Information Filtering and Retrieving of Context-Aware Applications Within the MoBe Framework

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

[1] Department of Mathematics and Computer Science,
University of Udine, Italy
{coppola, dellamea, mizzaro,
scagnetto, selva}@dimi.uniud.it,
{lucavax, zandepaolo}@inwind.it
[2] Department of Electrical, Management, and Mechanical Engineering,
University of Udine, Italy
l.digaspero@uniud.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 and to new non-trivial problems. 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 (whether physical or "virtual" or a mixture of both) 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 (cellular phones, smartphones, PDAs, etc.) on the basis of the current context the user is in. The context is determined by both an ad-hoc MoBe infrastructure and data from sensors on the mobile device (or in its surroundings). To avoid a "MoBeLet overload", our architecture exploits context-aware information retrieval and filtering techniques, which are briefly discussed.

1 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 about the right platform, delays, etc. Even if it is well known that current mobile devices can be used as computers, since they have computational and communication capabilities similar to com-

puters of a decade ago, how to achieve this goal is not clear. One approach might be to have an operating system continuously monitoring sensors on the mobile device, thus adapting backlight, volume, orientation, temperature, and so on, to the changing environment. 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, in which servers continuously push software applications to mobile devices, depending on the current context of use. Inspired by the Nicholas Negroponte's "Being Digital" term, 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. In Section 2 we recall the state-of-the-art in the related literature. In Section 3 we describe the structure of our model, detailing the key submodules. Section 4 presents some examples of MoBeLets. Section 5 discusses the information retrieval and filtering issues. The last section discusses several practical problems we found developing our first prototype of the MoBe architecture.

2 Related Work

Several research fields are related to this work. Context-aware computing is more than 10 years old, as it was first discussed in [1]. 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, 3]; other researchers searched for a more formal definition [2, 4, 5]; others identified context with location [1] or with location, time, season, etc. [6, 7]. Another related research field concerns mobile agents [8]. 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 [9, 10]. As an example, Google is starting to provide contextual (actually, localized) services as well. Peer-to-peer networks and wireless networks and technologies are of course involved as well.
A much related work is the AmbieSense System (www.ambiesense.com) which provides the user with context-aware information, while the MoBe framework is designed to push both data and applications to mobile devices on the basis of the current context they are in.

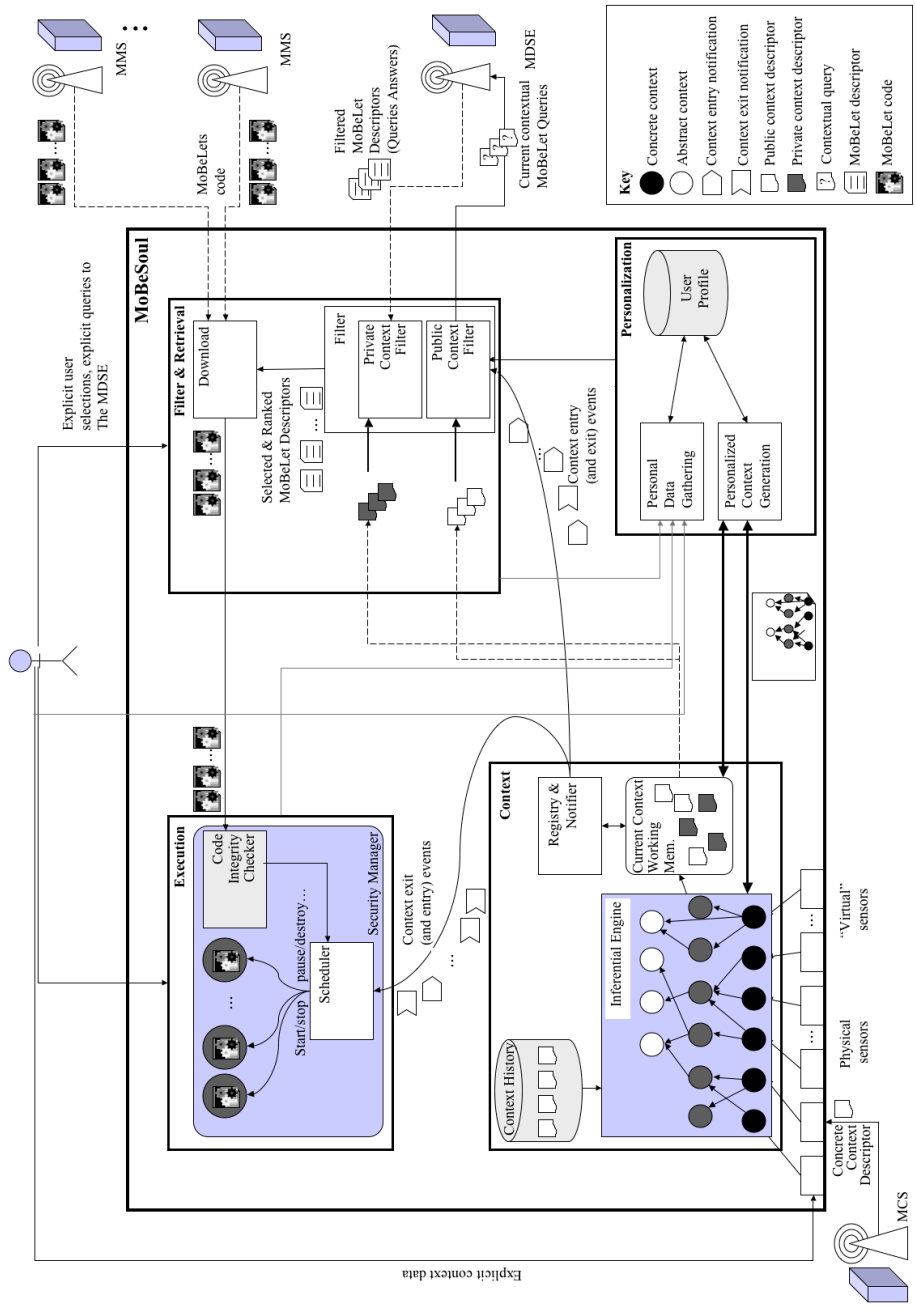

Figure 1. MoBe overall architecture.

3 The Overall Architecture of MoBe

Figure 1 shows the overall MoBe architecture. The mobile device runs a software module called MoBeSoul which 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.

3.1 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, Edge, 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.), some of which might be sent to the device by 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 Concrete Context Descriptor, which might contain a brief declarative description of the current context and a context ID (that, in the case of a Wi-Fi antenna might be the network SSID and/or 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.

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(s) 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 exploits Bayesian Belief Networks) it derives *abstract contexts* (i.e., contexts 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 contexts and the inferred abstract contexts have associated a probability measure representing the likelihood that the user is indeed in those contexts. The inferential engine exploits a database containing the history of past contexts and is tightly integrated with the Personalization submodule (explained later), managing user's preferences, current cognitive load, 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 exit event from that context is inferred.

Examples of concrete contexts are: the temperature is 20 degrees (with probability 0.9); the time is 12:30:00PM (p = 0.99); the MoBeContext ID is 00:0d:93:3d:f2:9c (p = 1.0); 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); and so on.

Contexts are divided into a *public* and a *private* part: the former can be straightforwardly distributed to servers and other entities and contains, for instance, user's approximate location, cognitive load, and so on; the latter is kept private inside the MoBeSoul and contains, for instance, user's exact position, credit card information or some other personal data, and so on. Of course, personal preferences of each user can change the public/privacy status of each item in a context descriptor.

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 events of context entry/exit happen. After the notification, the observers can independently decide, using their own criteria, to request the needed context descriptors to the context module.

Besides the dichotomies concrete-abstract and public-private, we also find useful to categorize the contexts into *artificial* and *natural*: the former are those ascertained from either a MoBeContext Descriptor provided by a MCS or the user him/herself; the latter are those inferred from all the other sensors.

3.2 Personalization Submodule

The *Personalization* submodule consists of two components:

- The *Personal Data Gathering* component collects data about user's preferences and habits and stores them into an internal *User Profile* database. The database contains several different kinds of data directly related to the user, 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. In addition, the *User Profile* database contains data about the behavior of the user w.r.t. the MoBeLets. For example, the database keeps track of 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 Generation* component 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, being an activity that would allow to include unforeseen contexts and inferences even after the system is deployed.

Summarizing, contextual information is derived by the mobile device on the basis of physical, virtual, ad-hoc sensors, and user data; the Context and Personalization submodules infer a more 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. The clear separation between context and personalization might seem difficult, but has important benefits: independent modification of the Context network, independent usage of well established techniques from both the personalization and context-awareness fields, initial development of a non-personalized version of the MoBeSoul, and so on.

3.3 Filter and Retrieval Submodule

The *Filter and Retrieval* submodule is in charge of selecting which MoBeLets to retrieve and to download their code. Its activity is triggered by notifications of context entry and exit events, received from the Context submodule. The *Filter* component 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 on their basis how to express a contextual query (i.e. a query that might contain also elements of the current public context) to be sent to a *MoBe Descriptors Search Engine* (*MDSE*). Furthermore, the MDSE can be also provided with explicit MoBeLet queries directly formulated by the user. The MDSE is in charge of selecting, on the basis of the received contextual query, those MoBeLets that are more relevant to user's current context.

Since not all the MoBeLets selected by the MDSE will be actually downloaded (nor executed), the MDSE does not store and send MoBeLet code, but just *MoBeLets descriptors*. Each descriptor is a simple XML file that contains several structured and unstructured data about the corresponding MoBeLet: a 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* component on the basis of 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 *Download* component retrieves, 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 Execution submodule (see the following subsection).

This design allows:

- To encapsulate inside the Filter component adequate strategies to send to the MDSE the contextual queries, for a more efficient resource usage: the Filter

might send a new query containing the new context descriptors at each context change or, rather, it might collect a certain number of context descriptors (perhaps removing those corresponding to context exit events received meanwhile) before sending a new query. Other possible strategies include sending contextual queries at fixed time points, and so on.

- To separate public and private context data: only the public data is considered to form the queries to be sent to MDSE, but both public and private are used to filter the MoBeLet descriptors received.
- To cache in a straightforward way 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 rather resource demanding MoBeLet, and so on. On the other side, the two stage filtering allows a lower cognitive load to the user.

3.4 Execution Submodule

The last entity of the pipeline is the *Execution* submodule. Its aim is to run each downloaded MoBeLet inside a *Sandbox*, managed by a *Security Manager*, in order to avoid malicious MoBeLets to use resources against user's will. Once downloaded, the code of each MoBeLet is first checked by a *Code Integrity Checker* component which verifies whether it has been corrupted during the download (and in this case asks the Filter & Retrieval submodule to repeat the download).

After this phase each MoBeLet is passed to the *Scheduler*, which has in charge its actual execution. This component is capable of starting, pausing, stopping, and destroying the MoBeLets and is notified of context exit (and entry) events, to stop or pause those MoBeLets that go out of context. Each MoBeLet can register itself with the Registry component inside the Context submodule, in order to be directly notified of relevant context change events.

The MoBeLets that have to use resources outside their sandbox are allowed to access those resources only through the Security Manager, which denies requests that are incompatible with MoBeLet manifest and prompts the user to confirm more heavy resource usages.

3.5 Current Status of the Framework

The framework is at its early stages of implementation. On the server side, simple prototypes for MDSE, MMS, and MCS have been realized (the latter providing only simple localization information; therefore, we currently handle only artificial contexts). On the client side, developed prototype modules include the Execution (with the exception of Security Manager), the Download and a rough Context module. The framework is currently being exploited for domotics applications.

4 A MobeLets Portfolio

In order to show the feasibility of the proposed approach in this section we provide a list of realistic MoBeLets. Here we present only few of them; the reader can further imagine several sensible examples to confirm that the scenario described above is feasible.

MoBeDisc: MoBeLet designed in order to provide to the user the following information and services: map of a discotheque, illustrating the different rooms along with the relative activities; possibility to get in contact with people with similar interests (matches inferred according to the information of the user profiles); in case of emergency the nearest safety exit is indicated.

MoBeCrash: in the case of a car accident this MoBeLet can automatically perform an emergency call, alerting both the rescue units and the police and an information center which, in turn, can alert other car drivers to pay attention when passing near the zone of the car accident. The information sent by the MoBeLet can be very accurate, including the geographic position, the identity of the driver and other useful information items obtained by querying the computer system of the car. Thus, the rescuers can have a clear idea of the number of cars involved in the accident and plan an adequate intervention.

MoBeBus: this MoBeLet informs the user about the route performed by the bus, displaying all the stops. It can also suggest the nearest stop for the desired destination, alerting the user when the bus is approaching it.

MoBeShop: news about particular offers and discounts. Possibility to be informed about the availability of a given product and to pay electronically (with an eventual separate shipment of the goods), using the mobile device by means of a secure connection to a payment gateway.

MoBeWakeUp: the usual alarm clock application, already installed on today mobile phones. Since the alarm clock is used in well specific contexts (i.e., before going to sleep and at wake up), it might be a (very simple) MoBeLet as well, thus freeing the resources when it is not used (during the day).

MoBeJam: this MoBeLet downloads a traffic profile of the current road and monitors the car sensors to detect abnormal traffic situations. In case a traffic jam is detected, the data are sent back to a centralized server, which informs all the other MoBeJam users about the traffic jam and suggests them alternative roads.

MoBeQueue: when standing in a queue, a user can use this MoBeLet in order to register him/herself, obtaining a virtual ticket with its position in the queue. Then he/she can make some queries in order to know the average time needed to serve him/her and other useful information.

MoBeSales: when the traveling salesman/saleswoman comes back to the factory, this application updates his/her records about the products (including newly produced ones) and the payment situations of his/her customers.

MoBeMuseum: when a tourist enters a museum empowered with the MoBe technology, this MoBeLet can be both an effective navigation aid (showing to the user a detailed map of the building with clear indications about the rooms and their contents) and a rich source of information about the item the user is currently looking at.

MoBeAirport: when a traveler enters into an airport, this MoBeLet can warn him about the scheduling time of his/her check-in. Moreover, the right path through the terminals can be indicated by means of a visual map. Other useful information displayed by the MoBeLet can be the locations of toilets, emergency exits etc.

MoBeHome: a useful suite of MoBeLets being able to control several facilities of a modern house. For instance, it is possible to enable/disable the alarm system (when exiting/coming back to home respectively). Moreover, for each room it is possible to control the temperature, to switch on/off the lights and to control several household-electric appliances (e.g.: TV, microwave oven etc.).

MoBeCar: this MoBeLet can ease the interaction with several electronic components of a modern car. For instance, it can automatically program the satellite navigator in order to show the best route to the next destination, according to the current time and the appointments recorded in the PDA's agenda. Moreover, it can show diagnostic information about critical components of the car by communicating with the on-board computer, alerting the driver if there is something wrong.

MoBeLibrary: when the user wants to quickly find a book about a particular topic, without disturbing the librarian, he/she can use this MoBeLet. Specifying some search keywords, it is possible to query the library database in order to find only the relevant records, suggesting the best path to reach them from the current position in the building.

5 Filtering and Retrieval of Context Descriptors, MoBeLet Descriptors, and MoBeLets

Following MoBe approach, dozens, hundreds, or perhaps even thousands of MoBeLets will be potentially relevant to a user in each instant, depending on the current context. To filter the MoBeLets in an effective way (i.e., to avoid a "MoBeLets overload" on the user), an appropriate and novel mix of Information Retrieval (IR) and Information Filtering (IF) techniques has to be exploited, and appropriate contextual IR models have to be designed. In this section we present a brief and preliminary discussion about several IR issues that are relevant to MoBe, and we show how MoBe would provide a rather novel environment for IR.

A first issue is the dichotomy IR vs. IF, i.e., whether to adopt a *push* or *pull* approach. MoBe does not endorse a simple "MoBeLet filtering" approach, since it would not be viable to continuously waste the available bandwidth by pushing MoBeLets that will usually not be executed. Conversely, a pull-only approach would leave to the mobile device, or to the user, the responsibility of continuously asking for the right MoBeLet. Instead, MoBe endorses a threefold approach that integrates: (i) filtering of context descriptors, which are pushed on the mobile; (ii) on the basis of the filtered context descriptors, automatic query construction and retrieval, through MDSE, of MoBeLet descriptors, which are therefore pulled by the mobile device; and (iii) filtering of the retrieved MoBeLet descriptors, on the basis of the private parts of the context descriptors.

Also, an IR component is needed when the context mechanism does not work, e.g., because a context descriptor fails to reach the mobile device, or because the inferen-

tial mechanism makes a wrong inference, or simply because the current context is not foreseen by the designers. In such a case, the user him/herself can run a search for MoBeLets, manually inputting a query describing the context he/she is in.

A second dichotomy is whether MDSE should be implemented as a content-based retrieval system or whether a social/collaborative approach could be effective as well, if not more. Collaborative filtering might have an important role since MoBeLets, being Java code, should be indexed manually through the MoBeLet descriptors. Actually, one might envisage descriptorless MoBeLets, retrieved on the basis of a collaborative approach: if two MoBeLets tend to be downloaded in the same contexts, those two MoBeLets can be considered as similar: therefore, if one of the two happens to be downloaded in a new context, it is likely that the other one will be relevant to the new context as well. The differences from standard collaborative filtering, still to be fully understood, include at least two issues: MoBeLets are retrieved on the basis of automatic queries (rather than filtered) and the feedback is implicit (i.e., based on MoBeLet downloading rather than user preferences/selections).

A third issue is the significant relationship with XML retrieval, which is gaining much attention today as witnessed, for example, by the INEX exercise. Both context descriptors and MoBeLet descriptors are XML files. Context descriptors are filtered as incoming XML documents, and then used as XML queries for MDSE, which returns MoBeLet descriptors as retrieved documents. MoBe might be an interesting environment in which to exploit the results obtained in the in XML retrieval field.

A fourth related field is software retrieval, i.e., the usage of IR techniques to retrieve items from a software repository. Again, here the situation is different since, within MoBe, it is neither the programmer which is interested in retrieving the relevant software entity (from a software library), nor the user which wants to install a new application, but a software module, on behalf of the user, which is interested in proactively retrieving the right software to be executed in the current context, which is continuously changing.

A fifth, and final, issue that is crucial for this research is the relationship between MoBe approach and context-aware retrieval [10]. So far, approaches to exploitation of context in IR have been focusing mainly on enriching a search engine query by adding information about the context (usually, geographical location only) with the aim of improving the precision of the search, and in defining innovative IR models that are capable of taking the notion of context into account (e.g., [12]). Again, MoBe has a novel standpoint: the context is the main, if not the only, component of the query; the query is automatically generated; the context is partially provided by the environment; and so on.

Thus, summarizing, MoBe is different from each of the above issues, for various reasons, and tries to integrate all of them. Turning to more theoretical issues, one might discuss which context retrieval models would be adequate in the MoBe scenario. Classical IR models (probabilistic, vector space, tf.idf, etc.) give different importance (weight) to terms with different features (e.g., very common terms, aka stopwords, are not used/useful for retrieval). Some models with similar features should be designed for MoBe: a rather common (and general) context should be less important than an uncommon (and specific) one, i.e., the former should be often filtered out, whereas the latter should be sent to the MDSE to obtain the corresponding (and more specific) MoBeLet descriptor.

Let's briefly see which kind of issues we need to study, and let's start by taking into account location only. On the basis of location only, a natural approach is to apply a weighting scheme analogous, for instance, to tf.idf: we might name it *lf.ilf* (location frequency – inverse location frequency). Given two MoBeLets m_1 and m_2, we refer to the locations in which m_1 and m_2 can be executed on user's mobile device by $l(m_1)$ and $l(m_2)$. If $l(m_1) \subseteq l(m_2)$, then m_1 should be preferred over m_2. For instance, if m_2 were the MoBeLet of a shopping mall and m_1 the MoBeLet of a shop inside the shopping mall, this means that when the user is in the shop (and therefore in the shopping mall too), m_1 should be preferred over m_2, as it is reasonable. However, $area(l(m_1)) < area(l(m_2))$ is not enough to assume that m_1 should be preferred over m_2, since $l(m_1)$, though smaller, might be a more common place $l(m_2)$ (both for a particular user and for the average user), and thus the choice between m_1 and m_2 is not obvious.

The situation becomes more complex when extending the above argument from location to context: if we denote by $c(m_1)$ and $c(m_2)$ the contexts in which m_1 and m_2 can be executed on user's mobile device, we do still have that if $c(m_1) \subseteq c(m_2)$, then m_1 should be preferred to m_2, but context inclusion is a much more complex matter than location inclusion, since it is not based on topological reasoning only.

6 Discussion and Open Problems

The architecture proposed in this paper is still in early development phase and in this section we briefly discuss some open issues. Since the MoBe framework is still in an early development stage, this section is rather speculative. Nonetheless, the following discussion can be useful in order to forecast some realistic scenarios of deployment of the MoBe architecture, highlighting in advance some crucial implementation problems and possible solutions.

6.1 Scalability Issues

MoBe architecture should be scalable for what concerns MCS and MMS: simply, 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 contextual queries will be sent to the main MDSE server (when the ID can't be provided). In the last case, there is the risk of overloading the main MDSE 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, thousands of queries per second (let's say 1000). 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., $(10^{12}) / (24 * 60 * 60) \approx$

10^7 ca. context change per second. Not all of them will be taken into account from MoBeSoul, since the Filter component selects and queues some public context descriptors, but let us be pessimistic and assume that this does not decrease 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 times higher than today's Google, but it is not so frightening; at worst, we might deploy 1000 MDSE 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, MDSE and MMS can be the same server.

6.2 Structured vs. Unstructured Approach

Turning to more general issues, we see two trends in current computer science and web technologies. The first trend 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 made of structured data; a MoBeLet descriptor 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 [11]. Also comments by other users are exploitable through social and collaborative filtering techniques.

6.3 Applications vs. Data

Within MoBe, applications are sent around, not just data. Of course, this is an arbitrary distinction. 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, which, we believe, is a short-sighted view or from systems like AmbieSense which are limited to sending context-aware data to mobile devices. 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: someone has to write a particular kind of content (i.e., software, not just data – this is another reason, besides portability, to use J2ME: it is rather easy to find Java programmers today); sending applications might lead to a proliferation of malicious MoBeLets (i.e., viruses); privacy issues, that we

tackled by distinguishing between public and private context parts, are indeed much more complex, and so on.

6.4 Context and Personalization

Downloaded and executed MoBeLets can be selected not only on the basis of the current context, but also exploiting the Statistics & Log databases inside the Filter & Download and Executor submodules.
This is another point in which the above mentioned separation between context and personalization is, although tricky, advantageous, since context histories management can be simplified and empowered by such a separation. Indeed, statistics and logs of MoBeLet usage by a user are rather sensible data, and therefore 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 MDS, 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 not simple issue that deserves further work.

7 Conclusions

We have presented a novel approach for retrieving and filtering context-aware applications on mobile devices, and discussed the important role that information retrieval and filtering techniques have in this novel scenario. Since MoBe architecture is not fully developed yet, there is plenty of work to be done, on both the theoretical and practical sides. We are currently working at refining the contextual model we sketched in Section 5, at a complete and formal definition of context, and at completing the implementation of the infrastructure. We are also implementing the first MoBeLets and a set of MoBeLets descriptors to evaluate our approach.

References

1. B. N. Schilit and M. M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network* 8(5): pp. 22-32. September/October 1994.
2. G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research, 2000
3. B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 85-90, Santa Cruz, California, December 1994. IEEE Computer Society Press.
4. A. Schmidt, K. Asante Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In Proceedings of First International Symposium on Handheld and Ubiquitous Computing, HUC'99, pages 89-101, Karlsruhe (Germany), September 1999. Springer Verlag.
5. A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, June 1999. <ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf>

6. P. J. Brown, J. D. Bovey and X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications* 4(5): pp. 58-64. October 1997.
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, Editors. Oxford (UK), 1998.
8. M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.
9. B. J. Rhodes, P. Maes. Just-in-time information retrieval agents, *IBM Systems Journal*, 39(3-4) 685 – 704, 2000
10. 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, Springer Lecture Notes in Computer Science, Vol. 2954, pp. 227-243, 2004,
11. R. Gonzales, K. van der Meer. Standard metadata applied to software retrieval. *Journal of Information Science*, 30(4): pp. 300-309, 2004.
12. J.R. Wen, N. Lao, and W.Y. Ma. Probabilistic Model for Contextual Retrieval. Proceedings of ACM SIGIR '04, pp. 57–63, 2004.