MoBe: Context-Aware Mobile Applications on Mobile Devices for Mobile Users

Paolo Coppola[1], Vincenzo Della Mea[1], Luca Di Gaspero[2], Stefano Mizzaro[1], Ivan Scagnetto[1], Andrea Selva[1], Luca Vassena[1], Paolo Zandegiacomo Riziò[1]

[1]Department of Mathematics and Computer Science
[2]Department of Electrical, Management, and Mechanical Engineering
via delle Scienze 206 – University of Udine – I-33100 Udine, Italy
{coppola,dellamea,mizzaro,scagnetto,selva}@dimi.uniud.it, l.digaspero@uniud.it, {lucavax,zandepaolo}@inwind.it
http://www.mobe.it

ABSTRACT

Due to the appearance and widespread diffusion of new mobile devices (PDAs, smartphones etc.), the traditional notion of computing is quickly fading away, giving birth to new paradigms, where concurrent entities, moving from one location to another, exchange data and cooperate towards a common goal. Hence, the scientific community is searching for models, technologies, and architectures in order to suitably describe and guide the implementation of this new computing scenario. It is clear that the notion of context plays a fundamental role, since it influences the computational capabilities of the devices that are in it.

The present work directly addresses this problem proposing MoBe, a novel architecture for sending, in push mode, mobile applications (that we call MoBeLets) to the mobile devices on the basis of the current context the user is in. The latter is determined by both an ad-hoc MoBe infrastructure and data from sensors on the mobile device (or in its surroundings).

Keywords

Mobile devices, context-aware, software architecture.

INTRODUCTION

We envisage a world in which the mobile devices that everybody currently uses (cellular phones, smart phones, PDAs, and so on) constantly and frequently change their functioning mode, automatically adapting their features to the surrounding environment and to the current context of use. For instance, when the user enters a shopping mall, the mobile phone can provide him/her with applications suitable for shopping, i.e., article locator, savings advertiser etc; when entering in a train station, the same device becomes a train timetable able to give information on the right train lane, delays, etc.

How to achieve this goal is not clear. It is well known that current mobile devices can be used as computers, since they have computational and communication capabilities similar to computers of a decade ago. One approach might be to have an operating system continuously monitoring sensors on the mobile device, thus inferring situational information and triggering the right (preloaded) application for the current context. Another approach is to have a Web browser showing to the user context-aware data selected by means of information filtering techniques.

In our opinion both these alternatives suffer from a lack of flexibility and a waste of computational power. We propose a different approach, where servers continuously push software applications to mobile devices, depending on the current context of use. Inspired by the well-known Nicholas Negroponte's "Being Digital" expression, we name our approach *MoBe* (Mobile Being), and the context-aware applications pushed and executed on the mobile device *MoBeLets*.

This is an interdisciplinary work: mobile agent community, context aware computing, software engineering and middleware, interaction with mobile devices applications, information retrieval and filtering, and privacy and security management are all disciplines that are deeply involved in our project.

In this paper we describe our approach and some details of its ongoing implementation, emphasizing how the MoBe architecture efficiently supports a notion of context history.

The paper is structured as follows. In Section "Related Work" we recall the state-of-the-art in the literature for the research fields related to our work. In Section "The Overall Architecture of MoBe" we describe the structure of our model. In particular we give some details about the key submodules dealing with the data sensing and context inference activities, with the personalization issues, and with the problem of filtering, downloading and executing the MoBeLets. Section "Discussion and Open Problems" is devoted to the analysis of several practical issues we found during our first prototype of the MoBe architecture. Moreover, we also explain how the MoBe architecture naturally supports context histories.

RELATED WORK

This is an interdisciplinary work and there are several related fields.

Context-aware computing is more than 10 years old, as it was first discussed in [8]. However, the field seems still in its infancy, as even the core definition of context is still unsatisfying. Some definitions are, like dictionary definitions, rather circular, since they simply define context in term of concepts like "situation", "environment", etc. Some researchers tried to define this concept by means of examples [2,9]; other researchers searched for a more formal definition [2,3,10]; others identified context with location [8] or with location, time, season, etc. [1,7].

An interesting framework for the development of location aware applications is described in [11], where a symbolic location model is used to represent the user's situational context and a map modeling tool links the symbolic information to the corresponding geographical coordinates. The resulting hierarchical structure is encoded in XML and can be accessed through the WWW, without the need of an explicit server infrastructure.

Another related research field concerns mobile agents [12]. Our approach tries to avoid all the resource load that these architectures usually carry with, and to provide a simpler implementation.

Information retrieval, context aware retrieval, just-in-time information retrieval, and information filtering deal with the information overload problem from different facets [6] [5]. Google is starting to provide contextual (actual, localized) services as well (http://www.google.com/lochp). Peer-to-peer networks and wireless networks and technologies are of course involved as well.

THE OVERALL ARCHITECTURE OF MOBE

Figure 1 shows the overall MoBe architecture. The mobile device runs a software module called *MoBeSoul* that is responsible of managing the whole lifecycle of a context-aware application. Let's follow the events that lead to pushing, downloading, and executing a MoBeLet on the mobile device.

Context submodule

The process starts with context data received through:

- *Physical sensors*. Almost all mobile devices are equipped with some form of wireless network technologies (GSM, GPRS, UMTS, Bluetooth, Wi-Fi, Radio Frequency, IrDA, etc.), and can therefore sense if there is a network connection around them (and the strength of the corresponding electromagnetic field). Moreover, the device might be equipped with sensors capable of sensing data about the physical world surrounding the mobile device (e.g., noise, light level, temperature, etc.); also, the device might be able to receive data about its environment (e.g., temperature, etc.) from some surrounding sensors.
- *"Virtual" sensors*. MoBeSoul might receive data from other processes running on user's mobile device, like an agenda, a timer, an alarm clock, and so on.
- *MoBeContext sensors*. MoBeSoul is capable of receiving context information provided by an ad-hoc *MoBe Context Server* (*MCS*). The MCS pushes information about the current context to the users devices, with the aim of providing a more precise and complete context description. MCS might be implemented by a Wi-Fi antenna, an RFID tag sensed by the mobile device, or any other technology. The MCS also broadcasts a Context ID (that, in the case of a Wi-Fi antenna might be the network SSID and its MAC address).
- *Explicit user actions*. The user can explicitly communicate, via the user interface, data about the current context. For instance, he/she might choose a connection/network provider, set the alarm clock, select the silent mode, and so on.
- *Context history representations*. Sequences of contexts traversed by the user in the past can be summarized in some abstract form and used as context data as well, together with the other kinds of context.

All these sensors data are processed by the MoBeSoul *Context* submodule. It is responsible of producing, storing, maintaining, and updating a description of the current context the user is in. The Context submodule starts its inferential activity from *concrete contexts* (i.e., contexts directly corresponding to sensors data). By some inferential mechanism (we are currently devising a mechanism that employs Bayesian Belief Networks) it derives *abstract contexts* (i.e., context which can be processed more conveniently; some of the abstract contexts might be just concrete contexts). The data and the inference are uncertain, and both the concrete contexts and the inferred abstract contexts have a probability measure representing how likely it is that the user is indeed in those contexts. The inferential engine exploits a database containing the history of past contexts and it is tightly integrated with the Personalization submodule (explained later), managing user's preferences, user's current cognitive load, and degree of attention, etc. Concrete and abstract contexts are represented by means of *context descriptors*; the inferred abstract contexts descriptors are stored in a *Current Context Working Memory*, and they survive until the event of exit from that context is inferred.

Examples of current contexts are: the temperature is 20 degrees (with probability 0.9); the time is 12:30:00PM (p = 0.99); the MoBeContext ID is 1234; and so on.

Examples of abstract contexts are: the user is in a shopping mall (p=0.75); the user is in the AirWood bookshop inside the shopping mall in Udine West; the user is in his/her car (p=0.56); the user is driving a car (p=0.8).

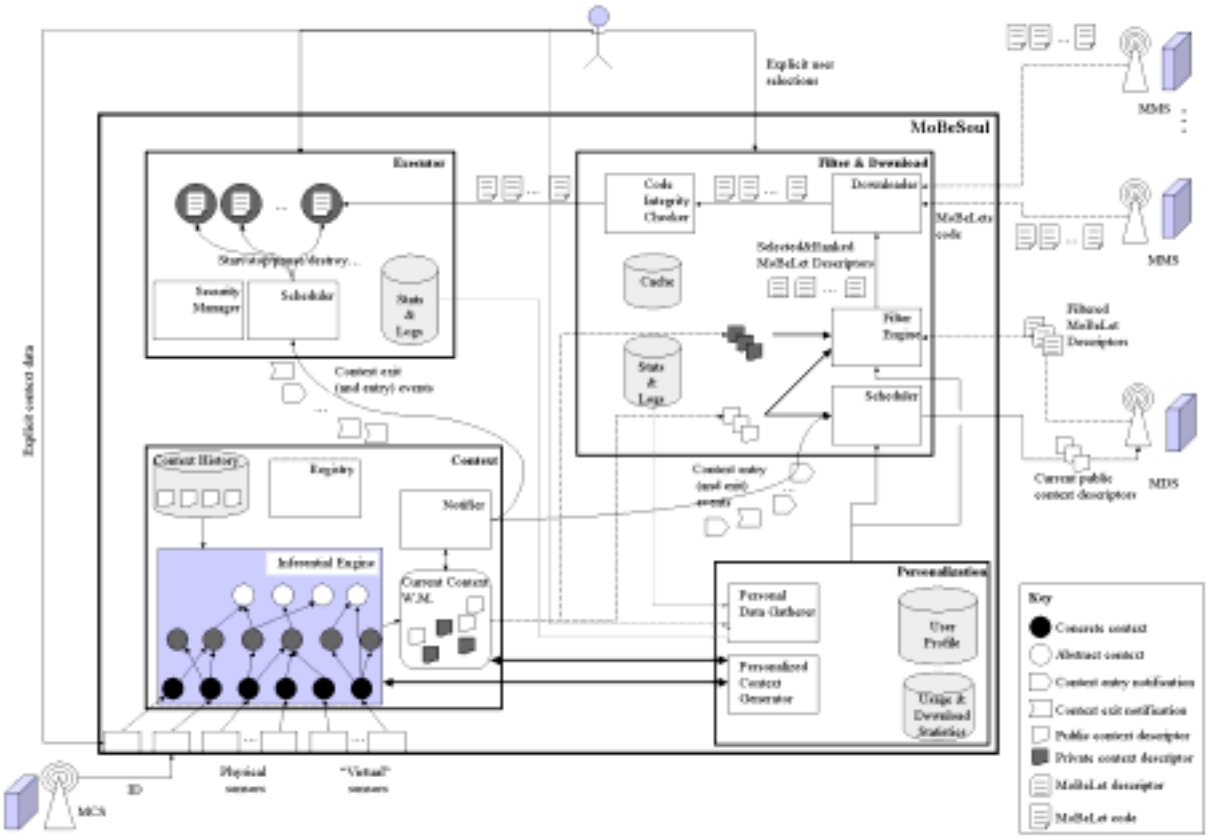

Figure 1. MoBe overall architecture.

Private context histories can be stored and processed only Contexts are divided into a *public* and a *private* part: the former can be distributed to servers and other entities and contains, e.g., user's approximate location, cognitive load, and so on; the latter is kept private inside the MoBeSoul and contains, e.g., user's exact position, credit card information or some other personal data, and so on.

Of course, personal user preferences can change the public/privacy status of each item in a context descriptor. on the user device; public context parts may be sent to external entities able to collect individual context histories and aggregate them for some purpose.

Context submodule does not send autonomously context descriptors to other parts of the system; rather, it keeps a *registry* of interested observers/listeners, which are notified by the *Notifier* when the context entry/exit events happen. After the notification, the observers can decide, using their own criteria, to request the needed context descriptors to the context module.

Personalization submodule

The *Personalization* submodule has two aims:

- The *Personal Data Gatherer* collects data about user's preferences and habits, storing them into two internal databases: the *User Profile* database contains several different kinds of data, like user's demographic information (age, gender, etc.), preferences about real world activities (e.g., restaurants, friends, etc.), habits (working hours, typical trips, etc.), and so on; the *Usage & Download Statistics* database contains data about which MoBeLets have been downloaded and executed in the past, for how much time, which resources have been used, and so on. User's data are collected both automatically (monitoring user's behavior) and manually, by explicit user intervention.
- The *Personalized Context Generator* interacts with the Context submodule, affecting the inference process with the aim of making it more tailored to individual needs. A useful metaphor to understand the interaction between Context and Personalization submodules is to see the Bayesian inferential network inside Context as a graph painted on a sheet of paper, and to imagine the Personalization activity as a transparent sheet of paper on top of it: the Personalization layer is specific to the single user, it has a higher priority and is capable to change the underlying (and more general) context network. The personalization layer can remove (hide) nodes and arcs, change arcs weights (probabilities) either in an absolute way (by specifying a new value) or in a relative way (by increasing or decreasing the underlying weight of a given amount). This also allows to modify in a seamless way the Context network, in order to include unforeseen contexts and inferences even after the system is deployed.

Summarizing, contextual information is derived by the mobile device from physical, virtual, ad-hoc sensors, and user data; the Context and Personalization submodules infer an abstract description of the current context taking into account, besides concrete context data, inference rules, user's preferences (history, user model, …), user's current activities, cognitive load, degree of attention, other devices proximity, etc. A clear separation between context and personalization seems difficult to realize, but has important benefits: independent modification of the Context network, independent usage of well established techniques from both the personalization and context-awareness fields, development of a non-personalized prototype of the MoBeSoul, and so on. However, the relationship between context-awareness and personalization should be carefully studied, since, e.g., context histories might be viewed as a source of personalization data.

Filter and Download submodule

The *Filter and Download* submodule is in charge of selecting which MoBeLets to download and to retrieve their code. It is triggered by notifications of context entry and exit events, received from the Context submodule. The *Scheduler* receives these notifications and, on the basis of its internal criteria, also depending on user's preferences, decides when to request the current public context descriptors to the Context submodule and to forward them to a *MoBe Descriptors Server* (*MDS*). The MDS is in charge of selecting, on the basis of the received context descriptors, those MoBeLets that are more relevant to user's current context.

Since not all the MoBeLets selected on the basis of the public context descriptors will be downloaded (nor executed), the MDS does not handle MoBeLet code, but just *MoBeLets descriptors*. Each descriptor is a simple XML file containing several data about the corresponding MoBeLet: an unique identifier, a textual description, a manifest declaring which resources the MoBeLet will need and use while executing, a download server from which the actual MoBeLet can be downloaded, and so on.

The received MoBeLet descriptors are filtered once again by the *Filter Engine*, using the private context descriptors. As a result of this step, the probability that the user will desire to run each MoBeLet is determined. Then the *Downloader* downloads, on the basis of its own internal criteria, the MoBeLets code, from the *MoBe MoBeLet Server* (*MMS*) specified in the corresponding descriptors. The stream of MoBeLets is then passed to the Executor.

This design allows:

- To encapsulate inside the Scheduler adequate strategies to send to the MDS the public context descriptors, for a more efficient resource usage: the Scheduler might send the context descriptors at each context change, it might collect a certain number of context descriptors (perhaps removing those corresponding to context exit events received meanwhile), it might send context descriptors at fixed time points, and so on.
- To separate public and private context data: only the public data are sent to MDS, but both public and private are used to filter the MoBeLet descriptors received.
- To easily cache both MoBeLet descriptors and code, in order to minimize bandwidth usage.
- To have the user controlling the whole process and to participate in MoBeLets filtering and selection: the user might proactively stop an undesired MoBeLet, or be requested a preference to a resource demanding MoBeLet, and so on. On the other side, the two stage filtering allows a lower cognitive load to the user.

Executor submodule

The last submodule of the pipeline is the *Executor*. Its aim is to run each downloaded MoBeLet inside a sandbox, in order to avoid malicious MoBeLets to use resources against user's will. Each MoBeLet is managed by the *Scheduler*, which is capable of starting, pausing, stopping, and destroying the MoBeLets. The Scheduler is notified of context exit (and entry) events, to stop those MoBeLets that go out of context. Each MoBeLet can register itself with the Registry inside the Context submodule, in order to be directly notified of relevant context change events.

Each MoBeLet that has to use resources outside its sandbox is allowed to do so only through the *Security Manager*, which will deny requests that are incompatible with MoBeLet manifest, prompting the user to confirm more heavy resource usages.

DISCUSSION AND OPEN PROBLEMS

We described an architecture that is still under development; in this section we focus on some open issues.

Scalability issues

MoBe architecture is scalable for what concerns MCS and MS: more servers can be added at will, since each of them does not provide a centralized service. The bottleneck of this architecture is the MDS: in some cases, the MoBeLet descriptors request will be sent to some local server (when the MCS provides a context ID); but in some other cases the MoBeLet descriptors request will be sent to the main MDS server (when the ID can't be provided). In the last case, there is the risk of overloading the main MDS server.

To understand if this is a serious problem, let us try to compare it to nowadays Google statistics. Google receives, and processes almost immediately, about 1000 queries per second. If MoBe will be adopted, we can estimate about 1 billion of MoBe enabled mobile devices, each of which will probably perform, on average, about 1000 context change per day (in daytime, about 50-100 context change per hour; no context change during the night). This would mean a total of 10^{12} context change per day, i.e., $(10^{12}) / (24 \times 60 \times 60) \approx 10^{7}$ ca. context change per second. Not all of them will be sent from MoBeSoul, since the Schedule submodule Filter & Download selects and queues some public context descriptors, but let us be pessimistic and assume that this does not decreases significantly the number of requests to the public server. Let us assume instead that the local

server allow to decrease of another factor of 10, leading to 10^6. This is 1000 higher than today's Google, but it is not so frightening; at worst, we might deploy 1000 MDS around the world, and configure the MoBeSouls so that each of them talks to one of these (e.g., randomly, or statically), thus distributing the load. As a last note on this issue, let us remark that in principle MCS, MDS and MMS can be the same server.

Structured vs. unstructured approach

Turning to more general issues, we see two major trends in current computer science and web technologies. The first one is to provide *structure* in the produced data: in databases, data are stored and retrieved accordingly to well defined schema; XML, HTML, XHTML can instill semantic information in otherwise almost unstructured natural language text; Web services are described on the basis of specific XML formats; Semantic Web is a hot word in the community; and one might go on. Research within the second trend is devoted to empower current algorithms, techniques, and software applications in order to deal with unstructured data: search engines are the second activity of web users (after email); Google GMail fosters an unstructured view of one's own mailboxes; images, sounds, and videos are often searched on the basis of their semantic content, which is hard to encapsulate in a-priori textual descriptions; and so on.

MoBe tries to combine both approaches: a context descriptor is are made of structured data; a MoBeLet descriptors can be mainly made of structured data, provided by the MoBeLet creator, but in principle it is possible to have also unstructured data like, e.g., the comments inserted in the code by the programmer and to exploit state-of-the-art software retrieval and filtering techniques [4].

Application vs. data

Within MoBe, applications are sent around, not just data. Of course, this is a subtle distinction: as every student knows, for a compiler an application is simply data; moreover, looking inside the memory of a computer, one cannot distinguish between bytes representing programs and bytes representing data on which programs run. However, from an abstract/semantic viewpoint, it is perfectly reasonable to distinguish between the two.

Therefore, MoBe approach is different from current mainstream that relies on Web browsers based on HTTP-like protocols (HTTP, WAP, etc.). We believe that this is a shortsighted view: using a well-known metaphor, we might be experiencing the QWERTY of mobile/contextual applications/devices. MoBe is a much more flexible and powerful architecture. Of course, we are aware that it has its own weaknesses: writing software instead of data is more difficult; sending applications might lead to spread malicious MoBeLets (i.e., viruses); privacy issues, handled by distinguishing between public and private context parts, are much more complex, and so on.

Context histories, context, and personalization

MoBe architecture is somehow neutral with respect to context histories, but it takes them into account in a rather natural way. First, the Inferential Mechanism inside the Context submodule infers the abstract context not only on the basis of the current data from the sensors, but also exploiting the context history database. Second, downloaded and executed MoBeLets can be selected not only on the basis of the current context, which in turns depends on the context history, but also exploiting the Statistics & Log databases inside the Filter & Download and Executor submodules.

This is another point in which the aforementioned separation between context and personalization is, although tricky, advantageous, since it can simplify and empower context histories management. Indeed, statistics and logs of MoBeLet usage by a user are rather sensible data; hence, they can be exploited at Personalization (rather than Context) level. On the other side, average statistics on MoBeLets download and usage could be kept on the MoBe Descriptor Server, to provide a more effective filtering by the public context descriptors. Finally, the distinction between context-aware and personalization (and public and private context) is a complex issue deserving further work.

REFERENCES

[1] P.J. Brown, J.D. Bovey and X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications* **4**(5): 58–64. 1997.

[2] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research, 2000

[3] A.K. Dey and G.D. Abowd. *Towards a Better Understanding of context and context-awareness*. TR GIT-GVU-99-22, Georgia Inst. of Tech., College of Computing, 1999. ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf

[4] R. Gonzales, K. van der Meer. Standard metadata applied to software retrieval. *J. of Inf. Science*, **30**(4): 300–309, 2004.

[5] G.J.F. Jones, P.J. Brown. Context-aware retrieval for ubiquitous computing environments, In F. Crestani, M. Dunlop, S. Mizzaro (eds.) *Mobile and ubiquitous information access*, vol. 2954 of LNCS: 227–243, Springer-Verlag, 2004.

[6] B.J. Rhodes, P. Maes. Just-in-time information retrieval agents, *IBM Systems J.*, **39**(3–4): 685 – 704, 2000.

[7] N. Ryan, J. Pascoe and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. *Computer Applications and Quantitative Methods in Archaeology*. V. Gaffney, M. van Leusen and S. Exxon (eds.). Oxford (UK), 1998.

[8] B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *IEEE Networks* **8**: 22–32, 1994.

[9] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, 85–90, Santa Cruz, CA, 1994.

[10] A. Schmidt, K. Asante Aidoo, A. Takaluoma, U. Tuomela, K. van Laerhoven, and W. van de Velde. Advanced interaction in context. In *Proc. of 1st Int'l Symp. on Handheld and Ubiquitous Computing*, 89–101, Karlsruhe (Germany), 1999.

[11] C. Stahl, D. Heckmann. Using Semantic Web Technology for Ubiquitous Hybrid Location Modeling. In *Workshop notes on Ubiquitous GIS*, in conj. with Geo-Informatics 2004.

[12] M. Wooldridge. *An Introduction to Multiagent Systems.* John Wiley & Sons, 2002.

Author Biographies

Paolo Coppola, PhD, is an assistant professor of computer science at the University of Udine, Italy. His main research interests include the implementation of functional languages, type systems, linear logic, optimal reduction, lambda calculus and computational complexity. He is currently investigating aspects of mobile systems and context-aware computing.

Vincenzo Della Mea, PhD, is an assistant professor of computer engineering at the University of Udine. His research focuses on medical informatics and telemedicine, evaluation of medical systems, hypermedia, eLearning technologies, and mobile devices.

Luca Di Gaspero, PhD, is an assistant professor of computer engineering at the University of Udine. His main research interest concern the investigation of search techniques for scheduling problems. Recently he became interested also in information retrieval and in mobile systems and context-aware computing.

Stefano Mizzaro, PhD, is an assistant professor of computer engineering at the University of Udine. His research activities are mainly in the fields of Web information retrieval, artificial intelligence, scholarly publishing, and mobile devices.

Ivan Scagnetto, PhD, is an assistant professor of computer science at the University of Udine. His research interests are formal methods, computer aided formal reasoning, process algebras and Logical Frameworks. He is currently investigating aspects of mobile systems and context-aware computing.

Andrea Selva holds a M.Sc. in Computer Science from the University of Udine. He is currently a research associate at the same university, working on the implementation of mobile systems.

Luca Vassena is a M.Sc. student in Computer Science at the University of Udine. His master thesis concerns the investigation of context representation techniques for mobile systems.

Paolo Zandegiacomo Riziò holds a M.Sc. in Computer Science from the University of Udine. He is currently a research associate at the same university, working on the implementation of mobile systems.