Software Engineering for Mobility: Reflecting on the Past, Peering into the Future

Gian Pietro Picco[†], Christine Julien[‡], Amy L. Murphy[*], Mirco Musolesi[**], Gruia-Catalin Roman[⊕]
[†]University of Trento (Italy), [‡]University of Texas-Austin (USA), [*]Bruno Kessler Foundation-IRST (Italy),
[**]University of Birmingham (UK) [⊕]University of New Mexico (USA)
Contact author: gianpietro.picco@unitn.it

ABSTRACT

At the end of the second millennium, mobility was a hot research topic. Physical mobility of devices was becoming commonplace with the availability of cheap wireless cards, the first attempts to transform phones into personal do-it-all devices were beginning to appear, and mobile ad hoc networks were attracting a huge interest from many research communities. Logical mobility of code was still going strong as a design option for distributed systems, with the Java language providing some of the ready-to-use building blocks. In 2000, when we put forth a research "roadmap" for software engineering for mobility, the challenges posed by this dynamic scenario were many.

A decade and a half later, many things have changed. Mobility is no longer exotic: we juggle multiple personal devices every day while on the move, plus we grab and update applications on a whim from virtual stores. Indeed, some trends and visions we considered in our original paper materialized, while others faded, disappeared, or morphed into something else. Moreover, some players unexpected at the time (e.g., cloud computing and online social networks) appeared on the scene as game changers.

In this paper we revisit critically our original vision, reflecting on the past and peering into the future of the lively and exciting research area of mobility. Further, we ask ourselves to what extent the software engineering community is still interested in taking up the challenges mobility bears.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Miscellaneous

General Terms

Design, Algorithms, Human Factors

Keywords

Mobility, mobile computing

FOSE'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2865-4/14/05...$15.00
http://dx.doi.org/10.1145/2593882.2593884

1. INTRODUCTION

In the context of computer science and engineering, *mobility* is a term with many nuances. Mobility is often associated with *mobile computing* using untethered computation devices, e.g., equipped with wireless communication abilities and autonomous power supply. At the end of the second millennium, technological advancements enabled a surge of opportunities in this realm. Moreover, alongside this form of *physical* mobility of devices, there was great interest also in the *logical* mobility of software components, whose code and state could be relocated entirely or partially as a design alternative to the classic client-server paradigm.

In other words, mobility was an exciting research field still relatively unexplored, with potential implications for software engineering. As researchers working in the field, some of us were charged by the ICSE community with the arduous, yet inspiring, objective to identify the software engineering challenges concerning mobility. The result was a paper [1] published in 2000 as part of the well-known "Future of Software Engineering" book [2].

In the paper, we deliberately took a broad notion of mobility, motivated by the desire to encompass both the aforementioned forms of mobility:

> *From a software engineering perspective, we view mobile computing to be the study of systems in which computational components may change location.*

A decade and half later, the conceptual and technological landscape for mobility has changed dramatically. Logical mobility is no longer a "hot topic": some of the design paradigms and implementation techniques are now taken for granted, others are relegated to historical interest, if not oblivion [3]. More importantly, the fast-paced world of computing and communication has reshaped itself a few times since our original paper, with implications on mobility. A critical re-evaluation of the challenges we identified in 2000 is therefore in order, along with a fresh look at the future that takes into account the conceptual and technological developments that appeared in the meantime.

The aforementioned considerations provide the roadmap for this paper, which indeed unfolds as a *travelogue* of sorts. In Section 2 we ideally rewind in time and place ourselves at the end of the second millennium. We first remind the reader with some anecdotal evidence of what "mobility" meant back then, from a technology standpoint, and follow with a concise summary of the considerations and challenges we[1] identified in [1]. At the end of the section, empowered by hindsight, we give a critical look at our own considerations, and comment on which of these still hold, which are irrelevant today, and which we missed. In Section 3, before the fast-

[1]The set of authors of this paper is a superset of the original one in [1]. We gloss over this detail, for the sake of readability.

forward to present day of Section 4, we analyze some of the disruptive, unexpected players—the "game changers"—that materialized in this decade and a half, and that determined the fate of some of our considerations, along with the groundwork for the present (and arguably future) state of affairs. The latter is dissected in Section 4, where we repeat the treacherous but stimulating exercise of eliciting the current macro-trends shaping the field of mobility at large, and of imagining the future ones and the challenges they bring to software engineering.

The mix of considerations over the past, present, and future that characterizes this paper is also the opportunity to ponder, in Section 5, whether the software engineering community still considers the topic of mobility important—or even just worth studying. Finally, Section 6 ends the paper with some brief concluding remarks.

2. REFLECTING ON THE PAST

In this section, we revisit critically the content of the original paper [1] we wrote in 2000. We begin by reminding ourselves and the readers what mobile computing meant back then, providing context for our following considerations. We then proceed by concisely summarizing the main challenges we identified in [1], maintaining the same high-level distinction between theory and systems that we initiated in 2000. Finally, we fast-forward to the present day and discuss what happened to those challenges—have they been solved or are they still open? Were they really relevant or were there others that, in the end, mattered more? Answering these questions helps look at the present and future state of affairs in later sections.

2.1 Mobile Computing in A.D. 2000

Fourteen years is a long time in the fast-moving field of computer science and communication engineering. Some of our youngest readers may have only a vague idea of what mobile computing meant back then and take the current wealth of pocket-size computation and communication power for granted. The considerations we expressed in [1] must be placed in context to be appreciated.

Our intent is not to provide a complete, exhaustive account. Our original paper contains a more in-depth description, and plenty of "historical" material is available elsewhere to the interested reader. Instead, in the following we single out, almost in an anecdotal fashion and not without a bit of nostalgia, some elements that are paradigmatic of the change mobile computing went through over nearly two decades.

State-of-the-art mobile device: PDA. When we wrote our paper, a distinct trend towards portable, affordable computing had already appeared. Laptops were increasingly cheap and powerful, although still typically weighing a few kilograms. However, a new form of personal computer had started to appear, targeting drastically smaller form factors: the Personal Digital Assistant (PDA).

PDAs were hardly a new concept. While they originally appeared as glorified organizers at the end of the 90's, the concept became popular again with the (in)famous Apple Newton, released in 1993, for which the term PDA was actually coined. It was only later, however, that the PDA finally exploded on the market, slowly evolving from the role of organizers to that of small computers with their own operating systems, either custom-designed like PalmOS or stripped-down versions of conventional ones, as in the case of WindowsCE. At some point, a whole continuum of devices appeared, connecting basic organizers to laptops through a plethora of devices ("palmtops" and "hand-helds") with different form factors and capabilities, such as computing power and screen real estate.

Initially, these devices were meant to be connected, at least sporadically, to a conventional computer, either through serial, USB, or infrared. Indeed, "synchronization" of user data was one of the main features available. As they were gaining power, however, these devices moved away from the organizer stereotype, and eventually gained also connectivity. In our paper we stated:

> *Recently 3Com released the Palm VII personal digital assistant with built in wireless capabilities for accessing the Internet.*

(a) Palm VII. (b) Communicator 9110.

Figure 1: Some ancestors of modern smartphones.

While this may trigger a benevolent smile today, it was big news at the time. Despite the fact that connectivity was provided by a proprietary network, the Palm VII (Figure 1(a)) proved actually quite successful and introduced other elements now commonplace in smartphones, such as location-based services, although based on inferred zipcodes rather than GPS. Yet another beast was the BlackBerry, introduced in 1999, an e-mail pager based on a proprietary network, a functionality that eventually proved key in securing a vast base of users, peaking at 79 million in 2012.

What about smartphones? PDAs can be regarded as one of the ancestors of modern-day smartphones. The term "smartphone" did not exist at the time, however, the concept of merging the functionality of a PDA with the one of a phone appeared shortly after, with the notion of a "communicator." The latter was borrowed from arguably the most successful of these devices, the Nokia Communicator, introduced in 1996. The concept of these devices was somehow similar to present-day smartphones: a hybrid between the PDA and the cellular phone, essentially extending the computation abilities of the former with the ubiquitous connectivity provided by the latter. These devices, however, were a far cry from those most of us carry in our pockets today. The clamshell design of the Nokia Communicator 9110, introduced in 1998, had a size of $158 \times 56 \times 27$ mm and a weight of 253 g, already a big leap forward w.r.t. the 397 g of the previous 9000 model. It was equipped with a 33 MHz, 486-like AMD processor and 8 MB of memory, of which only 2 MB were dedicated to user data. The "apps" were limited to the standard suite communication suite including email, SMS, fax (!), and a browser, along with productivity applications such as a notepad.

WAP: The mobile Internet (on a tiny screen). At the time of writing the original paper, the vision of ubiquitous Internet access was beginning to appear, enabled by an increasing penetration of cellular phones. To many users, the Internet essentially meant the World Wide Web. In this context, the most promising technology at the time was the Wireless Access Protocol (WAP), which appeared in 1997 and specified an entire protocol suite, from low-level datagrams up to the application, in support of Web browsing.

WAP was motivated by the impossibility to render HTML pages directly on the cellular phones available at the time. Web sites had to be replicated and written in a different markup language, the Wireless Markup Language (WML), expressly designed for rendering on the small screens of cellphones common at the time. Cell-

phones would download WML pages through a WAP proxy, sitting between the device and a common Web server.

WAP was accompanied by a lot of hype, which peaked at about the time of writing of [1]. However, the need to maintain separate Web content, along with the inherent limitations of the target platforms, impacted user experience and eventually led WAP to its current irrelevance. Nevertheless, it should be pointed out that, while WAP never really took off in Europe and the USA, it was very successful in Japan.

WiFi on PCMCIA cards. WiFi (or, more precisely, Wi-Fi) was actually established as an independent standard organization, the Wi-Fi Alliance, only in 1999, shortly before we published our paper [1]. At the time, WiFi was still definitely an exotic technology, whose actual impact remained still vague to many, even among researchers. For instance, one of us recalls a conversation with a senior, experienced colleague—also an author of a different chapter in the "Future of Software Engineering" book—who argued how useless wireless access was going to be, given that Ethernet plugs had become pervasive.

An element providing grounds for this reaction was the fact that the WLAN cards (as they were commonly called) were large, typically provided through PCMCIA connections, often with external antennas and hardware so bulky it had to be physically attached to the back of laptop screens. No wonder the aforementioned "communicators" did not have WiFi connectivity. WLAN cards were also rather slow: before IEEE 802.11b, published in 1999 and promising 11 Mbit/s, WLAN cards supported transmission rates up to 2 Mbit/s.

Mobile code and mobile agents were a "hot" topic. The examples put forth so far are all concerned with what we called *physical mobility* in [1], i.e., the ability of portable devices, carried by mobile users, to communicate and therefore perform a distributed computation while on the move.

Nevertheless, in our paper we also considered *logical mobility*, where the code and/or state associated with a process are moved across hosts. This notion was then very popular and relied on a small number of design paradigms (code on demand, remote evaluation, and mobile agent) that gave rise to a plethora of systems, roughly divided into those supporting strong (i.e., code and execution state) or weak (i.e., only code) mobility [4]. In particular, the notion of a *mobile agent* captured the imagination of many, especially in its most radical form, where a complete process is able to migrate of its own volition across hosts and yet retain its execution state (e.g., program counter).

Many applications of logical mobility were envisioned at the time, and the notion of mobile agent gave rise to a surge of formal models of mobility. Mobile code and agents were expected to radically change the way distributed applications were built. The gap between this dream and the current state of affairs is analyzed elsewhere (e.g., in [3]), and constitutes the basis for some of our considerations in the following.

2.2 What Were the Challenges

In our original paper, we outlined the open challenges with a classification into theory and systems, addressing models and algorithms in the former and applications and middleware in the latter. The intent was to focus on foundational research, then on research driven by application needs. Keeping in mind the state of mobile computing in 2000, as just summarized, we recap the challenges we saw, revisiting them in the next section to evaluate their evolution.

Theory: Models. Models allow us to step back from the details of the topic at hand to address fundamental aspects, which further allows us to focus on the core challenges. Regarding models for mobility, three critical elements we identified are the unit of mobility, location and context.

The *unit of mobility* can be defined as the smallest component of the system that moves. In a physically mobile system, models typically identify devices along with their applications and state. Analogously, models for logical mobility consider different granularities of application code, from single variables, to classes or libraries, to entire applications. One popular instantiation of a logical mobility model is *code on demand*, where a stationary application is augmented by functionality downloaded on the fly. Alternately, a model of *mobile agents* more closely resembles a physically mobile system as the execution does not necessarily stop when a mobility action is taken. Further, by separating the system state from the executing unit, models allow new perspectives on the design of languages to support mobility and its reconfiguration opportunities.

Mobility requires a notion of *location*, whether this is a physical point in space or a logical execution environment. The movement of units through these spaces and the corresponding reconfigurations make mobile systems unique; representing location explicitly in models is critical. In some modeling infrastructures, the precise definition of a location, e.g., as a Cartesian coordinate, is not critical; instead the ability to proactively change location and/or react to location changes are critical. In other frameworks, the space itself is structured, e.g., hierarchically. The chosen shape gives rise to specific movement rules among the available locations, and coordination rules among units are often tied to the location. In any case, by explicitly addressing location, mobility models uniquely emphasize the challenges arising from location changes.

While location is important, the notion of *context* moves beyond it, addressing the whole environment in which computation is performed, e.g., including the presence or absence of other mobile units, access to information, or even the task at hand. It is possible for two mobile units to share a location but perceive different contexts. Similarly, two units can share a context but not a location. In general, context-aware computing arises from an ability to react in a timely manner to changes in the environment. Although all computation takes place in some context, it is the abrupt and unpredictable nature of context changes in mobile systems that makes them unique. Taking a coordination perspective, one should specify how the mobile unit interacts with its context separately from the behavior of the unit itself. Mobile coordination approaches range from implicit reconfiguration of the coordination space to automatic, transient sharing of data. Such transparent coordination mechanisms are amenable to open systems, one of the challenges we posited for mobile systems.

To confront the challenges of context, we put forth the need for models to incorporate naming schemes, discovery capabilities, and registries to cope with changes in the availability of other nodes in disconnected environments. Additionally, we suggested the need for context-aware security mechanisms to prevent unauthorized access and simplify allowed interactions, balancing the expressive power with security concerns.

Theory: Algorithms. Next we shift from abstract models addressing mobility to the need for algorithms that generically solve recurring challenges such as location changes, frequent disconnections, resource variability, power limitations, communication constraints, and dynamic changes in the connectivity pattern. We supposed that some challenges such as power consumption would fade over time with newly developed technologies offering solutions. We also presumed that algorithms exploiting asymmetric communication would grow in importance due to the difference in reception and transmission powers.

Fundamentally, physical mobility requires a relationship with space; we presumed that algorithms addressing spatial problems would have a place. For example, in autonomous robotic systems, algorithms for systematic exploration of a space are important. Algorithms supporting spatial constraints such as coverage or traveled distance maximization/minimization were also needed.

Regarding algorithms supporting coordination, we outlined the need to support collective tasks among physically neighboring nodes. We emphasized the need for standard algorithms, such as checkpointing, event ordering, and leader election, to address disconnections among nodes, as would be the case without a support infrastructure. Without infrastructure, approaches would be required to meaningfully group the nodes according to logical or physical location, movement patterns, or the task at hand, with careful attention to expensive communication. Patterns exploiting these logical groups would be useful in fundamental algorithm development.

On the other hand, one strategy we foresaw was shifting computation and communication away from the mobile components and their wireless links and into the infrastructure, leveraging cheap and abundant communication and storage inside the wired network. Nevertheless, the need to support disconnection of the mobile units was key, and we foresaw standard techniques, such as randomized or epidemic algorithms, being adapted to help adapt to connectivity changes.

Systems: Applications. In our 2000 paper, we also focused on the applications envisioned back then, with the belief that

> *the most visible impact of software engineering research will be in the wide range of applications expected to emerge on the market*

At the time, continuous connectivity to the wired infrastructure simply did not exist, or came at too high a cost. Therefore, we foresaw a need for systems to address the needs of individual users and groups without infrastructure access. For the individual, access to resources, such as file systems, without relying on persistent communication was important. Caching and hoarding schemes offered one solution, but remote query execution also offered promise, as it limited the amount of data transmitted over expensive wireless links. Without the fixed network, we also foresaw groups of users with localized connectivity forming ad hoc communities. These groups, formed by students or conference attendees, could exchange business cards, session notes and schedules without any infrastructure support.

While this technological limitation implied one direction for applications, novel technologies opened new domains. For example, sensors combined with wireless communication could be used to measure the environment or, more generally, detect context changes. Combining GPS information with wireless communication offered opportunities such as long-lived collaboration among vehicles traveling in the same direction or data sharing among passing vehicles. Further, wireless kiosks, themselves attached to the fixed network, could exchange local information with passing cars, exploiting physical co-location and wireless capabilities.

Across these novel, mobile applications, we identified several cross-cutting issues. First, the degree of mobility awareness exposed to the user is key. In some applications the user is involved in decisions that affect functionality, e.g., specifying files to be cached. In others, the system transparently adapts to changing conditions, e.g., reducing video quality when changing networks or preventing access when the security domain changes. Further, applications are also shaped by mobile device capabilities as well as infrastructure access and support, which may limit interaction due to a small screen size or force only localized interactions when access points are not available.

Systems: Middleware. Middleware offers support for application development with new mechanisms situated on top of the operating system. In our original paper, we distinguished between logical mobility middleware as a new tool supporting the related novel design paradigms, and physical mobility middleware as supporting a new set of application and system requirements.

Support for logical mobility in a middleware focuses on programming language extensions of an existing programming languages, most commonly Java, for mobility of code and state. At the time, code on demand through the dynamic loading of Java classes was already integrated and available in the mainstream. However, abstractions and supporting mechanisms for rebinding strategies and architectural styles had not been addressed.

As for physical mobility, we argued that:

> *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.*

These challenges required bridging the gap between low-level information such as battery level, communication quality, neighborhood, and location, and a more abstract application developer interface. Accomplishing this in a general way was a challenge we foresaw for context-aware middleware with service discovery and service support, e.g., with message delivery adapting to disconnections and changing locations.

At the time, we saw coordination as one concrete possibility to both bring together the worlds of logical and physical mobility and to serve as a concrete, formal foundation for the middleware supporting both types of systems. For example, tuple spaces can be remotely accessed by mobile and non-mobile units, offering a single, abstract access mechanism to system data, including context information. Alternately, the data contained in tuple spaces can represent code to be accessed or moved throughout a distributed system. To make such state-based abstractions practical in dynamic environments, additional elements were being considered, e.g., enabling reaction to changing state, and thus event-like interfaces.

2.3 What Happened to Them?

Now that we have reminded ourselves about what mobile computing meant at the end of the last millennium and the challenges we identified at the time, we are in the position to look back critically at our considerations.

We structure this section by revisiting what we stated in 2000, with the obvious advantage of hindsight. Hereafter, we provide a concise account by focusing first on the challenges and dimensions of mobility we believe are still relevant, then on those dimensions we overemphasized, and those we underemphasized or even missed. Table 1 provides a quick, at-a-glance summary of how the relevance of the macro-challenges we outlined in [1] and in Section 2.2 shifted over the years.

What is still relevant? We believe the distinction between physical and logical mobility, along with their different roles—the former posing new *requirements*, the latter being essentially a *design* choice—brought some "order" in the research field at the time. The unified presentation of these concepts as characterized by unit of mobility, location, and context, is still valid, at least from a conceptual standpoint, although it did not generate the impactful modeling insights we advocated in the paper. On the other hand,

Table 1: Macro-challenges for mobility: now and then.

Models	$\downarrow$	Models of mobility, albeit intellectually stimulating, are today much less relevant than we expected.
Algorithms	$=$	Algorithms are still fundamental, although the concern shifted greatly from the mechanics of mobility and disconnection towards directly supporting users and harvesting the data they generate and gather.
Applications	$\uparrow$	The number and breadth of applications exploded since 2000, and the trend is still going strong, effectively (re)defining the challenges for software engineering.
Middleware	$=$	Middleware is still a key element of the picture, enabling rapid development, although the type of primitives required are shifting towards the user, similarly to algorithms.

the aforementioned central role of location and context is reflected in the system challenges we considered, applications and middleware, discussed next. Context modeling remains a key (and largely open) challenge for mobile systems today, though efforts continue to strike a balance between expressiveness and usefulness of such models [5, 6].

As for applications, we were well-aware they were the driver for the users and the market, ultimately determining the success of mobility, as well as shaping the research challenges involved, to the point that we argued for

> *a need to consider a style of research that is much more application centered than in the past.*

Today, this reasoning holds even more strongly, given that the application landscape is much richer and diverse than we could imagine back then; we further elaborate on this aspect in Section 4, where indeed we are faithful to our quotes above and outline research challenges along classes of applications. However, as already mentioned, in our original paper we also highlighted the importance of location as a first-class object, directly available to applications through the use of GPS devices. Still somewhat exotic (and utterly expensive) at the time, these devices are nowadays cheap and omnipresent and at the core of a new wave of applications relying on different uses of location, from geo-tagging to social networking. Another aspect we foresaw is the emergence of specialized computing devices enabling fine-grained sensing, which is today becoming more and more a reality, e.g., thanks to research in the fields of wireless sensor networks and, more generally, pervasive computing. In these realms, the aforementioned notion of context will gain increasing importance if the scenarios enabled by distributed sensing or, more generally, by the "Internet of Things" and "cyber-physical systems" materialize.

This last angle brings us to middleware, which is still a key challenge in enabling mobile applications. Today's mobile platforms provide rich libraries, but mostly focus on abstracting the *device* features, such as on-board sensors and access to the network. Properly managing *distributed* context, despite the conspicuous literature on the topic, is still central to mobile applications, for which ad hoc solutions are often employed and no prevailing approach has appeared. In part, this is also a consequence of the dominating invocation-based paradigm, originally incarnated by RPC and distributed objects and today revamped by service-oriented architectures, whose tight coupling is intrinsically at odds with the fluid, dynamically-changing context enabled by mobility. In this sense, our original emphasis on *coordination* as the "glue" among mobile components still stands as a challenge:

> *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.*

Interestingly, while it is true that this view is somewhat marginal in the mainstream, which is dominated by the invocation-based approaches inherited by distributed computing, the importance of coordination has grown in other fields related to mobility, such as pervasive computing and wireless sensor networks. For instance, in the latter field, many programming approaches rely on some form of data sharing as the primary means of communication [7]. The TeenyLIME [8] system developed by some of the authors, a tuple space middleware inspired by LIME [9], has been used successfully in real-world wireless sensor network deployments [10, 11] where it provided not only a significant reduction in the development effort, but also in the dimension of the resulting binary code—a very desirable property for these resource-constrained applications.

On the other hand, we did overemphasize the role of algorithms for coordination (e.g., supporting leader election, transactions, and the like) of mobile entities, along with other issues we discuss next. In our paper, we saw algorithms as a necessary element enabling the coordination mechanisms we envisioned mobile applications to be built on. Since coordination has still a relatively marginal role, their algorithms are not as central as we speculated. However, this is not to say that algorithms at large lost importance: on the contrary, they are becoming key, but above and below the "system waist" represented by middleware. Below it, algorithms for optimizing low-level system concerns (e.g., power consumption), possibly in concert with the application goals, are more and more important, as we further elaborate later in this section. Above it, algorithms for automatically mining and exploiting the wealth of "big data" harvested by mobile users are rapidly becoming one of the most exciting challenges of future computing, as discussed in Section 4.

What did we overemphasize? Looking back at the focus of the research community at that time, mirrored in our paper, it is evident that the emphasis we placed on models has not been matched by a corresponding impact. Formal models of mobility were definitely a popular topic when we wrote our paper, and there was a lot of expectation about their contribution to elucidating the underpinnings of mobility and consequently guiding the design of the corresponding systems. As we already mentioned, not all of these efforts are in vain, and some challenges (e.g., proper modeling of context) are still standing. Nevertheless, one reason why models did not have an impact is that they were detached from reality and practical use. For instance, the vast majority of models (e.g., Mobile ambients [12] and other derivatives of π-calculus) assumed the process or thread as the unit of mobility. This admittedly rather natural modeling decision leads to nice and elegant theories. However, on one hand it does not directly model the independent relocation of code and state, e.g., typical of the (successful) code on demand paradigm. On the other hand, these theories assumed movement in a logical space, ruling out the gory complexity stemming from fluid topology of wireless communications, and therefore precluding a direct application to physical mobility. Therefore, in practice these theories were useful only to model mobile agents; since this paradigm is today relegated to irrelevance, so are the corresponding models.

The last statement also brings us to comment about logical mobility, which was also a very active topic at the time, especially concerning mobile agents. In 2000, the census in the Mobile Agent List, maintained by Fritz Hohl at the University of Stüttgart, contained 72 systems. As noted elsewhere [3], if one assumes 1994—when Telescript [13], the first mobile agent, and Java appeared—as the birth year for mobile agents, this means that these systems were produced at the rate of one per month! Interestingly, the vast majority of the implementations relied on Java, which natively supports only a very constrained incarnation of mobile agents, supporting weak mobility, and therefore only very few systems were a real advance to the state of the art. As a result, the relative ease with which mobile agents could be implemented, combined with the aforementioned popularity and slant of mobility models, fueled each other and ultimately spiraled both into oblivion.

While it is true that logical mobility is no longer a hot topic, this does not necessarily mean that it is irrelevant. On the contrary, the code on demand design paradigm is very much alive, to the point that it is such a consolidated reality that it no longer deserves a place at the forefront of research. A great deal of today's dynamic Web content technology revolves around this paradigm, enabling dynamic loading and extension of applications. Mainstream middleware has support for some form of code on demand, e.g., the class loading mechanism in Java (e.g., in RMI and OSGi) and assembly loading in .NET. Therefore, in a sense, we overemphasized mobile agents because we did not predict their demise, and code on demand because we did not predict its success.

Finally, despite our emphasis on applications, those we envisioned were obviously constrained by the dominating scenarios at the time, which are not necessarily those that emerged. For instance, we put a great deal of emphasis on mobile ad hoc networks (MANETs), a very popular topic of research at the time. This radical form of mobility was particularly intriguing due to the uncompromising role of decentralization. This vision also implied a key role for managing disconnection—a fact of life in the fluid and dynamic topologies defined by these networks. For better or worse, the world evolved in a different direction, as we further elaborate upon in Section 3. Ubiquitous (and cheap) connectivity is now a reality, which relegates MANETs to the niche where they were originally conceived, i.e., military applications and disaster recovery. Similarly, without the continuously changing context of MANETs, the complexity of dealing with disconnection is nowadays reduced to the problem of "synchronizing" data with some server "in the cloud", accessible while on the move.

What did we underemphasize? On the other hand, there are a number of issues that we overlooked in our paper, and that probably we could have already anticipated at the time. In hindsight, some aspects of mobile computing should have been included in our original vision in 2000; we discuss those here before Section 3 discusses the more radical and largely surprising "game changers" that have occurred since.

One aspect we barely touched is the one of security. In relation to logical mobility, mobile code poses an intrinsic security concern. However, the degree of complexity security introduces varies greatly among the various design paradigms, and ultimately determined their fate [3]. At one extreme, the challenges posed by code on demand could be solved by and large by existing, well-known techniques, such as certificate-based schemes and sandboxing, and that is one reason why this paradigm still thrives today. At the other extreme, the security challenges posed by mobile agents were unprecedented, and the solutions often limited the very flexibility provided by this paradigm. Combined with the intrinsically higher implementation complexity and the absence of a clear, far-reaching application case, the security challenge eventually relegated mobile agents into irrelevance. As for physical mobility, security, and especially, privacy are very important challenges today. Interestingly, these issues can be cast into the "model" of our original paper by observing that the threats to privacy are generated by the location and context associated with the unit of mobility (the mobile user in this case) and the ability to control when and how they are shared with others. The tradeoff between the degree of control retained by the user and the effort necessary to exert such control is still a largely unresolved challenge, whose burden ultimately percolates to application design and therefore to software engineers.

Another angle we definitely overlooked is the importance of system concerns and algorithms. While we touched upon the importance of power management and similar issues, we also somewhat dismissed them as not essential:

> *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.*

It is indeed ironic that all of the authors of this paper had subsequent experience with wireless sensor networks, smartphones, and other power-constrained systems, and learned the hard way that energy constraints are essentially inescapable. Technology did not progress as we expected and, on the other hand, miniaturization continuously pushes the envelope of what one can do with only a minimal energy supply. Energy is only one of the system issues one should consider: the vagaries of wireless connectivity—the key enabler of mobile computing—is another aspect that can only partially be dismissed as a low-level issue, germane to other communities. The truth is, we believe that this attitude (which we shared at the time) is one of the reasons why mobile and pervasive computing, in all their nuances, have progressively disappeared from software engineering venues, as we further elaborate in Section 5.

3. GAME CHANGERS

Several aspects from the original paper did not unfold as we had foreseen for reasons that were largely unexpected. These disruptive events have in many ways dramatically changed both the perspective and direction of software engineering for mobile computing. In this section, we identify and explore the most significant of these events before turning our eyes in the next section to peer ahead once again at what may be just around the bend, including how these game changers may impact the future course of the field.

Omnipresence of smart devices. In our original paper, we played down elemental aspects of the devices comprising the mobile computing space. While we foresaw the extreme miniaturization of computing elements (to the point that they would be integrated in clothing, buildings, and vehicles), our focus was on the mobility abstractions for these components. What we did not foresee at the time was the degree to which the nature of the devices themselves would influence what mobile computing ultimately entailed. In 2000, mobile computing focused largely on the physical mobility of traditional computing devices (bulky laptops and first generation PDAs). Not many predicted the explosion in smartphone penetration; in fact through the year 2005, smartphone penetration globally remained at or below 1% and, in 2005, only the Western European region exceeded this figure with a 4% penetration[2]. The smartphone markets have since exploded, with major ones tip-

[2]Source: Gartner IT Hardware Survey, 2000-2010.

ping past 50% penetration[3] in 2013 and global penetration nearing 30%. While mobile tablets lag smartphones in time, the initial growth rates for tablet penetration are even steeper than the initial growth rates for smartphones. The rapid growth of these devices has resulted in an associated explosion in software tailored to these devices and a plethora of new software engineering concerns brought about by these platforms, from the need for applications to be energy-aware and energy-conserving to novel modes of human-device interaction to the need for expressive approaches to context- and situation-awareness.

Sensing on the small and cheap. Alongside this explosion in the availability of highly capable devices in ordinary users' hands came a raft of advances in the hardware technologies they embed and interact with. While our first paper did foresee the emergence and importance of miniaturized sensing [14], a divergence from our expectations in 2000 was the degree of integration of these capabilities with everyday technology, i.e., smartphones. Significant advances in manufacturing processes for micro-electro-mechanical systems (MEMS) have resulted in reduced costs and increased quality; today, mobile computing devices are a driving factor for the MEMS market[4]. The traditional use of MEMS in smartphones has been for accelerometers and gyroscopes, allowing detection of shakes, rotations, and navigation. However, MEMS have also become essential for supporting other types of on-device sensing (e.g., humidity and thermal sensing), improved sound and video quality, better communication performance, and increased battery lifetime. These new hardware advances demand a commensurate shift in support for integrating these new capabilities into applications, and recent research in software engineering for these spaces has shown an increased focus on handling these capabilities. Dandelion [15], for example, connects smartphone applications to wireless body sensors through programming abstractions called *senselets*, while Open Data Kit Sensors [16] provides Android programming abstractions for integrating data from both sensors internal and external to the smartphone.

This miniaturization of computation, communication, and sensing also gave rise to remarkable developments in the interconnection of tiny sensing and actuating devices, dispersed in the environment. Wireless sensor networks are arguably the most prominent technology in this respect, born at approximately the same time as we were writing our original paper. Although wireless sensor networks received little attention from the software engineering community [17], they are nowadays the cornerstone of pervasive computing at large, including recent popular trends like the Internet of Things and cyber-physical systems. The connection with mobility is provided not only by the many scenarios where sensing and actuating nodes are themselves mobile (e.g., placed on humans, animals, robots, or vehicles) but also by the scenarios where these nodes, albeit fixed in the environment, provide key information to a user's personal devices.

Ubiquitous connectivity. All of these device innovations have given users the availability and expectation of ubiquitous and persistent connectivity to the Internet. Even more recently, the advent and popularity of cloud resources have connected this "always on" mentality to one of virtually unlimited remote storage and computational resources. In just a few years time, we have arrived at a point where we no longer even consider whether a resource we are using is "local" to our machine or resident in some amorphous "cloud" that we assume is persistently available. The notion of being "disconnected" from such resources has gone from a fundamental concern in 2000 to something that we rarely even consider. The key point is that the advent of the cloud fundamentally changed the way we think about the architectural structure of applications, enabling innovations in how we engineer systems that integrate mobile devices with the cloud *and* in how we engineer systems that integrate mobile devices with each other.

A social world. Finally, no discussion of the game changers in mobile computing over the last decade would be complete without a mention of social networks.

In our 2000 paper, we discussed social connectivity somewhat obliquely, connecting coordination protocols with the requirement to support open mechanisms to make mobile users aware of the *physical* presence of others. Clearly, social networking has evolved way beyond this simple awareness, inserting itself into every facet of our mobile interactions. In the applications available today, our mobile devices use social network information to connect us to locally present friends and acquaintances. At the same time, sharing of information mainly happens through a remote server and not through ad hoc connectivity. For example, Facebook had[5] 874 million monthly active users that used mobile products as of September 30, 2013. Therefore, social connectivity is no longer concerned solely with physical contacts, but also—and, actually, predominately—with distant connections. Nonetheless, the role of location is still fundamental. Users are increasingly sharing location-aware information and data (such as geo-tagged photos) with people not co-located with them, leading to an increasing interest into geo-social systems [18], which combine features related to social connections and geographic information.

4. PEERING AT THE FUTURE

This section takes the context of the game changers in the previous section and attempts, once again, to look forward, both at current and emerging trends and our expectations for the future. In today's world, mobile computing is simply a fact of our everyday lives. We use our mobile phones to access services (e.g., mass transit trip planners) and to generate content (e.g., taking photos or writing tweets). Through our mobile devices, we further generate a sort of "digital breadcrumbs," which can be exploited to build applications and systems that were unthinkable just 10 years ago. In this section, an important cross-cutting theme is the need to deal with the increasing complexity of the technological scenarios; for this reason, the middle of this section rallies around some specific emerging scenarios, highlighting their connections to traditional and fundamentally new challenges in software engineering.

4.1 Trends

Here we discuss the trends that are shaping the future of mobile computing, outlining the challenges for the software engineering community.

Mobile sensing. Enabled by the sensing on the small and cheap, developments in mobile sensing have been tremendous in recent years [19]. Stemming as a natural evolution of the last decade of work in wireless sensor networks [20] and smart spaces [21], efforts to personalize mobile sensing and bring it to commodity mobile devices are only a handful of years old [22–24]. This availability of mobile sensing capability has already engendered powerful and readily available libraries that are part of standard software development kits for common mobile platforms [25].

Advances in mobile sensing have deep impacts and implications on software engineering, both in terms of the novel capabilities en-

[3]Source: eMarketer, May 2013.

[4]Source: Yolé MEMS Market Report.

[5]Source: Facebook: `http://newsroom.fb.com/Key-Facts`.

abled and in the complexities created. These capabilities and complexities are perhaps most obvious in the area of digital behavior intervention systems [26]. These applications "close the loop" by not only collecting data from mobile phones but also providing effective feedback to the user according to his current personal state and social context. Middleware, algorithms, and software architectures for supporting the flexible development of these context-aware and adaptive systems are emerging as interesting research challenges. This new complexity intimately intertwines novel software engineering contributions with human-computer interaction concerns, control theory, and machine learning.

Building software that integrates mobile sensing retains existing mobile software concerns, including those related to resource constraints and intermittent connectivity. However, as the capabilities of our mobile devices increase, processing power and memory are no longer critical issues. On the other hand, energy resources are still a critical concern, especially given today's energy-hungry devices with their large, high-definition screens and powerful processors. These energy concerns are rapidly creeping into the software engineering process [27, 28].

As capable and effective as our mobile sensors are becoming, the data is often messy. Another current focus in this space, therefore, is the development of algorithms and components that help applications and their developers to process noisy data in a robust way. These approaches commonly apply machine learning algorithms, whose outputs are characterized by a degree of uncertainty. Software that incorporates this sort of result cannot be deterministic in the classic sense. This has strong implications on the software testing community, as new methods are needed that can account for these uncertainties. In this context, the recent advent of *probabilistic programming* [29] may hold promise.

Mining of mobile "big data". Given the pervasiveness of mobile and sensing devices and the fact that these devices are almost always on, it is possible to collect vast amounts of data from and about mobile devices (and their users) in real-time [30]. Analyzing this data can give detailed information about people's behavior, either individually or collectively [31, 32]. The potential uses for this data are many, with wide reaching implications on everyday aspects of the lives of millions of people. A typical case is the use of mobile data for improving transportation [33]. Using mobile phone data, we can reconstruct and quantify social interactions in groups and communities, leading to new fields of computational social science [34] and social computing [35]. Systems for mining big data from mobile devices may play a particularly important role in developing countries, where no governmental and health infrastructures are present. In these countries, systems based on mobile phone data analysis can be used to improve basic services, such as mass transit planning [36] and disease prevention [37].

The implications of this trend on software engineering are more open-ended and range from the emerging new needs for defining the requirements of these applications to testing such a highly decentralized system whose emergent behavior is difficult to characterize *a priori*. The data amount is on a scale that existing software engineering techniques have not yet tackled. The software engineering process must allow for flexible extensions as new types of data become available. Another requirement is the support for managing relations and links in data, like in the case of the Semantic Web [38]. Privacy is clearly a fundamental issue, and software design techniques must consider the problem of data privacy. One possible approach is based on the privacy-by-design principle [39].

Location, location, location. Thanks to the availability of inexpensive GPS receiver chipsets, the vast majority of today's mobile devices seamlessly enable outdoor localization. Current efforts focus on robust technologies and techniques for supporting indoor positioning [40], enabling new classes of applications, e.g., in retail and in the assistance and care of the elderly or people with disabilities [41]. Interestingly, even applications that do not exploit location directly (e.g., games) collect location information for analyzing and/or re-selling the data for marketing and advertisement purposes. More generally, location is an essential component in almost all of today's mobile applications [42–44], as we anticipated in our original paper.

Nevertheless, a principled approach to the design, implementation, and validation of location-aware systems is still missing. Applications must be able to deal with location values at different levels of precision (e.g., due to the quality of the GPS signal) or granularity (e.g., due hardware limitations or to intentionally protect the user's privacy).

Off-loading and opportunism. As described previously, the persistent availability of the "cloud" has fundamentally changed mobile computing. While precursors to computational clouds have existed since the beginning of computing, the integration of this cloud with persistent connectivity has given rise to many opportunities in *offloading*, most commonly moving heavy computational loads from lightweight mobile devices to the cloud [45–47], but also in offloading sensing tasks to a persistently connected ambient infrastructure [28]. These directions have even come full circle to offloading to opportunistically available mobile partners [48], hearkening back to some of our original predictions regarding uses of mobile ad hoc networks. These trends are a current focus of the mobile and pervasive computing communities; we expect attention to continue focus on how to intelligently and adaptively merge local and infrastructure resources for the maximal benefit of applications.

Wearable computing goes mainstream. In recent months, wearable computing has gone from a mostly niche science fiction vision to a commercial reality. Highly capable smart watches (e.g., the Samsung Galaxy Gear[6]) are available on the market and several companies are exploring opportunities in the space, including e-textiles [49] and smart fabrics [50]. One of the most famous examples is indeed represented by Google Glass[7], a device with an optical head-mounted display connected to the Internet and able to process natural language voice commands.

Although designing for heterogeneity has been a recurrent research theme for the software engineering community, the emergence of this new class of systems makes the need for principled techniques for development of cross-platform and cross-device software increasingly urgent. For example, there is a real need for effective Integrated Development Environment (IDE) support for the development of applications that can be seamlessly ported to different platforms and devices. Wearable computing systems push the envelope of hardware/software co-design (discussed next) as these systems must be aware of both the environment and user simultaneously. Further, the extremely limited form factor of wearable devices renders their capabilities quite constrained, emphasizing the aforementioned need for an efficient and effective balance between offloading and local computation.

Further, the context in which applications will operate is nearly impossible to completely predict *a priori*, in terms of connectivity, user activities, and social interactions. This poses significant challenges, especially for applications that are based on sensing and processing contextual information, such as those based on activity

[6] `http://www.samsung.com/uk/consumer/mobile-devices/galaxy-gear/`

[7] `http://www.google.com/glass/start/`

recognition algorithms. For these reasons, software engineers have to deal with truly open systems, which must be at the same time aware of, and robust against, the unpredictability of both the users and the environment.

Blurring the boundary between hardware and software. While perhaps not specific to mobile computing, the current trend to make the distinction between software functionality and hardware functionality transparent has significant impact on the landscape of mobile computing. Increasingly, functions that were obviously in the realm of software are being pushed to (potentially specialized) hardware components. This is particularly true for computationally demanding algorithms such as those for learning and signal processing (e.g., for activity recognition). The implementation of these functionalities in hardware enables better performance in terms of power consumption and processing speed.

Exciting emerging efforts are taking advantage of these blurred lines both within mobile applications and within supporting capabilities for developing these applications. One interesting example is Consia[8], which learns when to switch on and off sensors and radio interfaces by learning user behavior. Looking forward, applications that can creatively navigate this new and murky boundary may be able to provide richer functionality more efficiently, offering up new challenges in the space of tools and techniques to support their entire software life-cycles.

Mobile first, then desktop (maybe). We are witnessing a remarkable economic (and, in a sense, cultural) change in the development process of mobile applications. Until very recently, applications were developed first for desktop computers and then "ported" to mobile devices. Now, the opposite is increasingly common; mobile applications come first and are sometimes never even followed by desktop applications. This is particularly true, obviously, for location-aware applications. Sometimes, a Web portal complements the mobile app, but it does not provide all functionalities of the mobile version, as for instance in Foursquare[9].

Further, a platform-agnostic approach to applications is being increasingly pursued, where good software engineering fundamentals enable an application's code base to be largely ignorant of the target device. This trend is particularly visible in the development of Web-based applications, where the end user may not even observe the natural, behind-the-scenes application adaptations that occur to map the Web application to the target platform or browser. HTML5, the most recent version of the standard, offers features that foster the natural integration of mobile application development, such as support for offline Web storage [51] and a Geolocation API [52]. HTML5 has also been regarded as an enabler of interoperability, as one of the major issues in mobile applications today is supporting cross-platform development [53, 54].

A market for your app. Perhaps one of the most surprising current trends is the emergence of online marketplaces (e.g., the Apple App Store, Google Play, and the Microsoft Store) for distributing mobile software. For the first time in the history of software development, single developers or small companies can access distribution infrastructures that allows them to sell (mobile) applications to millions of potential customers at the tap of a finger. This is also driving new economic models, also in relation to the pricing of applications [55]. Prices can usually be kept very low (a few dollars) given the economy of scale enabled by these online distribution platforms. Marketplaces also allow pushing updates to mobile applications in a seamless way, which enables a fast deployment cycle and a tendency to release applications that are frequently updated, following comments and requests of users sometimes made through the "stores" themselves. Another interesting trend is the release of free applications that provide a set of basic functionalities, which can be enhanced or expanded through a subscription, in the so-called *freemium model*. As these trends evolve, we can easily envision macro applications that assemble smaller components into specialized functional units (e.g., a home entertainment control center, assembled out various home automation components or a hospital service application, assembled out of multiple vendors' device-specific apps). This trend, combined with the previous one about mobile development, are essentially disrupting, once again, what we know "by-the-book" in terms of software development practices and software economics.

[8] `http://www.qualcomm.com/media/videos/consia`

[9] `http://www.foursquare.com`

4.2 Challenges (and Opportunities) Ahead

The considerations in our original paper revolved around a delineation between theory and systems. Today, we see a more blurred view, where sytems and applications issues have become predominant, and models have only a supporting role, if any.

Indeed, the future of mobility will be characterized by an increasing mixing with other fields both inside computer science (e.g., machine learning, data mining, human-computer interaction) and outside of it (e.g., cognitive science, psychology, sociology). Further, as we already mentioned several times, it is likely to be dominated, once more, by the flavor of applications that will emerge. Therefore, we chose to cherry-pick some of the application scenarios that, in our opinion, are most representative of challenges and opportunities that lay ahead of us. Although these scenarios are already discussed here through a software engineering lens, in Section 4.3 we distill some more general considerations on the impact the challenges we outline here have on software engineering at large.

Anticipatory computing: Using your mobile as a crystal ball. Anticipatory mobile computing [56] builds on our current capabilities to sense, model, and even predict context to construct algorithms and mechanisms that enable automated decision making processes based on these sensing and prediction capabilities. Anticipatory computing applications have long been the holy grail of mobile and pervasive computing.

> *As you prepare at home for your morning run before work, your smart glasses provide you with a weather forecast for the next hour. An accurate training plan is also automatically generated for you, based on your estimated performance (extracted from the logs of your previous runs and competitions in the previous month) and physical indicators collected by wearable biomedical sensors. Since the readings of the sensors show that the previous training sessions were rather stressful for you, the planner indicates a very flat running course for today's run. Since a friend of yours who lives nearby is also headed out for a run at the same time, a message is displayed in your glass, and the device asks you if you want to call him to meet and run together.*

Anticipatory mobile computing is not limited to personal assistants like this one but also includes visions related to smart healthcare and smart cities. Anticipatory computing scenarios are commonly used to motivate research contributions or services that generally solve a very specific piece of the puzzle, for example predicting relevant apps [57,58], prefetching relevant information [59] mining personal context, or context-adaptive route planning. While such application scenarios seemed like science fiction in 2000, the

components are largely realized in today's landscape. The potential of the combinations have even been envisioned, for example as the *cognitive phone* [60]. Yet the integrated applications remain unrealized at any real scale. Anticipatory mobile computing at scale will require supporting collective sensing and prediction, further exacerbating concerns related to privacy and resource constraints. However, the trends in big data described in the previous section are a major enabler of these anticipatory applications, and we expect that this availability will continue to drive their evolution. Given the current state of the art in the supporting research, realizing anticipatory mobile computing has reached a critical stage that demands principled software engineering.

These anticipatory mobile applications must take advantage of a wide array of potentially intensely personal information, including behavior patterns and intents, emotions and moods, interpersonal relationships, etc. While much focus in mobile computing has recently promoted *off-loading* computation to a highly available infrastructure, enabling privacy-preserving computation may require an intelligent approach to *on-loading* [61] computation from surrounding and wearable sensing devices on our mobile personal ones (e.g., our smartphone), demanding a reinvestigation of the co-design of their hardware and software platforms.

Anticipatory mobile systems must integrate both on-device hardware and software capabilities with infrastructure hardware and software capabilities in a manner that is fluid, flexible, and aware of resource limitations. Many existing middleware for mobile computing focus on context-awareness and reflection as adaptation-enablers. However, the needs of anticipatory mobile computing applications are well beyond the mainstream of today's software engineering visions for self-adaptation [62]. We contend that anticipatory mobile computing will be a defining disruptive technology for the engineering of software systems that dynamically adjust their architectures in response to a wide variety of context factors.

Testing anticipatory mobile computing applications demands novel approaches that function *in situ* based on an individual user's execution. Application function is tailored to an individual user and his context; not only some of their combinations may not have been part of the testing suite, but they may not have even been envisioned or possible during testing. Further, the true anticipatory mobile computing vision is not one of multiple siloed apps but one of a single integrated system with varied functionalities that share data, resources, and tasks and ultimately work together to provide a complete experience tailored to an individual user. From a traditional software engineering perspective, such a vision is frightening and necessitates an ecosystem with a myriad diverse players, from the device and software developers through the functionality and content distributors to the user and his data.

I, Mobile: Mobile systems for robotics. The past few years have witnessed an explosion in autonomous robotics, although still either in the hands of tinkerers or in very specific (mostly industrial) applications. Given the maturing hardware and control capabilities, software now stands between these niche applications and a more wide ranging vision of mobile robotics systems.

> *As the site supervisor for a busy urban construction site arrives in the morning, one of the site's fleet of quadcopters delivers his morning cup of coffee (without spilling it!), which he drinks as he reviews the project's progress on his smartphone. The quadcopter is quickly summoned away, however, as it is needed to provide aerial views as a crane is shifted from one location to another. Quadcopters also coordinate with each other, with sensors embedded on the site, and with devices carried by workers to determine where, when, and how to safely deliver materials exactly when needed. The quadcopters dynamically form teams as the tasks demand; a single quadcopter can deliver small items (e.g., a cup of coffee, a handheld tool), while multiple quadcopters have to work together and with a human team to deliver palettes of material.*

This scenario may be a bit sci-fi, but while we were writing this paper both Amazon and Deutsche Post announced their tests and plans for package deliveries supported by quadcopter drones. Automation, almost entirely in the form of autonomous robots, is changing the way we live and work. It is likely that, over the next few decades, mobile robots will take over many tasks commonly performed by humans, from the mundane to the more intellectually taxing [63]. Fundamental research is already in place to support this, from the sophisticated hardware [64] to algorithms for control and coordination of individual robots [65] and groups of robots [66]. Today's solutions and demonstrations, however, are almost entirely custom built, while the revolution that we anticipate will require general-purpose components and approaches that enable rapid development and assembly of complete systems. Further, while the fundamentals in artificial intelligence that underpin many robotic coordination and control algorithms can be very sophisticated and are often successfully demonstrated in simulation using large number of autonomous components [67], moving these same algorithms into real robotic systems brings significant additional constraints, which result in severely constrained implementations [68].

Just as software engineers of the future will need to be able to target applications to the smart personal mobile devices described in the previous section, they will also need to be able to build software that can control fleets of robotic systems. Enabling the development of these novel applications will require new tools and techniques supporting the complexity arising in this domain.

Coordination is clearly a basic necessity of robotic systems, both internally, as the robots coordinate with each other, and externally, as the robots coordinate with humans. The former was a key focus of our vision in 2000:

> *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.*

Algorithms for control of fleets of mobile robots have received significant attention over the last decade, but these efforts have not, to date, resulted in *primitives* for more general purpose programming of coordination activities. Coordination of robots with humans is also gaining attention; this coordination must go beyond simple game console-style remote control to include natural and intuitive interactions [69], again in ways that are simple to integrate into a system design.

Enabling these more complex forms of coordination also requires better representations of the various aspects of the state of the world *and* the ability to adequately share these states among the coordinating partners. This challenge is in fact exacerbated when considering the integration of a human in the loop; the human may make assumptions about the state of the world that the robot could invalidate, given the proper abstractions to communicate its accumulated knowledge about both the state of the environment and its own internal state [69]. Indeed, another emerging trend is represented by

autonomous vehicles, such as Google self-driving cars [70], which constitute a new class of self-adaptive systems that interact in unknown environments with high degree of uncertainty and, at the same time, with very strict safety requirements.

Finally, it is well-known that verifying and validating these complex and interoperating robotics and, more in general, autonomous systems in simulation or even in controlled environments is not at all reflective of their performance "in the wild." The unexpected and unpredictable situations that will arise require both developing for uncertainty and being able to learn about the correctness of the implementation *in situ*. In the former case, the software (and to a more limited extent, the hardware) present on these systems must be able to self-organize and adapt to changing conditions with a degree of flexibility that existing self-adaptive middleware do not support. Recent work has headed in this direction [71], but, again, these efforts opt for making a very specific application work instead of investigating the design primitives and programming constructs that will be needed to support robotic adaptation in general.

Neural interfaces: Your mobile and your brain get together. Recent advances have enabled brain-controlled prosthetic limbs, i.e., *neural prosthetics* [72], related to the previous scenario. However, more recent visions of neural interfaces go even further.

> *During the morning lesson at a primary school, an eight year old boy sits quietly with his smart tablet at his desk. His severe autism prevents him from interacting with his peers and his teacher through traditional means. An inconspicuous set of dry electrodes embedded in a headband the boy wears captures EEG signals that allow him to interact with his tablet; apps on the tablet help him form relevant conversations, and he can respond to the teacher's questions to the class and his friends' social interactions in real time. During a particularly frenzied and active lesson, the teacher receives a quiet notification on her personal device that the boy's level of frustration is rapidly rising, allowing her to naturally intervene before he loses attention or his frustration turns into an outburst disrupting learning.*

Brain Computer Interfaces (BCI) are gaining attention in a wide variety of domains, from providing communication aid for autistic people [73], to monitoring and managing an individual's cognitive load, especially in the context of dangerous working conditions [74]. In both cases, the goal is enabling behavioral intervention or augmentation, using a variety of computational modalities available, though most often through "smart" devices. As "far out" as these examples may sound, the technology exists to support them. For example, using the NeuroPhone and a non-intrusive cap made of dry electrodes, a user can "think" a smart phone through the dialing process [75]; a similar cap can differentiate a wearer's emotional responses [76].

A major challenge in BCI is that the desire to perform non-intrusive neural monitoring (for obvious reasons) results in very low quality and low bit rate information. These signals must therefore be aggregated and fused with other context that characterizes the state of the user and the environment before using it to influence an application's behavior. Middleware constructs are necessary to support this acquisition and aggregation, which will span multiple devices, both human-borne and embedded in the environment. Perhaps more importantly, middleware must also offer constructs that enable an application to internalize the *quality* of acquired data and to tailor its responses based on the relative quality of the information.

Cleanly and clearly specifying such systems that intertwine the brain and mobile devices is not straightforward. Existing assets in the software engineering toolkit are not prepared to deal with the predominance of the non-functional requirements in these situations nor the fuzziness of requirements that this integral human element entails. That is, even when a system has an obvious and easily stated "goal," exactly stating or identifying "correct" or "incorrect" behavior may be, at best, subjective, and at worst, impossible. The theme of "fuzziness" continues in the verification and validation of these systems. Again, *in situ* approaches must prevail, but the validation of a system that is so dominated by non-functional requirements must itself tolerate fuzzy results. Validation approaches here, then, must match in nature to the specifications of these systems.

The future of (mobile) privacy. Privacy is a key element of the design of any mobile system. Indeed, mobile applications increasingly collect, store, and share highly personal information. In fact, as discussed above, the recent progress in mobile sensing has enabled new applications that we can define as people-centric [30], i.e., built around the daily life of people. Most of today's applications collect privacy-sensitive information, in particular location information [77]. Another issue is related to the fact that, increasingly, sensed data are used to extract information about the future behaviour and context of the users (such as future location) through machine learning algorithms. Privacy issues are not just related to the data itself, but also to the additional information that can be extracted from it [32]. Additional concerns are related to the visualisation and sharing of this privacy-sensitive information.

> *After cycling to your office, you sit at your desk. The wall on your right displays information related to your calorie consumption and your carbon footprint, collected by your phone and other wearable devices. Information about your average heart rate and other health indicators are also shown. When one of your colleagues enters the room, the data items related to your physical activity are automatically removed from the wall since they are considered privacy sensitive. Instead, the information about your carbon footprint is displayed against that of your colleague (who came to the office by car) in order to try to "nudge" his behavior. Your activity data (in anonymized form) is also sent to a department of your company that tries to understand the transportation patterns of the employees to implement a series of green policies.*

We believe that the aspects related to the definition of requirements, design and testing of privacy-preserving mobile systems should become a major area of investigation for the software engineering community in the next years. Various approaches can be taken, such as embracing a privacy-by-design philosophy [39]. We think that an in-depth investigation of the principles at the basis of these methodologies should be undertaken by our community, given the fact privacy is a core issue in many mobile applications, whose presence permeates through the entire development process. For example, there is a need to formalize privacy requirements by considering the trade-offs between user privacy and the need to collect and mine the sensed data. Moreover, explicit tracking of privacy information exposure is increasingly considered a fundamental aspect that should be guaranteed in the design of mobile systems [78]. The problem is even more complex when concerns related to differential privacy (i.e., the possibility of using auxiliary sources for compromising the privacy of people) [79] are taken into consideration. For example, differential privacy issues emerge in the design of back-end systems that merge data collected from the

mobile devices and the cellular and WiFi infrastructure with other databases. A typical example is the cross-correlation between location information and geographic databases, which, for instance, can be used to infer a person's political or religious orientation.

Another key concern is related to the design of systems that are privacy-preserving (and secure) but at the same time usable [80]. Dealing with this additional trade-off will be increasingly important as devices are more and more seamlessly integrated in smart spaces and pervasive infrastructures [81–84].

The Disappearing Computer—this time for real. Mark Weiser starts his article "The Computer for the 21^{st} Century" [85] with the famous sentences:

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*

The next step for mobile computing will probably be the disappearance of mobile devices as we know them. Devices will likely still identify us and allow us to connect to the Internet infrastructure but their form will be entirely different. There will be an increasing integration and interaction with the pervasive infrastructure around us, accompanied by increasing use of shared displays, smart environments, and other ambient computing solutions.

> *You are preparing to go out to work in the morning. You cannot find your personal universal identifier anywhere. Then, you spot a small object under the chair. "Found!" you think, as you collect it and put it in your pocket. And you think that maybe you should get the bracelet version instead next time. If you lose it again, it would be very complicated. It would be impossible to access any of your devices, wearables and screens at the bus stop, at work or at the gym.*

So what about the software engineering challenges for the disappearing computer? We believe that they will be related to the increasing distribution, heterogeneity, scale, interoperability, or, in a word, complexity of the existing solutions. The computer will not really disappear and in reality will fragment into a collection of pervasive computing units that will rely on seamless coordination and communication. Lightweight programming abstractions enabling impromptu interaction among this plethora of devices will be fundamental. At the lower level, interoperability is a major ordeal that will have to be overcome either through standardized interfaces or the emergence of a mainstream, dominating solution. Robustness and failure tolerance will be, as in the past, key elements of these future computing systems in our opinion, combined with the aforementioned privacy challenges.

4.3 Impact on Software Engineering

The trends and challenges induced by mobile computing necessarily disrupt traditional views of software engineering. A significant portion of today's software systems have some element of mobility, ubiquity, or sensing; many applications combine all three. To truly address the inherent challenges that these new paradigms bring, software engineering must, in many ways, reinvent itself to follow (and lead) lest principled software engineering be relegated to irrelevance in the future computing ecosystem.

Requirements and Specification. While our existing tools and techniques for capturing and expressing artefacts in the early stages of the software engineering lifecycle should still largely apply, they must be augmented with new considerations. For instance, considering the domains described above, it is clear that purely deterministic specifications no longer apply. Humans are such an integral piece of these systems that the requirements and specifications must incorporate a significant degree of "fuzziness" while retaining strong semantic meaning. For similar reasons, in many instances, non-functional requirements may become predominant.

Traditionally "low-level" concerns will also creep into the requirements and specifications process. Already there in today's processes is security. The rise of inexpensive and pervasive sensing requires the ability to enable applications to provide flexible specifications of their dependence on sensor data and the manner in which that sensor data (ideally, when available) is connected to both functional and non-functional system requirements. As we described in Section 2.3, device and other physical resource constraints (most specifically, energy) have become primary concerns. We must be able to easily specify the desired (and achievable) relationships between application behavior and energy concerns or device lifetimes.

Design. While we posited in 2000 that disconnected operation would be a key design element, the availability of ubiquitous connectivity has changed this perspective to one of how best to leverage available connected resources given resource constraints. With respect to software engineering, this demands patterns and architectures that capture common approaches and best practices to address these concerns within applications. Similarly, differently capable devices and target environments force software engineers to consider, at design time, uniform application capabilities across platforms and unified user experiences across devices. Coordination is still a relevant topic for our community, but the focus has shifted over the last decade to more practical considerations driven by application-level protocols. The emerging trends described above emphasize a lack of general-purpose inter- and intra-application coordination; future software engineering design tools must support flexible, expressive, and context-aware coordination. Another fundamental question for the software engineering community is a deep analysis of the processes currently adopted by major software companies, such as Google, Apple, and the like, and smaller ones, including startups, which are not completely consistent with the traditional practices still taught in our University courses. We believe that an in-depth reflection on these perspectives should be undertaken by researchers working in this area.

Implementation. One piece of our original vision still holds strongly: middleware support will be critical to realizing mobile applications. Showing some similarity, the concept of a "platform ecosystem" is emerging, and, in the end, in a sense, the iOS and Android platforms, and some associated toolkits provide limited middleware-like capabilities, e.g., specifically w.r.t. supporting access to on-board sensors. Following this lead, and supporting the trends described above, middleware for mobile computing must emphasize support not only for communication and coordination but also for sensing. This must go beyond the current toolkit capabilities of accessing the device's sensors to connecting to location-dependent resources available in the immediate environment. Future systems will also provide high-level information extracted from the context, such as the current activity of a person or his/her friends. Middleware is also important in its ability to aid developers in creating software for multiple platforms without having to maintain multiple parallel native versions. This dovetails with the above motivation to unify the requirements and specifications of applications and users' experiences across platforms and devices.

Verification and Validation. Testing mobile computing systems and applications in one sense has not changed at all in the past decade and in another sense is radically different. The pervasiveness of mobile applications, especially in mission critical deploy-

ments, still demands rigorous approaches to verification and validation, and, at the unit level, these efforts may be well supported by existing tools and techniques. However, at the integrated level, we have entered an entirely new realm of software verification. Mobile applications and systems of tomorrow will almost certainly encounter situations "in the wild" that were simply unforeseeable, whether because the human is unpredictable, unexpected sensed values are input, or the application component encounters and coordinates with new components that were not even conceived of when the system was constructed. This is even more true as we consider wearable or implantable devices; no two human bodies or environments are identical, so accounting for every possible deployment without actually deploying the software is impossible. We must develop testing approaches that provide the ability to test (and ideally fix) mobile software *in situ* based on an individual execution. These test results must also be shared and applied to other (running) instances of the same (or similar) software.

As the requirements of these systems become "fuzzy" as described above, checking correctness in a traditional sense may not even be possible, or desirable; we touched on this previously in discussing that, in many emerging mobile computing domains, it may be impossible to even clearly identify "correct" behavior objectively. This is not to say that these systems are untestable, but our tools and techniques need new semantics that still enable reasoning about the desired function of the mobile system.

Maintenance and Deployment. Deployment of mobile applications in the future is likely to be driven by app stores, which enable software developers to make their software rapidly and widely available. This is a boon in lowering the barrier to enter the software market, but it exacerbates at least the previous discussion in verifying software systems since it is difficult to guarantee the quality of the software in this distribution model. While the software may be "safe" (and validated as such by the app store owner), it may not perform as expected or have available the necessary resources (e.g., in terms of physical device capabilities or in terms of data or sensor information).

Sensing capabilities on a user's device are one thing, but another challenge is managing a vision of "smart spaces" in which a plethora of sensing devices is dispersed in our environments. We need generalized support to deploying and maintaining these capabilities; these issues connect to the middleware concerns above that will enable the myriad applications and mobile devices to access these sensors and their data.

While we generally think in terms of individual apps, we could also envision macro-applications that can effectively self-assemble out of components; when a different set of components comes together, different functionality or different quality is enabled. This demands the development of pluggable frameworks that can support these dynamic and unpredictable component connections.

Even as we discuss these issues along the traditional lines of the software engineering lifecycle, it becomes clear that the lines are ever more blurred when one considers the domain of mobile computing. Siloed views of "areas" within software engineering may need to be reconsidered or at least punctured to allow for these concerns to seep from maintenance and deployment to middleware or from design to verification in untraditional ways.

5. IS THE COMMUNITY INTERESTED?

We close our time travel in the past and the future of software engineering for mobility by asking ourselves an important question: is mobility as a topic still able to capture the interest of the software engineering community?

Table 2: Number of papers whose title contains the string `"mobil"` (accounting for both `"mobile"` and `"mobility"`) published in flagship software engineering venues (TSE, TOSEM, ICSE, ASE, ESEC/FSE).

	all	`"mobil"`	%
before 1995	2664	1	0.04%
1995-1997	537	4	0.74%
1998-2000	766	19	2.48%
2001-2003	853	21	2.46%
2004-2006	984	9	0.91%
2007-2009	794	8	1.01%
2010-2012	1173	5	0.43%

As we previously mentioned, mobility was definitely a popular topic when we wrote our original paper. Nevertheless, when writing this paper we all shared the same impression that the topic had somehow faded away.

Without the pretence to be exhaustive, here we offer a few quantitative facts based on a simple analysis of the papers[10] published in five flagship software engineering venues from 1975 to 2012: *IEEE Trans. on Software Engineering (TSE)*, *ACM Trans. on Software Engineering and Methodology (TOSEM)*, *IEEE/ACM Int. Conf. on Software Engineering (ICSE)*, *IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, and *SIGSOFT Int. Symp. on Foundation on Software Engineering (FSE) and the European Software Engineering Conf. (ESEC)*. We determine the presence of mobility as a topic by counting the number of papers whose title contains the string `"mobil"`, thus accounting for both `"mobile"` and `"mobility"`. We focused only on the paper title because the dataset does not contain abstracts for all papers.

Table 2 shows the results. This analysis, albeit simple and with its own limitations, appears to confirm our aforementioned impression. Mobility was essentially unmentioned before 1995. However, it then rapidly surged in popularity, reaching a peak in the period 1998-2003: our paper was published in the middle of this "golden period". After 2003, mobility became significantly less present in these venues, both in absolute (i.e., number of papers) and relative terms.

There are a number of potential explanations for this observation. In part, this can be ascribed to the fact that, as mentioned in Section 2.3, the research community at large lost interest in logical mobility. Digging into the dataset confirms that, after 2003, "mobile" papers focusing on logical mobility are quite rare. Nevertheless, this alone cannot explain the dwindling numbers. After all, the iPhone was released in 2007, and the rise of ubiquitous connectivity began even before that.

One could conclude that, for whatever reason, the software engineering community has not been really engaged in what may well be one of the turning points of information technology. To add to the negativity of this remark, we also note that mobility is not the only casualty of this attitude. Other related topics that flourished over the last decade (e.g., pervasive computing, wireless sensor networks) had an even more sparse presence at flagship software engineering venues[11]. We do not know why the software engineering community did not take a more active part in these research topics. We speculate that this may be due to the fact that many of the inher-

[10]Many thanks to Carlo Ghezzi and Mario Sangiorgio for making the raw publication data available to us.

[11]A query for `pervasive OR ubiquitous OR ambient intelligence` returns 12 entries (all in 2005–2012), while a query for `sensor` returns 2 entries (in 2011–2012).

Table 3: Distribution in the last three years and for different keywords, for papers published in the same venues as Table 2.

	2010	2011	2012
`mobil*`	1	0	4
`Android`	0	0	4
`phone`	0	0	1
Total	1	0	9

ent challenges in these fields entail low-level system issues, which our community has a tendency to sweep under the rug of some layer of abstraction—as we ourselves did, as noted in Section 2.3. A possible confirmation for this consideration can be found by analyzing the presence of papers concerned with middleware, a topic that is arguably at the crossroads of software engineering and systems research and that constitutes one of the main challenges in mobility as well, as noted earlier. A search for `"middleware"` returns 18 entries in the 6-year period 2001–2006, and only 4 entries in 2007–2012. This significant reduction could be attributed to the software engineering community losing interest in these topics but also to the desire of software engineering researchers to publish into the flagship venues specific to these topics. We do not have enough elements to concretely support either hypothesis. The bottomline, however, is that the flagship software engineering venues are increasingly losing contact not only with mobility, but also with other system-oriented topics that are thriving in computing at large.

On the other hand, it is also possible that the analysis above tells only one part of the story and that we may actually be on the verge of a new golden period for mobility. Table 3 shows, for the same venues, a "zoomed in" view of the last 3-year period, considering additional keywords related to smartphone devices. Two observations can be made. First, after two years (2010 and 2011) in which mobility essentially disappeared from the radar of software engineering, in 2012 alone there were a total of 9 papers related to mobility topics, which is more than in each of the two preceding 3-year periods. Second, the two additional keywords we considered, `"phone"` and `"Android"`, both appeared for the first time[12] in 2012. Of course, it would be a stretch to extrapolate a long-term trend based on such a small sample. Despite the obvious limited statistical significance, this data points to the fact that some emphasis on mobility is returning, although mostly constrained to the devices themselves, rather than their distributed interaction.

Only time will tell if these are signs of a renewed interest of the software engineering community for the mobility topic, or instead nothing more than outliers. Given the challenges and scenarios we outlined, our hopes are on the former: the stakes are simply too high for the software engineering community to ignore the game.

6. CONCLUSIONS

The rate of change in mobile computing and communications is unlikely to slow down. Therefore, nowadays, making predictions about the future of this area is even more difficult. Nevertheless, the lessons learned by examining retrospectively why forecasts missed the mark might provide valuable insights in terms of factors that must be given increased significance in the future agenda of the research community. In an interconnected world and at a time in history when we recognize the importance of multidisciplinary approaches in solving complex problems, we should not be surprised by the tight integration of technology, business, social, and psychological concerns that seems to characterize mobility as a field of study. The real question is how will the software engineering community become a major player in this exciting field.

[12]The keyword `"phone"` appeared in 1993 and 1996, but in relation to telephone switches.

7. REFERENCES

[1] G.-C. Roman, G.P. Picco, and A.L. Murphy. Software Engineering for Mobility: A Roadmap. In Finkelstein [2].

[2] A. Finkelstein, editor. *The Future of Software Engineering*. ACM Press, 2000.

[3] A. Carzaniga, G.P. Picco, and G. Vigna. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *Proc. of ICSE*, 2007.

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

[5] S. Nath. ACE: Exploiting correlation for energy-efficient and continuous context sensing. In *Proc. of MobiSys*, 2012.

[6] N. Roy, A. Misra, C. Julien, S.K. Das, and J. Biswas. An energy-efficient quality adaptive framework for multi-modal sensor context recognition. In *Proc. of PerCom*, 2011.

[7] L. Mottola and G.P. Picco. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. *ACM Computing Surveys*, 43(11), 2011.

[8] P. Costa, L. Mottola, A.L. Murphy, and G.P. Picco. Programming Wireless Sensor Networks with the TeenyLIME Middleware. In *Proc. of Middleware*, 2007.

[9] A.L. Murphy, G.P. Picco, and G.-C. Roman. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. *ACM Trans. on Software Engineering and Methodology (TOSEM)*, 15(3), 2006.

[10] M. Ceriotti, L. Mottola, G.P. Picco, A.L. Murphy, S. Guna, M. Corrà, M. Pozzi, D. Zonta, and P. Zanon. Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment. In *Proc. of IPSN*, 2009.

[11] M. Ceriotti, M. Corrà, L. D'Orazio, R. Doriguzzi, D. Facchin, S. Guna, G.P. Jesi, R. Lo Cigno, L. Mottola, A.L. Murphy, M. Pescalli, G.P. Picco, D. Pregnolato, and C. Torghele. Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels. In *Proc. of IPSN*, 2011.

[12] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1), 2000.

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

[14] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Mobile networking for smart dust. In *Proc. of MobiCom*, 1999.

[15] F. X. Lin, A. Rahmati, and L. Zhong. Dandelion: a framework for transparently programming phone-centered wireless body sensor applications for health. In *Proc. of Wireless Health*, 2010.

[16] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, and G. Borriello. Open data kit sensors: a sensor integration framework for android at the application-level. In *Proc. of MobiSys*, 2012.

[17] G.P. Picco. Software Engineering and Wireless Sensor Networks: Happy Marriage or Consensual Divorce? In *Proc. of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSER'10), co-located with the 18th ACM Int. Symp. on the Foundations of Software Engineering (FSE)*, 2010.

[18] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil. An

empirical study of geographic user activity patterns in Foursquare. In *Proc. of ICWSM*, 2011.

[19] N.D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A.T. Campbell. A Survey of Mobile Phone Sensing. *IEEE Communications Magazine*, 48, 2010.

[20] L. Mottola and G.P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Surveys (CSUR)*, 43(3), 2011.

[21] C. Lee, D. Nordstedt, and S. Helal. Enabling smart spaces with osgi. *IEEE Pervasive Computing*, 2(3), 2003.

[22] E. Miluzzo, N.D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S.B. Eisenman, X. Zheng, and A.T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proc. of SenSys*. ACM, 2008.

[23] J. Kukkonen, E. Lagerspetz, P. Nurmi, and M. Andersson. Betelgeuse: A platform for gathering and processing situational data. *IEEE Pervasive Computing*, 8(2), 2009.

[24] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proc. of MobiSys*, 2009.

[25] Android Developers. Recognizing the user's current activity. `http://developer.android.com/training/location/activity-recognition.html`.

[26] N. Lathia, V. Pejovic, K. Rachuri, C. Mascolo, M. Musolesi, and P.J. Rentfrow. Smartphones for large-scale behaviour change intervention. *IEEE Pervasive Computing*, 12, 2013.

[27] B. Priyantha, D. Lymberopoulos, and J. Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, 10(2), 2011.

[28] K.K. Rachuri, C. Efstratiou, I. Leontiadis, and C. Mascolo. METIS: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *Proc. of PerCom*, 2013.

[29] A. McCallum, K. Schultz, and S. Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Proc. of NIPS*, 2009.

[30] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, R.A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12(4), 2008.

[31] G. Chittaranjan, J. Blom, and D. Gatica-Perez. Mining large-scale smartphone data for personality studies. *Personal and Ubiquitous Computing*, 17(3), 2013.

[32] M. Musolesi. Big Mobile Data Mining: Good or Evil? *IEEE Internet Computing*, 2014.

[33] T. Hunter, R. Herring, P. Abbeel, and A. Bayen. Path and travel time inference from GPS probe vehicle data. *NIPS Wkshp. on Analyzing Networks and Learning with Graphs*, 2009.

[34] D. Lazer, A. Pentland, L. Adamic, S. Aral, A.-L. Barabasi, D. Brewer, N. Christakis, N. Contractor, J. Fowler, M. Gutmann, T. Jebara, G. King, M. Macy, D. Roy, and M. Van Alstyne. Computational social science. *Science*, 323(5915), 2009.

[35] A. Pentland. *Honest signals*. MIT press, 2010.

[36] M. Berlingerio, F. Calabrese, G. Di Lorenzo, R. Nair, F. Pinelli, and M. L. Sbodio. AllAboard: A system for exploring urban mobility and optimizing public transport using cellphone data. In *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2013.

[37] A. Lima, M. De Domenico, V. Pejovic, and M. Musolesi. Exploiting Cellular Data for Disease Containment and Information Campaigns Strategies in Country-wide Epidemics. In *Proc. of NetMob*, 2013.

[38] T. Berners-Lee, J. Hendler, O. Lassila, et al. The Semantic Web. *Scientific American*, 284(5), 2001.

[39] M. Langheinrich. Privacy by design principles of privacy-aware ubiquitous systems. In *Proc. of UbiComp*. Springer, 2001.

[40] J. Wang and D. Katabi. Dude, where's my card?: RFID positioning that works with multipath and non-line of sight. In *Proc. of SIGCOMM*, 2013.

[41] Y. Rogers. Moving on from Weiser's Vision of Calm Computing: Engaging Ubicomp Experiences. In *Proc. of UbiComp*. Springer, 2006.

[42] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*. Springer, 2011.

[43] M. Hazas, J. Scott, and J. Krumm. Location-aware computing comes of age. *Computer*, 37(2), 2004.

[44] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman. I'm the mayor of my house: examining why people use foursquare-a social-driven location sharing application. In *Proc. of CHI*. ACM, 2011.

[45] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys*, 2011.

[46] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In *Proc. of MobiSys*, 2010.

[47] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *IEEE Computer*, 43(4), 2010.

[48] M. Pitkänen, T. Kärkkäinen, Jörg Ott, M. Conti, A. Passarella, S. Giordano, D. Puccinelli, F. Legendre, S. Trifunovic, K. Hummel, M. May, N. Hegde, and R. Spyropoulos. SCAMPI: Service platform for social aware mobile and pervasive computing. *ACM SIGCOMM Computer Communication Review*, 42(4), 2012.

[49] L. Buechley. A Construction Kit for Electronic Textiles. In *Proc. of ISWC*. IEEE, 2006.

[50] M. Hamedi, R. Forchheimer, and O. Inganäs. Towards woven logic from organic electronic fibres. *Nature materials*, 6(5):357–362, 2007.

[51] M. Firtman. *Programming the Mobile Web*. O'Reilly, 2013.

[52] A. Popescu. Geolocation API Specification. *W3C Candidate Recommendation*, 2013.

[53] G. Blair and P. Grace. Emergent middleware: Tackling the interoperability problem. *IEEE Internet Computing*, 16(1), 2012.

[54] A. Holzinger, P. Treitler, and W. Slany. Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones. In *Multidisciplinary Research and Practice for Information Systems*. Springer, 2012.

[55] A. Holzer and J. Ondrus. Trends in mobile application development. In *Mobile Wireless Middleware, Operating Systems, and Applications-Workshops*. Springer, 2009.

[56] V. Pejovic and M. Musolesi. Anticipatory mobile computing: A survey of the state of the art and research challenges. arXiv:1306.2356 [cs.HC], 2013.

[57] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, and T. Choudhury. Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns. In *Proc. of ISWC*, 2013.

[58] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu. Fast app launching for mobile devices using predictive user context. In *Proc. of MobiSys*, 2012.

[59] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. PocketWeb: Instant web browsing for mobile devices. In *Proc. of ASPLOS*, 2012.

[60] A. Campbell and T. Choudhury. From smart to cognitive phones. *IEEE Pervasive Computing*, 11(3), 2012.

[61] S. Han and M. Philipose. The case for onloading continuous high-datarate perception to the phone. In *Proc. of HotOS*, 2013.

[62] R. de Lemos, H. Giese, H.A. Müller, and M. Shaw. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, 2013.

[63] K. Kelly. Better than human: Why robots will—and must—take our jobs. *Wired*, 2012.

[64] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter. BigDog, the rough-terrain quadruped robot. In *Proc. of IFAC*, 2008.

[65] W. Bluethmann, R. Ambrose, M. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder. Robonaut: A robot designed to work with humans in space. *Autonomous Robots*, 14(2–3), 2003.

[66] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler. Aerial robotic construction: Towards a new field of architectural research. *Int. Journal of Architectural Computing*, 10(3), 2012.

[67] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31, 2008.

[68] T.-C. Au, C.-L. Fok, S. Vishwanath, C. Julien, and P. Stone. Evasion planning for autonomous vehicles at intersections. In *Proc. of IROS*, 2012.

[69] J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(3), 2003.

[70] E. Guizzo. How Google's Self-driving Car Works. *IEEE Spectrum Online*, 18, 2011.

[71] R. Oung and R. D'Andrea. The distributed flight array: Design, implementation, and analysis of a modular vertical take-off and landing vehicle. *Int. Journal of Robotics Research*, 2013.

[72] A.B. Schwartz, X.T. Cui, D.J. Weber, and D.W. Moran. Brain-controlled interfaces: Movement restoration with neural prosthetics. *Neuron*, 52(1), 2006.

[73] J.R. Wolpaw, N. Birbaumer, D.J. McFarland, and G. Pfurtscheller T.M. Vaughan. Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 113, 2002.

[74] B.S. Moon, H.C. Lee, Y.H. Lee, J.C. Park, I.S. Oh, and J.W. Lee. Fuzzy systems to process the ECG and EEG signals for quantification of the mental workload. *Information Sciences*, 142(1–4), 2002.

[75] A. Campbell, T. Choudhury, S. Hu, H. Lu, M.K. Mukerjee, M. Rabbi, and R.D.S. Raizada. NeuroPhone: Brain-mobile phone interface using a wireless EEG headset. In *Proc. of MobiHeld*, 2010.

[76] M.K. Petersen, C. Stahlhut, A. Stopczynski, J.E. Larsen, and L.K. Hansen. Smartphones get emotional: Mind reading images and reconstructing the neural sources. In *Proc. of ACLI*, 2011.

[77] A.R. Beresford and F. Stajano. Location privacy in pervasive computing. *Pervasive Computing, IEEE*, 2(1), 2003.

[78] W. Enck, P. Gilbert, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of OSDI*, 2010.

[79] C. Dwork. Differential privacy. In *Automata, languages and programming*. Springer, 2006.

[80] E. Kaasinen. User needs for location-aware mobile services. *Personal and Ubiquitous computing*, 7(1), 2003.

[81] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4), 2001.

[82] X. Jiang and J.A. Landay. Modeling privacy control in context-aware systems. *IEEE Pervasive Computing*, 1(3), 2002.

[83] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M.D. Mickunas. Towards security and privacy for pervasive computing. In *Software Security Theories and Systems*. Springer, 2003.

[84] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M.D. Mickunas. Cerberus: a context-aware security scheme for smart spaces. In *Proc. of PerCom*, 2003.

[85] M. Weiser. The computer for the 21^{st} century. *Scientific American*, 265(3), 1991.