change location. Not surprisingly, this notion has a direct counterpart in physical mobility, where the *alter ego* of code and state are the applications and the data they use on some device.

So far, despite its theoretical and practical relevance, fine-grained mobility received only limited attention in the formal models community. A commonly used approach is to view the code and the state associated with an executing unit as degenerate cases of the unit, e.g., state may be carried by a unit in which the code is missing or has no effect on the computation. Because code and state are not treated as first-class units of mobility, this approach is not sufficiently expressive, e.g., it cannot capture code assemblies still under construction. To our knowledge, the only model that addresses fine-grained logical mobility explicitly is the one presented in [26]. In that work, this idea is pushed to an extreme by investigating a model where the unit of mobility is as small as a single variable or statement in a programming language. This radical perspective, readily encompassing more common situations where the unit of mobility is as coarse as a class or an object, is expected to provide new insights in the design of programming languages that foster high degrees of reconfigurability.

Location identifies the position of a mobile unit in space. This view of location is tied to the intuitive notion of mobility and distinguishes models of interest to us in this paper from other highly dynamic models that equate mobility with a more general notion of change. In π-calculus [30], for instance, there is no notion of location built into the model, and yet the structure of the system can change dynamically. Processes exchange communication channels (represented by names) and, in some extensions [43], even processes. This provides the expressive power needed to describe systems whose structure evolves but fails to treat location as a first class concept. It is important for a model to be capable of dealing with location throughout the software development lifecycle, starting from the definition of the environment where mobility occurs, through designing and reasoning about a mobile application, and down to the tools provided to programmers. For this reason, numerous researchers are investigating calculi [9, 13, 33, 4] which extend π-calculus with some notion of location and also approaches that are not based on process algebras but on state transitions and logic [27].

The type of location is affected by the choice of unit of mobility. For instance, location could be represented by Cartesian coordinates for a mobile device, by a host address for a mobile agent, or by a process identifier in the case of a code fragment. For this reason, some models avoid specifying the details of location altogether and focus on how to effect movement and on how to detect and handle location changes and their consequences. This is precisely the case of Mobile UNITY [41], where location is modeled explicitly as a distinguished variable that belongs to the state of a mobile component. Changes in its value correspond to changes in the position of the component. Other models start with different assumptions and impose a predefined structure on the space (typically hierarchical). Such is the case with MobiS [25] where locations are nested spaces containing tuples, which in turn may contain code as well as data with migration taking place upwards in the hierarchy of spaces. Ambients [9] provides a richer model where locations are ambients containing processes or other ambients. The boundary of an ambient, however, can be reconfigured dynamically to change the overall system structure. These latter approaches combine the notion of location, which only abstracts the notion of position in space, and the notion of context described below.

Context represents the peculiar and novel aspect of mobile computing, to the point that some researchers characterize mobility as "context-aware computing." The context of a mobile unit is determined by its current location which, in turn, defines the environment where the computation associated with the unit is performed. The context may include resources, services, as well as other components of the system. Conventional computing tends to foster a static notion of context, where changes are absent, small, or predictable. In a mobile setting, changes in location may lead to sudden changes in the context a unit perceives. Moreover, these changes are likely to be abrupt and unpredictable. A handheld wireless device carried across the floors of an office building has access to different resources (e.g., printers or directory information) on each floor; a mobile agent migrating on different servers may use different sets of services on each of them; in a fine-grained model, a statement with free identifiers may be bound to different variable instances each time it is linked into a different unit of execution.

Even though they are intimately related, location and context are fundamentally different notions. Two mobile units may be at the same location but perceive different contexts because they belong to different administrative domains. Similarly, two units may be at different locations and yet share the same context, e.g., two handheld devices in communication range. Failing to discriminate between location and context can limit the expressive power of a model and can lead to missed opportunities. As shown in the remainder of this section, many open research issues in mobility are tied into the notion of context. In mobile computing, precise formulation of the notion of context and of the mechanisms for inducing and managing context changes are important challenges facing software engineering formalists and practitioners alike.

The context seen by each unit is determined by the kind of coordination mechanisms that are supported by the model. Coordination [24], as an area of research, investigates models and languages that separate the specification of the behavior of the individual processes from the communication needed to coordinate such behaviors—a goal shared by research on software architecture [45]. The kind of coordination-centered mentality we are promoting in this paper suggests that one should specify how the unit of mobility interacts with its context separately from the behavior of the unit itself. Many models for coordination are based on the Linda model [15] which provides essentially a shared memory abstraction, a globally shared tuple space that defines a fixed context for the entire application. The result is a style of communication that is decoupled and implicit. These features are particularly desirable in the mobile setting, where mobile units are frequently changing context and do not necessarily know which partners are present at any given time. Tuple spaces may reside on servers and be accessible as long as connectivity exists but they may also be partitioned among the units of an ad hoc network and recombined whenever units are in contact with each other as in the case of LIME [38]. Other forms of transparent, transient and transitive sharing of data have been investigated in Mobile UNITY for use in ad hoc applications while mobile agent systems define binding rules that allow agents to interface with resources on the current server. In general, transparent coordination mechanisms may be desirable because of their ability to accommodate the design of open systems. An important question here is which model is most likely to increase software productivity and dependability in the next generation of mobile systems.

One aspect of coordination involves the ability to determine who else is around. Even though fully transparent communication, à la Linda, promises to simplify the programming effort, there are applications that require explicit knowledge of the participants in the computation. This information may also be useful for performance optimization purposes. Furthermore, models that aspire to broad applicability must be able to express both high level coordination constructs made available to the application programmer (e.g., transparent variable sharing) as well as low level coordination constructs relating to system programming. These issues are well-known in distributed computing, and gave rise to a variety of naming and lookup schemes, including brokering and trading. However, mobility poses some novel challenges. First, the information about the components defining some unit's context varies with high frequency. Second, many of the naming schemes devised so far, distributed as they may be, assume close coupling between name repositories, e.g., in the Internet's Domain Name Service. This is impractical in logical mobility, where mobile agents are exploited in order to provide disconnected operation, and almost impossible in physical mobility, where the fluidity of the network disallows any assumptions about the availability of nodes. In the end, models of mobility are likely to include naming schemes, discovery capabilities, and registries and they will need to cope with inconsistent views among units.

The ability to detect whether the context has changed, e.g., whether a given unit is now part of the context, is often a precondition for the ability to react to such a change. Timely reaction is often a requirement, because some actions may be enabled for a limited time after an event occurs (e.g., after two mobile agents become co-located, or after the noise level on a wireless link goes beyond a given threshold). Letting the component interested in handling an event probe for its occurrence proactively may not be acceptable, due to the potentially high number of conditions to be verified and of parties involved. Instead, a reactive approach may be more appropriate, allowing the interested component to provide a specification of the event condition and of the actions that should handle of it. The portion of context considered for evaluating the enabling condition and the degree of reactivity (i.e., the degree of atomicity of the reaction with respect to the event occurrence) is what discriminates among these models. At one extreme, event-based systems [42, 11] consider only the *occurrence* of events that are filtered through a given specification. The corresponding reaction is guaranteed to execute eventually. At the other extreme, there are models [38] where the enabling condition is a particular *state* of the system (i.e., of the context), and the reaction to a state change is completed before any other state change is performed. The question about what degree of atomicity and style of reaction is more reasonable for mobility is still an open one in the research community.

Another issue related to the context-aware style of computing fostered by mobility is how aware should a component be of what is around and how much should it tell others about itself. This brings up the issue of security. Mobile devices should not be able to access services their owners are not entitled to and, similarly, program fragments should not misuse the resources available in the current context. Research on security has focused mostly on models that allow representation and reasoning about security protocols [2]. The real challenge is to identify proper tradeoffs between expressive power and security concerns, to investigate them formally, and to achieve the ability to prove immunity to attacks as suggested in Volpano et al. [48]. Interestingly, the notion of reactivity discussed above may help in devising security mechanisms that are decoupled from the abstractions they protect, and yet can filter out undesired state changes by executing compensatory reactions.

The manner in which we deal with the context is greatly affected by whether it is distributed or localized. In logical mobility, for instance, the context is typically localized within the boundary of a host. A code fragment is moved onto a different host in order to exploit some resource or service provided locally. Network communication is exploited only during the migration process. In contrast, physical mobility seems to require a distributed notion of context. Mobile hosts construct the context through wireless communication and the resources and services that contribute to defining the context are provided by the other components and are accessed in a distributed fashion. While it might be reasonable to look at both logical and physical mobility under the same modeling lens, their nature is intrinsically different. The extent to which it is reasonable to treat both forms of mobility as one remains an open question that demands careful consideration. Where is the threshold separating the realm of logical mobility from the one of physical mobility?

The extent to which a model is implementable is determined by the kinds of constructs it provides and the target environment for the implementation effort. For instance, a model providing strong atomicity guarantees (e.g., a notion of transaction) can be implemented easily for logical mobility, if the context of the computation is localized to the node that is hosting the migrating code. Trying to provide the same guarantees in the realm of physical mobility may not even be possible. The impossibility of distributed consensus sets the limit for the guarantees that can be provided. Moreover, communication failures are no longer temporary and relatively rare as it is assumed in distributed computing models. Disconnection may last for long periods of time. In some mobile cases (e.g., low density ad hoc networks) two parties that have been in contact once may never be in range again. Disconnection is frequent and occurs both due to the lower reliability of the communication link, and due to explicit user actions, e.g., the desire to save battery power. Faults cannot be ignored and cannot be pushed outside the model as an implementation issue.

In this light, one may consider two kinds of mobility scenarios. In one case, the specifics of the environment in which the host operates is carefully factored into the choice of constructs which are tailored for that particular setting. For instance, transactions with strong atomicity guarantees may still be appropriate in a scenario where hosts are allowed to disconnect only voluntarily (e.g., in an ad hoc network constituted by a group of co-workers meeting in the hall of a hotel while on travel), or where hosts move always as a cluster (e.g., a group of robots). The other option is to provide a rich set of constructs, some of which may turn out not to be appropriate in a particular setting and to rely on schemes that restrict the generality of the model in a particular setting. In this manner a model provides a range of atomicity guarantees and associated costs. It becomes the responsibility of the designer to exercise proper discipline over the usage of the model in a specific setting. In this manner the number of hosts, frequency and patterns of movement, power and noise constraints may be factored into the way the model is employed. These issues, typically not considered in models of computation, are central to a full understanding of mobility and they must receive proper consideration.

Formal models enable precise description of the semantics of existing languages and systems and formal reasoning about their correctness. In the novel field of mobility, models appear to assume an increased level of significance. Models must be used as intellectual tools to uncover the conceptual grounds of mobility and, armed with the power of abstraction, highlight parallels and differences among the various forms of mobility as well as conventional distributed computing. Mobility may even throw a different light on the role of reasoning and correctness proofs. Reasoning about locations could be exploited not only to determine the correctness of a system, but also to optimize its configuration. For instance, by analyzing formally the patterns of migration of a group of mobile agents, proper placement of code could be determined in advance in order to minimize remote dynamic linking.

In the past the impact of models was felt most directly through the development of new languages and associated tools. This is no longer true today. Novel mobile applications with great intellectual and commercial success are likely to benefit much more from the development of appropriate middleware than from any advances in language technologies. As such, we see middleware as the conduit through which research on models for mobile computing will exercise its greatest influence of software engineering practice.

Algorithms

The algorithms we employ reflect the assumptions we make about the underlying systems. As the shift to mobile computing is taking place, it is natural to expect that new algorithms would need to be developed. Location changes, frequent disconnections, resource variability, power limitations, communication constraints, dynamic changes in the connectivity pattern, all contribute to a demand for new algorithm design strategies. Given the diversity of mobile systems, the range of options is enormous and indeed research on mobile algorithms spans a broad spectrum. Some of the work, however, reflects what one might consider short-term technological limitations that will eventually be overcome or do not enjoy universal applicability. Power consumption falls in this category. Research on energy efficient algorithms is interesting but not necessarily fundamental. Even the concern with quality of service, particularly in multimedia applications, is probably not of the essence. Such research fits best in the category of system infrastructure design rather than algorithms. Of course, some specific elements of these problem areas may survive the process of abstraction and make their way into fundamental algorithms. For instance, algorithms involving asymmetric communication channels may end up being studied because it takes less power to listen to a signal than to broadcast it. Ultimately, it is the treatment of space and coordination again that shape the landscape of mobile algorithms.

The ability of a mobile component to move through space requires new algorithms to control and manage information about its location and that of other components. Spatial knowledge is important in many applications involving independent purposeful movement, cooperative activities, or involuntary movement. In settings where components have control over their own location, forming and maintaining geometrical shapes proves useful. For example in the task of robot exploration of an open field for unexploded ordnance [28], the ability to follow a leader through a known safe path is one useful application of a group movement strategy. Similarly, clustering around or encircling an object can be used to identify an object's boundaries, protect other group members from danger, or protect the object itself. Both of these are examples of geometrical global invariants which have been specified and achieved by describing algorithms to effect local, independent movements.

Less specifically tied to geometry is the necessity in a sparse network to maintain connectivity among all components. Maximizing functions such as total covered area or distance between the farthest components guide individual movements while keeping the group goals. In highly populated networks with possibly millions of nodes, connectivity is almost guaranteed, but organization is critical. Hierarchical structures that mimic the organization of the human body from cells into organs, and organs into a functioning whole offer immediate applications to scoping issues, communication capability, and possible movement patterns [10]. While these examples tend to highlight opportunities in ad hoc mobility, nomadic computing and logical mobility also demand the ability to leverage off knowledge about component locations. This is usually accomplished by keeping track of where mobile units are located on location servers that are queried for up to date information. Variations in the assumptions made about the number and placement of servers, and in update and query procedures are likely to lead to a rich set of algorithmic studies of practical significance [40]. Other sources of potentially interesting algorithms may be the result of exploiting metrics over the space and relative distances. Distance information, for instance, is commonly utilized in route optimization.

Other aspects of mobility entail more of a coordination perspective on algorithm development. Mobile components often work together to perform collective tasks which need to be monitored and controlled. Although many of these task oriented algorithms have been solved for traditional distributed computing, the reality of voluntary disconnection of mobile components demands the redesign of these algorithms with mobility in mind. For example, a traditional distributed snapshot relies on the availability of communication between neighboring nodes. In a mobile system, not only do neighbor sets change, but disconnections often prohibit communication with some components for extended periods of time. Global checkpointing [3], causal event ordering [39], leader election, and termination are other examples of algorithms which are meaningful to mobile distributed processing and must be revisited to account for disconnections. Transactions involving mobile components must be reexamined to address the movement of components, location dependent queries, and data delivery to future locations [12].

In addition to coping with disconnection, algorithms must address the issue of mobile component interactions even in the presence of connectivity. For example, the ability for components to communicate via message passing can no longer be taken for granted because components can constantly change location making delivery difficult while still remaining connected.

Strategies used in the development of algorithms for mobility vary widely. In the presence of a fixed support infrastructure, the most common strategy is to push computation and communication away from the mobile components and wireless links and onto the infrastructure [5]. For example, in the case of checkpointing, while storage on physically mobile devices may be limited and even inaccessible due to disconnection, the state of the mobile components can be stored at a fixed node and communicated to other nodes along a fixed, higher bandwidth communication medium.

When a network infrastructure does not exist or the network has no inherent structure of its own, an artificial structure can be imposed over the components, grouping them for communication concerns or creating a hierarchy for management. The ability to maintain and rely on this structure depends on the patterns of movement of the mobile components. Different situations call for a variety of patterns ranging from general connectivity constraints such as eventual transitive communication between all pairs of components, to physical movement characteristics such as a predetermined path or direction of movement. These general patterns can be exploited by any fundamental algorithms.

Other strategies try to exploit the advantages of known algorithm design paradigms and re-adjust them for mobility. For example, randomized algorithms can be used to generate probabilistic results when component reconnection is uncertain. Alternately, if connectivity is guaranteed to be reestablished, disconnection may be viewed in a manner similar to a network fault. In this case, fault tolerant algorithms and self stabilizing techniques can be applied. Epidemic algorithms may prove to be the key to distributing information to components when connectivity is available. In logical and nomadic computing, traditional distributed algorithms can be exploited for the purposes of mobility. For example, a distributed snapshot can be manipulated to provide unicast and multicast message delivery by treating the mobile units as messages and delivering a message rather than recording state [32]. After a similar transformation, diffusing computations can be altered to track the movement of a mobile node through the network rather than to track the expansion of a distributed computing application [31].

The availability of a standard and well-understood set of algorithms, supported through formal models and middleware, is a measure of the field's level of maturity but also an asset for the developer community. Experience with distributed computing has shown that problems that may appear to be simple have very subtle solutions prone to error. This is likely to continue to be the case in the area of mobile computing.

4 Systems research

Within the research community there is a growing recognition of the fact that systems research can no longer focus almost exclusively on performance but must shift its attention towards the end-user requirements for dependability and ease of use [16]. This need is even more acute in the mobility field where the most visible impact of software engineering research will be in the wide range of applications expected to emerge on the market in the very near future. This suggests a need to consider a style of research that is much more application centered than in the past. In this section we first examine the range of mobility applications currently under consideration and the application characteristics most likely to shape the kind of middleware that will be required to make their development a success.

Applications

Current trends in computing technology include the manufacturing of increasingly smaller, more powerful, and more portable computing devices. A glance around any airport terminal shows that notebook computers are pervasive among business travelers. Common usage of these computers is for tasks that require no interaction with outside resources, also referred to as disconnected operation. The Coda filesystem [22], for instance, supports this by allowing users to specify a set of files to be hoarded on disconnection. On reconnection, any update conflicts within this set of files must be explicitly handled by the user.

Another common task for mobile users is access to remote resources such as the Internet or company database systems. Recently 3Com released the Palm VII personal digital assistant with built in wireless capabilities for accessing the Internet [1]. By simply raising an external antenna, a connection is made to the nationwide private 3Com network. No wireless ethernet or cellular modem is necessary. Cellular telephones with limited Internet access are also becoming commonplace. Although the user interface is limited by screen size and resolution, the ability to access information is key. To access a corporate database from a mobile device, Oracle provides support for three common database operations [34]. First, users are able to manage a database remotely. Second, partial database replication allows mobile devices to carry a piece of the data and have constant access, possibly out of date with the original. Third, by using a mobile agent paradigm, mobile users can pose queries while disconnected, an agent collects these queries and when a connection is available to the database the agent moves to the server. The user can then disconnect while the queries are being processed, and when the connection is reestablished, the agent moves back to the mobile host were the results are accessible.

Smaller devices, such as active badges [49], provide several interesting application scenarios. If a badge is associated with an individual, when that user moves to a new room, the environment in that room can automatically adjust to predefined user preferences. Alternately, badges can be attached to equipment and be used to locate those objects as they are moved into different laboratories throughout an office complex. These systems rely on an infrastructure to track and make available such information.

Another mobility scenario, different from the client-server model, describes a group of individuals coordinating on a project in an environment without network support. For example, laptops carried to a program committee meeting should be able to interact to construct a short-lived network during a plenary session or allow division into multiple independent networks to support individual working groups. Similarly, the participants in a conference may form an ad hoc group with the need to share information such as business cards, schedules, session notes, etc.

Global positioning systems are becoming popular devices in many automobiles and, while the design of these devices does not require access to remote data, once wireless access becomes readily available, new kinds of applications may be considered. For example, cars moving in opposite directions could share information about road conditions on recently traveled roads. Those moving in the same direction may be able to coordinate for extended periods of time on a variety of tasks. Another interesting possibility is the placement of information kiosks at key places throughout a city or countryside. These kiosks could provide location specific information such as tourist information, available to the automobiles over a low power wireless link while themselves being connected to a fixed network.

Specialized computing devices can contribute to the emergence of yet other interesting applications. In a teaching laboratory, multiple devices can coordinate to assist a student with an experiment by providing instructions, performing computations, and collecting and displaying information from multiple instruments in a single place. At a smaller level weaving processors into clothing enables wearable computing, and thus a more natural way to carry and access computation power while moving. Tiny devices, such as those proposed by the Dust project [20], have potential as sensing devices spread throughout a room or desktop. Although some of these scenarios may appear to be the stuff of science fiction, both society and technology are moving in this direction. Of course many technical challenges must be addressed before they become reality and they place new requirements on the middleware technology. Some of them are highlighted in the remainder of this section.

One of the first concerns a developer must address is defining the user perception of the application with respect to the degree to which mobility is exposed at the application level. If the user is location-aware, one must face the related question of how the user is made aware of this property. In a mobile filesystem such as Coda, the user explicitly specifies information to be hoarded on disconnection and explicitly resolves conflicts on reconnection. Alternately, when accessing location dependent information, such as a query for *local* resources in Odyssey [44], the user should be able to make a generalized query and have the system perform the specific resolution. In robot scenarios with autonomous movement, it may be reasonable to hide the absolute location and expose only relative positions among components, thus allowing each component to think of itself as the center of the universe.

Variability in quality of service parameters is another factor that may contribute to the user's perception of location and movement. As a user moves, possible bandwidth degradation requires some form of adaptation in the behavior of the application. Odyssey provides a nice illustration of this feature by allowing control over the fidelity of data on the fly. In a video session, for example, frame rate and frame quality provide two tuning parameters. In general, applications must offer a variety of adaptive parameters that affect the presentation style and make other adjustments reflecting different levels of knowledge about the overall configuration and available information.

Similar situations are encountered when entering and leaving administrative domains especially when they have diverse levels of security. From the user perspective, the amount of personal information to be shared must vary depending on context. The ability to express and alter both individual security policies and security demands of a domain is important to many mobile applications. This ties in the issue of open environments in which applications on mobile components must be able to interact with other mobile components about which no prior knowledge exists and, similarly, with applications never before encountered. While it is possible to prohibit such interaction entirely, it is much more preferable to provide mechanisms that are capable of discovering beneficial modes of interaction in new circumstances. The ability to adapt to an open environment must be weighed against the associated costs. Openness, for instance, may compromise security while excessive generality may require too many resources.

Assessing the capabilities of the environment is also important for effective performance of an application. Mobile devices range from relatively high-power portable notebook computers to low-power personal digital assistants with limited display and computation; communication capabilities may include powerful base stations enabling full connectivity among all mobile components or may be limited to ad hoc environments in which repartitioning and changes in connectivity pattern are frequent. Finally, the speed and pattern of movement can also exhibit great variability. This variety of environmental conditions makes application development challenging, but the ability to accommodate increases the potential degree of penetration by mobile applications in the society at large.

Middleware

Middleware supports the software development task by enhancing the level of abstraction associated with the programming effort. Middleware adds mechanisms and services that are much more specialized than those provided by the operating system, within the context of established languages without modifying their syntax and semantics. Recent years have seen a flurry of middleware developments for distributed systems. It is then reasonable to expect that a new generation of middleware specialized for mobility will follow suit. Despite the similarities between logical and physical mobility, research on middleware tends to treat the two forms of mobility very differently. Besides factors that have to do with separation of the related research communities, a compelling reason for this situation rests with the different roles logical and physical mobility play with respect to application development.

Logical mobility is essentially a new *design* tool for the developers of distributed applications. The ability to reconfigure dynamically the binding between hosts and application components provides additional flexibility and, under given conditions, improved bandwidth utilization. On the other hand, physical mobility poses new *requirements* for distributed applications, by defining a very challenging target execution environment. These different roles are mirrored in the characteristics of the corresponding middleware. Middleware for logical mobility is centered around new abstractions that enable code and state relocation, whereas middleware for physical mobility often tends to minimize differences with respect to non-mobile middleware, by relegating, as much as possible, the differences into the underlying runtime support. In the remainder of this section, we report about the state of the art in the field and highlight some of the open research issues.

Traditionally, middleware for *physical mobility* has been application centered. For instance, the Bayou [47] system provided the core functionality needed to build database applications that can handle disconnection through reconciliation and data hoarding. This approach was symptomatic of an interpretation of mobile computing as a very specialized and rare form of computing that could be accommodated with application specific support and by exposing as little as possible of its characteristics to the user. Although this view may still hold true for many applications, with the rise of mobility as the base of future computing, general purpose middleware becomes more of a necessity. Hiding mobility becomes more difficult, if at all meaningful, and a new core of abstractions that extend distributed middleware with support for mobility must be devised.

Following the concepts illustrated in Section 3, the unit of mobility we consider here is a mobile host. Finer grained mobility is conceivable but requires one to consider software and data residing on the host, a situation that is the privy of logical mobility, a topic covered later. In regard to physical mobility, the challenge for mobile middleware is to devise mechanisms and constructs to allow detection of changes in location, to specify what belongs to the context of the computation, to relate changes in location to context modifications, and to determine how the computation itself is affected by changes in the context.

Many issues related to tracking the dynamics of location and context require tight interaction with the underlying operating system and device. Of particular significance is the availability of mechanisms that enable detection of connectivity, of variations in the quality of service of communication, of the appearance of new mobile hosts within communication range, and of battery power status. All these considerations are of paramount importance for the core of mobile applications and constitute a major point of departure from distributed computing, where the need for primitives that dig so deeply into the underlying machine is more the exception than the rule. For the time being, availability of such mechanisms and primitives is heavily constrained by the lack of appropriate programming interfaces at the underlying wireless device level.

Similar constraints exist for detecting changes in the location of a mobile device. Location management is a novel and interesting requirement of mobile middleware, one that is likely to become more and more important as experience with a wide range of truly mobile applications becomes available. Managing the location of a mobile host may assume many different nuances. It is desirable to have mechanisms that allow the programmer to determine where the host currently is and to maintain a history of the visited locations. Furthermore, it is natural to think about their integration with mechanisms that allow reactive modification of the context. It should be possible, for instance, to have location changes trigger specialized computations in order to reconcile data or to determine the role the mobile host must assume upon entering a new administrative domain. Location may be absolute or relative to that of other neighbors. In both cases, primitives are needed to define a notion of space and the associated notions of position and distance. It should be noted that relative locations pose demanding requirements on location management, as they presuppose the ability to track continuously the movements of a given set of mobile hosts. In a world of autonomous mobile entities, tracking services may become fundamental to enable cooperation when decoupled computation is not possible.

A different set of issues that middleware for mobility must consider are actually well known in distributed computing, but need to be redefined in the new context. Service lookup belongs to this category. In distributed computing, the problem of discovering available services is often solved by forcing service providers to register with a server. In many popular architectures, e.g., Jini [29], the server is essentially centralized and more sophisticated schemes that take into account mobility being hand-coded on top of Jini. Instead, mobility scenarios often require constructs that allow the programmer to perform service lookup without any knowledge about the configuration of the current context. LIME [38] is one system that provides such capability.

A well known alternative to centralized service discovery is the use of an event dispatching mechanism, which provides also for reactive capabilities. Although most commercially available event dispatching systems are indeed centralized, there is a significant body of research on distributed events growing both in industry and academia. Mobility complicates further the picture of dispatching events in a distributed fashion. Hierarchical configurations of dispatchers, like those proposed in [11], are no longer suitable when confronted with the fluid configuration of mobile hosts. Disconnection translates to the impossibility of delivering an event to a subscriber for a given time interval, thus raising the problem of how to reconcile the view of the subscriber upon reconnection. If events generated during disconnection are discarded, the subscriber may miss relevant events; if, on the other hand, events are queued and transmitted to the subscriber, the overhead of this bulk transmission may be prohibitive. Finally, delivering an event to a mobile unit may become a problem itself, even in presence of a fault-free network. Other issues that need to be revisited for mobility include mechanisms for security and access control, as well as support for transactions (which have been already discussed, although at a different level of abstraction, in Section 3).

Early approaches to *logical mobility* started out as what nowadays would be called middleware. For instance, the REV system [46] provided an extended version of remote procedure call where the client could specify the code of the procedure to be executed, and the Emerald [19] system provided an object-oriented layer on top of an operating system that handled transparent object migration. By contrast, recent approaches to logical mobility focused initially on the design of new languages or on the extension of already existing languages with primitives expressly conceived for handling logical mobility. This is the case of Telescript [50] and Facile [23], among the others. The creation of a brand new language was justified by the absence, in traditional languages, of hooks into the runtime support to enable relocation of code and state. The fact that today these systems, that nevertheless influenced heavily subsequent developments, are relegated to a totally marginal role is a symptom of the current trend dominated by systems based on the Java language. Java provides some of the runtime hooks, notably the ability to reprogram dynamic linking, combined with a degree of portability and security that, although not optimal, is still higher than what many other platforms provide.

However, current middleware for logical mobility is falling short of expectations. On one hand, there are mobile agent systems, i.e., systems providing as the main abstraction a unit of mobility coincident with the unit of execution. Despite the initial excitement about this notion of mobile agents, technology did not meet the expectations. Most existing systems provide basically the same abstractions with the same limitations. In many respects, rather than building mobile agent systems as a facility that can interoperate with mainstream distributed middleware, many systems reimplement support mechanisms like events, dispatching, directory services, transactions, messaging. This could be justified by the challenges logical mobility poses on the implementation of such services (very similar to those present in physical mobility). Yet, in many instances the tough problems are left unsolved and the mobility of agents is curtailed (impacting negatively on the very reason for the existence of mobile agent system). The key observation that logical mobility is just another design tool, and it should be made available to the programmer in combination, and not in alternative, to distributed middleware is not acknowledged by these systems. Notable exceptions, representative of very different design strategies, are Voyager [21], a distributed middleware that provides object mobility as one of the many features of a full-fledged platform, and μCODE [37], a minimal, lightweight support for mobile code providing abstractions that enable the relocation of any mixture of code and state, thus encompassing also the notion of mobile agent.

At the other extreme there are systems that exploit logical mobility by choosing a unit of mobility smaller than the unit of execution, typically the Java class. In contrast to the notion of mobile agent, this finer-grained logical mobility is finding its way into popular distributed middleware like Java/RMI and Jini [29]. In these systems, logical mobility is exploited for the sake of improved flexibility. While the benefits of static type checking are retained through the notion of a mutually agreed service interface between client and server, the implementation of such service may be changed dynamically by using subtyping and code mobility. The problem with this form of middleware, however, is that it exploits only a minimal fraction of the power provided by logical mobility. Only the code on demand paradigm [14] is supported; other paradigms, like mobile agent or remote evaluation, that have been proven useful [6], must be hand-coded. No relocation of state is allowed, except for the ability to copy the entire closure of an object that is being passed as a parameter of a remote invocation.

Contrary to popular belief, building support for relocation of code and state is not a monumental endeavor, especially using the Java language which already provides many of the necessary building blocks. The real issue is the design of the constructs that are made available to the programmer and their underlying conceptual model. Researchers have only begun to scratch the surface of discovering the level of flexibility provided by logical mobility. The next challenge is to provide support for varying grains of mobility, mechanisms allowing different rebinding strategies, and different architectural styles for relocation—all in a single, uniform programming interface.

In doing this, the position of Java as the supporting language for these efforts can be challenged. It has already been shown how it heavily constrains some choices related to logical mobility. For instance, the lack of a mechanism in the Java virtual machine for saving and restoring the execution state of a thread complicates the implementation of systems supporting strong mobility. Similarly, the lack of a resource monitoring mechanism severely limits the development of systems that provide mechanisms for security and accounting. Finally, the units of mobility supported by Java (i.e., classes and objects) and the related serialization mechanism may need modification, as they proved to be too coarse and heavyweight for many applications.

Coordination, by abstracting away from the behavior of the mobile units and focusing on high level communication protocols, may provide a way to rejoin the logical and physical mobility in a single, uniform framework. In particular, systems based on tuple spaces provide a suitable and direct abstraction for an unstructured (and thus general) representation of the context where a mobile computation is performed. This way, coordination middleware does not impose specific data structures to represent the constituent of the context, instead, it provides basic mechanisms that rule the access, modification, and consistency of such data structures.

It is interesting to note that the advantages of coordinating distributed components through a Linda-like model are well recognized also by the industry, where companies like IBM and Sun compete with their Java-based implementations of a tuple space called TSpaces [17] and JavaSpaces [18], respectively. The two systems have slightly different implementations but a very similar philosophy. The degree of distribution is still extremely limited, as these systems essentially provide remote access to a centralized tuple space which acts as a tuple server providing shared access to clients. No support for disconnection is provided, and the presence of a centralized, well-known server almost instantly rules out applicability to an ad hoc network setting. Logical mobility is more of an hindrance than an asset for these systems, as downloading of tuple code is not handled automatically. The Linda model is coupled with a primitive event system that augments the expressive power, but its policies and guarantees are not easily adaptable by the user in need of specialized and reactive cooperating behavior.

Some academic systems push further the coordination perspective by providing systems that tie together Linda with mobility. For instance, the MARS and TuCSoN systems [8] provide the notion of a reactive tuple space. Changes in the tuple space content trigger reactions that modify the tuple space. This view of a reactive tuple space is common also to LIME [38], which combines it with a notion of transiently shared tuple space, designed to accommodate support for dynamic reconfiguration of mobile agents and mobile hosts within the same programming interface.

By and large, these coordination approaches tend to adopt a coarse grain perspective, providing support for coordination of mobile agents and mobile hosts. Nevertheless, mobile code could be exploited as a means to modify dynamically the behavior of such mobile components, e.g., by employing schemes where tuples actually contain code, as in the MobiS model [25], and providing reactive rules for their dynamic linking and execution.

Independently of the slant towards coordination, however, middleware systems are ultimately generated through a design mindset and, as such, they are the result of compromises resulting from proper evaluation of tradeoffs. A first relevant tradeoff is about how much power should be put in the hands of the programmer. Middleware platforms nowadays tend to provide extremely rich interfaces, i.e., powerful and expressive constructs, at the cost of increased complexity, poor conceptual cohesion, and high performance overhead. The other tradeoff is between horizontal coverage for a broad range of scenarios and configurations (e.g., a platform providing abstractions that span from the fixed to the ad hoc setting) in contrast with a vertical coverage of specific scenarios (e.g., providing support only for palmtop devices in a nomadic setting). Identification of the proper balance between these opposing forces, combined with effective and validated support to real world applications, is what will ultimately determine the emergence of a new generation of mobile middleware.

5 Conclusions

Advances in wireless technology, extensive investments in telephony, and the Internet's ability to provide ubiquitous access to information are the main forces that shaped the emergent field of mobile computing. Low level protocols, personal communication appliances, and web content delivery have been some of the most visible elements of this new computing arena. The success and popular acceptance of this technology is accompanied by rapid growth, increased demand for novel applications, and high expectations with regard to quality and dependability. The time has come for the software engineering community to embrace mobile computing as the next frontier to be conquered. Across the entire spectrum of software engineering endeavors, mobility challenges old assumptions and demands new kinds of solutions. In this paper we sought to convey the intellectual excitement generated by research opportunities in mobile computing and to identify some of the main research issues the field is facing today.

REFERENCES

[1] 3Com. Palm VII Connected Organizer web page. `http://www.3com.com/palm/palm_vii/`, 1999.

[2] M. Abadi and A.D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.

[3] A. Acharya, B.R. Badrinath, and T. Imielinski. Checkpointing Distributed Applications on Mobile Computers. In *Proc. of the 3rd Int. Conf. on Parallel and Distributed Information Systems*, October 1994.

[4] R. Amadio. An Asynchronous Model of Locality, Failure, and Process Mobility. In *Proc. of the 2nd Int. Conf. on Coordination Models and Languages (COORDINATION '97)*, LNCS 1282. Springer, 1997.

[5] B.R. Badrinath, A. Acharya, and T. Imielinski. Designing Distributed Algorithms for Mobile Computing Networks. *Computer Communications*, 19(4):309–320, April 1996.

[6] M. Baldi and G.P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proc. of the 20th Int. Conf. on Software Engineering*, 1998.

[7] J. Broch, D.B. Johnson, and D.A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks . Internet Draft, October 1999. IETF Mobile Ad Hoc Networking Working Group.

[8] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. In *Proc. of the 2nd Int. Workshop on Mobile Agents*, LNCS 1477. Springer, 1998.

[9] L. Cardelli and A. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1), 2000. To appear.

[10] D. Coore, R. Nagpal, and R. Weiss. Paradigms for Structure in an Amorphous Computer. A.I. Memo No. 1614, Massachusetts Institute of Technology Artificial Intelligence Laboratory, October 1997.

[11] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. on Software Engineering.* To appear.

[12] M. Dunham, A. Helal, and S. Balakrsihnan. A Mobile Transaction Model that Captures both the Data and Movement Behavior. *ACM-Baltzer Journal on Mobile Networks and Applications (MONET)*, 2(2):149–162, October 1997.

[13] C. Fournet, G. Gonthier, J.J. Levy, L. Maranget, and D. Remy. A Calculus of Mobile Agents. In *Proc. of the 7th Int. Conf. on Concurrency Theory (CONCUR)*, LNCS 1119. Springer, 1996.

[14] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Trans. on Software Engineering*, 24(5), 1998.

[15] D. Gelernter. Generative Communication in Linda. *ACM Computing Surveys*, 7(1):80–112, Jan. 1985.

[16] J. Hennessy. The Future of Systems Research. *Computer*, 32(8):27–33, August 1999.

[17] IBM. T Spaces web page. `http://www.almaden.ibm.com/cs/TSpaces/`, 1999.

[18] JavaSpaces. The JavaSpaces Specification web page. `http://www.sun.com/jini/specs/js-spec.html`, 1999.

[19] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained Mobility in the Emerald System. *ACM Trans. on Computer Systems*, 6(2):109–133, February 1988.

[20] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Mobile Networking for Smart Dust. In *Proc. of the 5th Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking*, Seattle, WA, USA, August 1999. ACM.

[21] J. Kiniry and D. Zimmerman. A Hands-On Look at Java Mobile Agents. *IEEE Internet Computing*, 1(4), 1997.

[22] J.J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Trans. on Computer Systems*, 10(1):3–25, 1992.

[23] F.C. Knabe. *Language Support for Mobile Agents.* PhD thesis, Carnegie Mellon Univ., Pittsburgh, PA, USA, December 1995.

[24] T.M. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[25] C. Mascolo. MobiS: A Specification Language for Mobile Systems. In P. Ciancarini and A. Wolf, editors, *Proceedings of the 3rd Int. Conf. on Coordination Languages and Models (COORDINATION)*, volume 1594 of *LNCS*, pages 37–52. Springer, April 1999.

[26] C. Mascolo, G.P. Picco, and G.-C. Roman. A Fine-Grained Model for Code Mobility. In *Proc. of the 7th European Software Engineering Conf. held jointly with the 7th ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE '99)*, LNCS, Toulouse (France), September 1999. Springer.

[27] P.J. McCann and G.-C. Roman. Compositional Programming Abstractions for Mobile Computing. *IEEE Trans. on Software Engineering*, 24(2), 1998.

[28] J. McLurkin. Using Cooperative Robots for Explosive Ordnance Disposal. Massachusetts Institute of Technology Artificial Intelligence Laboratory.

[29] Sun Microsystems. Jini web page. `http://www.sun.com/jini`.

[30] R. Milner. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.

[31] A. Murphy, G.-C. Roman, and G. Varghese. Tracking Mobile Units for Dependable Message Delivery. Technical Report WUCS-99-30, Washington University, Dept. of Computer Science, St. Louis, MO, USA, December 1999.

[32] A.L. Murphy and G.P. Picco. Reliable Communication for Highly Mobile Agents. In *Proc. of the 1st Int. Symp. on Agent Systems and Applications and 3rd Int. Symp. on Mobile Agents (ASA/MA '99)*, pages 141–150, Palm Springs, CA, USA, October 1999. IEEE Computer Society.

[33] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Trans. on Software Engineering*, 24(5), 1998.

[34] Oracle. Oracle 8*i* Lite web page. `http://www.oracle.com/`, 1999.

[35] V. Park and S. Corson. Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification. Internet Draft, October 1999. IETF Mobile Ad Hoc Networking Working Group.

[36] C.E. Perkins, E.M. Royer, and S.R. Das. Ad Hoc On Demand Distance Vector (AODV) Routing. Internet Draft, October 1999. IETF Mobile Ad Hoc Networking Working Group.

[37] G.P. Picco. μCODE: A Lightweight and Flexible Mobile Code Toolkit. In *Proc. of the 2nd Int. Workshop on Mobile Agents*, LNCS 1477. Springer, 1998.

[38] G.P. Picco, A.L. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In D. Garlan, editor, *Proc. of the 21st Int. Conf. on Software Engineering*, pages 368–377, May 1999.

[39] R. Prakash, M. Raynal, and M. Singhal. An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, pages 190–204, March 1997.

[40] R. Prakash and M. Singhal. A Dynamic Approach to Location Management in Mobile Computing Systems. In *Proc. of the 8th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'96)*, pages 488–495, June 1996.

[41] G.-C. Roman, P.J. McCann, and J.Y. Plun. Mobile UNITY: Reasoning and specification in mobile computing. *ACM Transactions on Software Engineering and Methodology*, 6(3):250–282, 1997.

[42] D.S. Rosenblum and A.L. Wolf. A Design Framework for Internet-Scale Event Observation and Notification. In *Proc. of the 6th European Software Engineering Conf. held jointly with the 5th ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE97)*, number 1301 in LNCS, Zurich (Switzerland), September 1997. Springer.

[43] D. Sangiorgi. *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms.* PhD thesis, Computer Science Dept., Univ. of Edinburgh, 1993.

[44] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), 1996.

[45] M. Shaw and D. Garlan. *Software Architecture: Perspective on an Emerging Discipline.* Prentice Hall, 1996.

[46] J.W. Stamos and D.K. Gifford. Remote Evaluation. *ACM Trans. on Programming Languages and Systems*, 12(4):537–565, October 1990.

[47] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. *Operating Systems Review*, 29(5):172–183, 1995.

[48] D. Volpano. Provably-Secure Programming Languages for Remote Evaluation. *ACM Computing Surveys*, 28A, December 1996. Participation statement for ACM Workshop on Strategic Directions in Computing Research.

[49] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Trans. on Information Systems*, 10(1):91–102, January 1992.

[50] J.E. White. Telescript Technology: Mobile Agents. In J. Bradshaw, editor, *Software Agents.* AAAI Press/MIT Press, 1996.