Environment Awareness:
Telling applications about the present and the future

João Garcia, Luís Veiga, Paulo Ferreira

{ **jog**, **lveiga** }@gsd.inesc-id.pt, **paulo.ferreira@inesc.pt**

INESC / IST, Rua Alves Redol 9, 1000 LISBOA, Portugal

http://www.gsd.inesc-id.pt/

Abstract

Computer mobility has the potential to let users take advantage of computers in new, varied and changing environments. Applications would be able to do much more for the user (e.g. providing hints concerning power management, network connection establishment, synchronization opportunities, update propagation, location-based services, etc.) if they were aware of their computational and physical environment.

However, the development of applications with such functionality is obstructed by the large amount of different modules, sensors, platforms and ad-hoc solutions, i.e. there is no middleware supporting such applications and their development in a uniform way. In addition, current approaches do not take into account neither future user actions nor his habits; consequently, the user is left alone w.r.t. a large number of decisions concerning the best usage of the computational resources he has access to.

Therefore, our goal is to develop an infrastructure, i.e. an environment awareness system, supporting the interaction between applications and any computer-based mechanism able to provide clues regarding the surrounding environment and relevant future conditions or actions.

This paper presents the architecture for an environment awareness system (EAS): an operating system module that acts as an intermediary between applications and all mechanisms available to assess the surrounding environment. The EAS provides additional environmental information based on simple reasoning and forecasting derived from gathered environment properties. Thus, the system is capable of anticipating future user behaviour so that applications may advice him in order to take full advantage of the resources available therefore increasing his productivity. Finally, the EAS aids the exploitation of many synergies between different perception modules (or sensors). Applications can derive these synergies by simple reasoning mechanisms using data from several of such modules.

1 Introduction

Computer mobility has the potential to let users take advantage of computers in new, varied and changing environments. For example, we envisage a scenario in which a user will want to access data using a PC in his office, using a laptop while in the airport or in the hotel, using a PDA in a taxi, etc. The user wants to live in this "data ubiquitous world" with no other concern besides doing his own work and, as much as possible, keeping on working in spite of any system problem that may occur (e.g. network partitions). In addition, he wants to take advantage of any resources that may be available as he moves from one place to another (e.g. cheap wireless connections) and does not want to spend time deciding system specific issues such as:

- What is the best moment for recharging my laptop's battery (so that I'm certain to work continuously for the next 3 hours)?

- While at the airport should I synchronize my PDA with my PC using the available wireless GSM connection available right now or later (so that I will have all the data I need in the PDA for the meetings I'll have today)?
- While at the hotel lobby should I copy the data from my PDA to my friend's PDA because he will be connecting to the fixed network first and my data can then be safely stored into some server?
- Etc.....

The problem concerning these scenarios, related to application development and execution, is twofold: i) the large amount of different modules, sensors, platforms and ad-hoc solutions that currently available increase the difficulty of building such applications; a middleware layer supporting such applications and their development in a uniform way is clearly desirable; ii) current approaches do not take into account neither future actions of the user nor his habits, thus the user is on his own regarding a large number of decisions (as those previously exemplified) concerning the best usage of the computational/communication/physical resources he has access to.

Therefore, the increasing geographical mobility of devices and the mobility of data among devices require applications to be aware of the environment around them and of the possibilities that the environment can provide to applications. An application's environment is composed by the computer where the application is running, by the network(s) to which this computer may be connected, the surrounding devices and by the involving physical environment. Traditionally, in the absence of a structured method to provide this information to applications, application developers must explicitly program code to query the computer's devices (i.e. network card, GPS, RFID reader, etc…) and from there derive conclusions about the environment.

There have already been many attempts to define a set procedures and metrics for evaluating specific environmental properties: CPU and network benchmarks, memory and battery charge measurement utilities, GPS and location devices, etc… Our goal is to develop an infrastructure, i.e. an environment awareness system (EAS), supporting the interaction between applications and any computer-based mechanism able to provide clues regarding the surrounding environment. The EAS provides a set of standard application programming interfaces (API) to access the system's environment perception modules (EPM)[1], simple mechanisms for combining EPMs and a database of past, scheduled and forecast environment conditions. An application can perceive its environment via an encompassing set of environmental properties, which can be obtained through a set of standards based EPM, and which can be queried or complemented by applications in order to improve or extend the applications' functionality.

In addition to existing and future EPMs, an EAS exploits many synergies between EPMs, which can be derived by simple reasoning mechanisms using data from several EPMs. As a simple example, personal organizer information regarding the future location of a device can aid in adjusting power management so that the device is able to operate until the next place where it can be recharged. Additionally, and most important, an EAS is able to forecast future conditions and/or user actions by detecting patterns in the temporal series of past conditions.

In the rest of this paper, we begin by presenting the architecture of the EAS. Then, we list a number of EPMs that could currently be integrated into an implementation of an EAS and describe practical applications. Then we describe how our design relates to existing systems. Finally, we present some of our future work and conclusions.

[1] An environment perception module (EPM) is a component of an operating system or user environment capable of detecting, foreseeing and/or memorizing an environmental property.

2 Architecture

The environment awareness system (EAS) is an operating system module, which provides applications with a simple and structured mechanism to query the computational and physical environment surrounding the device where the applications are executing. An EAS (see Figure 1) consists of:

- An API layer accessible to applications for probing environment conditions and for setting up callbacks.
- A callback registry.
- A schedule of EPM probe actions.
- A repository of past, scheduled and forecast events.
- A forecasting model for calculating probable future conditions.
- Several environment perception modules (EPMs).
- A remote invocation interface.

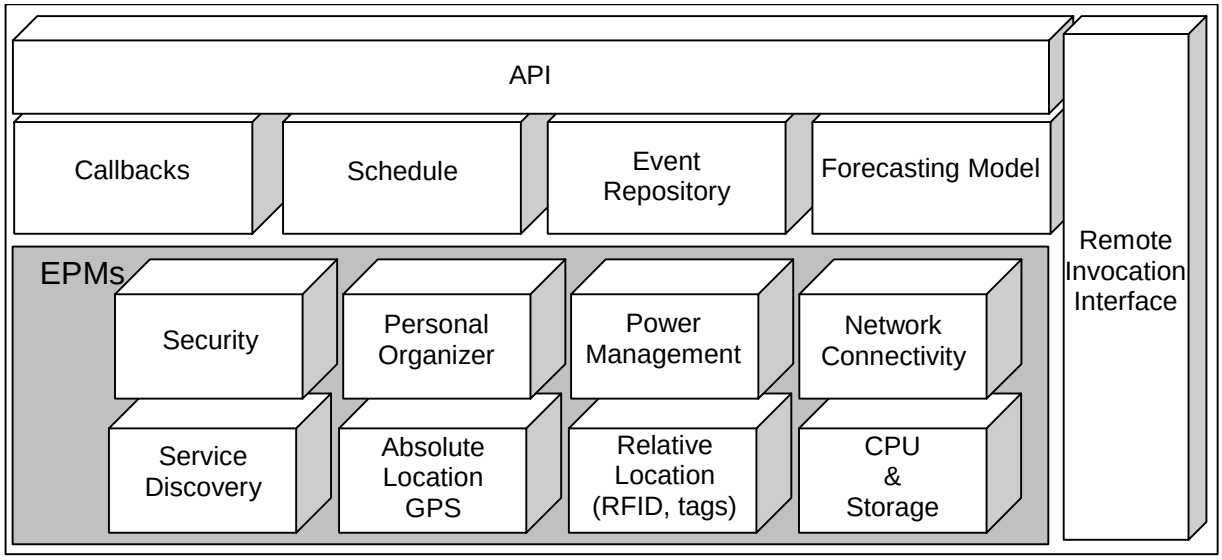

Figure 1 - Architecture of the Environment Awareness System (EAS)

The EAS' runtime is based on events. An event is a set of attribute-value pairs representing a change in one or more environmental properties. Consequently, a relation must be established between each sensor and an abstract real-world attribute of the environmental property it is able to detect. Both attributes and values are strings. An attribute is the name of an environmental property such as "AbsoluteLocation" or "NetworkConnectivity". Each attribute has a domain describing the values it can assume, e.g. "NetworkConnectivity" can be "None", "Poor", "Medium" or "Good". A value describes the status of a property such as "443.23N 217.98W" (a possible value for "AbsoluteLocation") or "None" (a "NetworkConnectivity" value).

Users can perform queries and callbacks. Queries relate to the current situation and will return an indication whether pretended properties are true. Callbacks are a request for a notification to occur when certain user-defined conditions happen in the future. Callbacks may also have an expiry date. Regarding space, queries and callbacks can be directed to the local device or to another device. Other devices can be specified using a machine name or network address or a geographical radius with respect to the local machine. Within a query or callback, attribute values can also be specified as ranges.

Additionally, there are situations, i.e. labelled events. Any situation can be classified with respect to the environment properties it describes and to other situations. Users can submit labelled situations, which are stored by the system and can be referred to in subsequent queries (Figure 2).

There are two aspects to the characterization of an environment: physical characterization and computational characterization. In each of these aspects, an environmental property can be classified in multiple categories.

The hierarchical organization of environment properties and of events combine on the one hand a simple way of manipulating information about the hardware and, on the other hand, an expressive mechanism to describe situations to which applications want to respond.

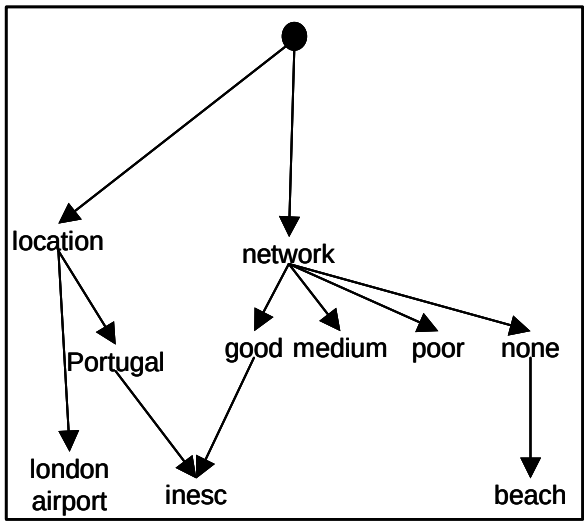

Figure 2 – Example of a user-input situation hierarchy

2.1 API layer

The API layer allows applications to interact with the EAS by submitting queries, registering callbacks and adding new elements to the property hierarchy. The most relevant methods in the EAS API are:

```
CallbackReturnRecord RegisterCallback(Situation, ExpiryDate, GeoReference,
MethodReference);
```

The RegisterCallback method allows applications to register a callback by submitting a situation, which it would like to detect. This situation is expressed as a list of (attribute, value range) pairs. The callback includes an expiry date indicating until when the application is interested in being notified and a geographical reference defining the relevant radius for the desired situation. Should the callback be activated, a method reference that is also passed to the EAS, will be invoked to notify the application.

```
QueryReturnRecord Query(AttributeList, DateRange, GeoReference);
```

The Query method is used by applications to inquire about current conditions. The EAS receives a list of attributes

```
RegisterSituation(Label, ParentList, AttributeRange);
```

RegisterSituation allows applications to add new labelled situations to the existing situation hierarchy. The calling application supplies the situation's label, its parents in the hierarchy and a list of attribute ranges. For each relevant attribute, the range of values within which the given situation is said to occur is provided.

```
SituationList QuerySituation(Label);
```

QuerySituation is the method used to navigate the situation hierarchy. It receives a situation's label and returns the list of all descendants of that situation.

```
RemoveSituation(Label);
```

RemoveSituation removes the situation with a particular label from the hierarchy.

```
RemoveCallback(CallbackHandle);
```

RemoveCallback is used to remove a callback before it expires. The callback to be removed is identified by a

CallbackHandle, which was included in the CallbackReturnRecord returned to the application when it had previously called RegisterCallback.

Any query or callback will be converted into a sequence of numeric values (or ranges) describing the desired values of a set of environment properties as situations included in the query are replaced by concrete physical values.Queries are evaluated immediately, whereas callbacks are stored.

2.2 Callbacks

Callbacks to applications are an essential component of an EAS in order to allow applications to be notified of future conditions: location, battery status, connectivity, etc… Callbacks are associated to changes in the values of one or more environment properties. Callbacks are registered according to the environment properties they involve. Each time the EAS detects a change in an environment property, the set of registered events is searched for callbacks, which may be triggered by that change. Callbacks are stored in a data structure indexed by environmental property and will be referred to by all the properties included in this callback store. Whenever a change occurs in an environment property, e.g. "Power Supply", all callbacks that refer to that property will be reviewed to check whether all necessary conditions have been met and whether the application, which submitted them should be notified.

If the callback's requested situation is already scheduled or forecast when the callback is registered, the EAS will return the first future time and date when the situation will occur. If a callback for a particular property is registered and the property assumes the requested value more than once, the callback will always be acted upon until it expires.

2.3 Probe Schedule

An EAS periodically probes the physical devices that provide environment property values. Many environment properties are machine characteristics that do not change frequently, if at all. Therefore, they can be stored in the EAS and only be recalculated when the local hardware is reconfigured. There is a schedule of the next moment when each of the available devices must be probed in order to update the properties it relates to. Probe actions can be disabled during periods for which there are no registered callbacks.

2.4 Event Repository

The event repository stores all changes in environment properties that are detected by the EAS. Events are labelled as observed, scheduled, forecast or probable. Observed events are the result of actual probes of EPMs (see section 3); forecast events are calculated by the forecasting model (see 2.5); scheduled events are explicitly inscribed in personal organizer information and probable events are derived from personal organizer data (see 3.7).

The event repository is indexed by time so that the current status of all environment properties is easily obtained by looking back in time and collecting the last moment when each of the properties changed.

2.5 Forecasting Model

A forecasting model was included in order to be able to forecast future situations and to allow applications to perform proactive actions regarding future environment conditions. Many future conditions can be deducted from personal organizer schedules but frequently that leaves a significant amount of time with unknown occupation. This can be complemented with future events forecast based on past recurring conditions. This is performed by an ARIMA [MAK98] forecasting model for discrete variables, which analyses the event log and tries to detect temporal patterns for each of the detected environment variables and stored labelled situations and inserts new events, adequately labelled as "forecast" into the event repository. Time series, which are the input of the forecasting algorithm, are generated by analysing the

event repository and deriving which property values were valid at each of the sampling times. The amount and precision of forecasting computation that is performed can be adjusted to the devices computational and storage possibilities.

2.6 Remote Invocation Interface

An EAS provides a remote service that can be queried by EASs on other machines. This opens up the possibility of sharing information among a group of devices so that in some cases other trusted devices can work as extensions of a device. In general, the remote interface is used for queries that refer to devices around the current device and are most useful to allow applications to take advantage of available resources around it. An interesting example is using future connectivity information to decide whether to pass a file or e-mail message onto a trusted mobile device, which will have adequate connectivity to send the data earlier than the computer where the data are currently stored.

3 Environment Perception Modules

Environment Perception Modules (EPMs) are the components of an EAS that interact with the device's physical devices and assess current environment properties. They represent the greatest challenge in building an EAS due to the issue of the heterogeneity of devices. For instance, a user would like to assess memory occupation through the EAS by querying the "MemoryOccupation" attribute. However, the mechanism to obtain memory characteristics from the operating system varies between OSs. Each EPM has to include code to assess it's properties in the current OS. This may seem a daunting task, but currently many of the OS modules and devices that provide environment properties are accessible through standard APIs (e.g. ACPI, see 3.3, for power management), which simplifies the EPM code greatly. In those cases where such standard APIs are not available we are aiming to develop EPMs able to interact with the most common OS, specially those for mobile devices (Windows CE, Palm OS, Linux, Simbian). However, in those cases where neither a standard API nor an OS interface is available, being able to interact with a large number of devices may be a significant problem. The EAS adjusts its behaviour to the absence or inability to operate of an EPM, and consequent absence of means to detect a particular environment property, by falling back to previous estimates or by rejecting queries and callback registrations, which refer to that property.

In the following sections, we describe several of the most important environment properties and the mechanisms to obtain them.

3.1 Processing Power

A property of a device that is essential to determine whether to perform expensive computations is the device's processing power. This is best evaluated with a benchmark. There are a number of benchmarks available. We suggest two possibilities: LINPACK and HINT.

LINPACK does a series of linear equation calculations. It has the advantage of having been ported onto a very large number of architectures [DON01] and having been widely accepted as a flexible CPU benchmark.

HINT, a more recent benchmark than LINPACK, is an interesting option because it can be used to represent many different benchmarks. HINT performs the same calculations using a variable sized set of data. Results [GUS95] show that varying the data size used by HINT, makes it emulate many other benchmarks, which demonstrates its versatility.

Running a benchmark is by definition a resource consuming operation. Therefore benchmark results are cached at each machine. These results will be updated if it is specifically requested or if the system detects a new machine configuration for which the benchmark has not been run yet.

HINT expresses its results as QUIPS (Quality Improvement Per Second), which represent progress towards a solution, in this case solving a numerical integration. Ranges of QUIPS values can be delimited to provide a user-meaningful measure of the CPU's power (e.g. "CPUPower" values of "low", "medium", "high",

"superior"). Most machines are also able to provide users with the degree of CPU availability which is also convened to the user as a percentage of "CPUOccupation".

Additionally, processing power objectives can be suggested by applications and may lead to adjustments in CPU speed. In that respect, it is also worth mentioning the Transmeta Crusoe chip [KLA00]. Usually it is suggested that applications should adjust to available resources. The Crusoe CPU does the reverse adaptation by regulating its CPU clock so that it is only as high as it is necessary for the applications currently running.

3.2 Memory

Information about a device's memory capacity is also useful to assess a device's ability to perform heavy computation. Often in applications there is room to adapt to memory availability. For example, an application can decide whether to cache data or not, whether to pre-calculate temporary values, how detailed to generate graphics, etc... The size of the total and available memory to applications is generally provided by simple system utilities. The EAS interest regarding memory is in its forecasting and callback abilities. For instance, once a pattern of memory use is known at a device, memory intensive applications can choose an appropriate moment to run or even adjust their memory use in relation to available memory.

3.3 Power Management

Power management is probably the area where there has been more research on adaptability, since battery technology is one of the major hurdles to an even greater use of mobile technologies.

Our first effort will be to interface EAS with ACPI [ACPI99]. ACPI (Advanced Configuration & Power Interface) is a standard which defines how the operating system should manage the power of a machine's devices. Basically, it provides an OS interface, which allows applications and device drivers to query current power conditions such as the percentage of remaining battery charge or the existence of a power supply connection.

As will be demonstrated in the "Applications" section, power management can be significantly enhanced when there is information about the future. For instance, power management can be adjusted w.r.t. the device's future location. For example, a variable speed CPU can be adjusted so that the existing battery charge lasts until arrival at the next location where the device can be charged.

3.4 Absolute Location

Global Positioning System is the most widespread means of geographic location. It provides absolute geographic location based on satellite microwave signals. Currently, more and more devices are equipped with GPS. An EAS working on a computer with GPS information can associate user provided locations ("office", "home", "supermarket") with GPS positions. This can provide excellent feedback for location dependent applications such as a personal organiser. GPS information can also be used to automatically configure wireless network interfaces.

3.5 Relative Location

In addition to absolute location, there is a growing number of locations where location and tagging mechanisms are used in order to detect devices. Many of these systems, e.g. the RFID tagging system, can be used in reverse to allow devices to know where they are. A particularly interesting use of location mechanisms is transport ticket tagging. There are a growing number of transportation authorities using RFID to track passengers and this can be used by an EAS to detect which type of transportation is being used. This can be interesting, for example, for selecting user notifications: there is no point in letting a user know that he is right next to the post office where, for instance, his personal organizer knows that he wanted to buy stamps, if he passing under the post office inside a subway train at 60 km/h.

3.6 Connectivity

Connectivity is one of the most variable characteristics in mobile environments. Furthermore, computers are equipped with a growing number of network interfaces (Ethernet, Phone, GSM, IR, Bluetooth, etc...) which raises the issue of knowing not only how good the current connection to the network is ("NetworkQuality") but also which network interfaces should be chosen ("BestNetworkAdapter").

Regarding general network evaluation, it is very difficult to obtain a general metric of network quality. Hence, we chose two ways of looking at network quality: quality of the connection to the gateway and point-to-point quality to destination. Assessing the connection to the gateway is interesting because it is usually the worst portion of the network in wireless networks which are the most popular with mobile devices and because it is the only portion of the network which will be common to any communication.

Assessing current point-to-point connectivity is not technically more complex than assessing the connection to the gateway. However, we must decide which destination points must be chosen, since we cannot evaluate all possible destinations of network communications. We envisage to maintain a cache of previously used locations in the form of a 2D matrix relating origins and destinations, so that the EAS is able to test network quality to those machines, which are most frequently connected to.

Although, we have not foreseen it at EAS level, it would be interesting to associate network quality with cost of service in order to provide the user with the best possible choice in network connections.

3.7 Personal Organizer

Most computers currently have personal organizer software, which can be a source of valuable information regarding the future environment of the device. This information is not presently taken into account by computing systems.

Microsoft's personal organizer, Outlook, provides automation objects APIs, which can be used to programatically acquire information from Outlook and store it into a future location table in the EAS. This information could also be acquired from other similar software with a specific EPM.

Once integrated in the EAS, this information can be used to somehow predict future environment properties and its values. These can be further processed to provide applications with hints and to determine which events, of those that are stored in the event repository, should be fired and the corresponding callbacks invoked. Thus applications can, asynchronously, send results to hardware not present, assured that data will be sent to it when it connects to the network and becomes available.

Whenever a location is provided by a personal organizer appointment, the situation hierarchy is scanned to check whether that location has been submitted as a situation label. If so, for each of the attributes contained in that situation, scheduled events are inserted into the event repository.

Naturally, forecast events generated with the forecasting model do not collide with scheduled events provided by the personal organizer, which have a higher degree of certainty and therefore override forecast events.

It is likely that some users maintain schedule information only in one device, e.g their PDA, although they use several computing devices. However, any device should be able to obtain personal organizer information through the EAS remote interface.

Location is a particularly important characteristic of the environment. Therefore, a index of previous locations, and of the environment properties that existed there, is kept in order to derive more information about future locations which are known from scheduled events (see 3.7). Events, which have a high probability of occurring at a scheduled event's location, will be added to the event repository as probable events.

An example of a group of probable events are those relating to connectivity. The future connectivity of a device can be established based on a combination of information from personal organizer software and the table establishing a relationship between location and probable connectivity to other points in the network.

3.8 Services

Once a computer is connected to a network, there are a number of protocols that can be used to detect services and/or verify their availability. A classical example of device that, once connected to the network, should become promptly available to all other participants is the printer. This example can be portrayed using different technologies.

Using present Jini [JINI01] technology, the aforementioned device, the printer, should register itself, i.e., the services it provides, in a lookup service. If the lookup service is known beforehand, the printer contacts it; otherwise, the latter will engage in a UDP multicast to a specified well-known port (4160) and wait for some response. This response holds a *registrar* object that will act as a proxy to the lookup service, typically using remote method invocation. This proxy allows the printer to register its services in the lookup service.

Let's imagine the printer provides two major services: document printing and queue information. The printer should register two services in the lookup service that will store a specific, arbitrary object for each service. These will be the objects delivered to any client that requests any of the two services. When a client queries the lookup server, after discovering one, for a printing service, it will get, in response, the service object stored by the printer. This object is specifically implemented to communicate, via RMI, with the *real* printing service in the printer, which is listening for calls.

With Microsoft's Universal Plug and Play [UPNP00], the printer should, once connected to the network, advertise its services multicasting discovery messages stating its availability (ssdp:alive) without the use of any lookup server. There would be multicasts describing the printer root device, its embedded devices (it could be just one) and the services provided, in this case, the two services mentioned earlier. These messages are in essence, identical to responses given to clients, named control points, requesting discovery services searching for a printer. Clients can, then, obtain further service description with URLs for control and eventing, and XML data describing the actions performed by the service and its arguments. The actual invocation of the service is achieved using SOAP.

Besides the obvious differences, namely the existence of lookup servers and dynamic code downloading only present in Jini, these two approaches can be made seamlessly interchangeable. We can extend this classical example resorting to environment awareness data. An application, after performing some work, e.g., producing some report, can register its interest in being informed of a combination of events: good network connectivity and printer availability. These two properties have different value domains but can be used in a combined event. Once stored in the event repository, the system can use both Jini and UPnP to advise the application whenever a printer is connected to the network. At that moment, the combined event would be fired and the corresponding application callback would be invoked. A dialog could pop-up asking if the user would be interested in printing the saved data. Thus, environment awareness can be achieved using different technologies in an integrated manner.

In summary, our aim is to allow a mobile device to keep track of the most useful computational resources around, such as communication (web access, fax, telephony), storage, printing and other peripheral devices, by querying the most common service location facilities.

3.9 Future Developments

Naturally, the EPMs that were presented above are a fraction of those that will exist and be of interest to applications in the future. We can envisage many situations were other environment data and services become relevant: assessing lighting to manage solar charged batteries, using voice processing services at nearby devices, etc.... Naturally, as EAS implementations evolve it will be able to incorporate such novelties as new attributes in its event repository.

4 Application

As we showed above, once simple elementary pieces of information about the environment are collected and associated, a wealth of knowledge emerges. Using simple queries to the EAS, many useful applications can be created. Examples of possible applications based on EAS information are described now.

Location aware personal organiser

It is common for current personal organiser software to notify users based on the dates of events. In particular, a personal organiser that resorts to EAS' location awareness would be able to notify users based on the proximity to the places where tasks have to be performed. For example, a user notes in his personal organiser that he must send a letter by surface mail (which implies going to the post office).

Thus, whenever the user approaches a post office he will be notified accordingly. This would involve adding the location of the post office to the part of the situation hierarchy related to location, and register a single callback with the EAS. This callback would be invoked for a situation with the following properties: location within a user-defined radius of the post-office, during the post-office's open hours and possibly without transportation (meaning "on foot").

Future location aware connectivity

In this case the goal is not only to minimize the cost associated to network connection but also to ensure that data is propagated on time to other devices or to some server (in the fixed network). In other words, given some data that must be transmitted/received to/from some other device the system should be able to notify the user about the appropriate moment to do so, or to make all the needed operations automatically.

This requires the system to be aware not only of its physical location but also where it will be in the future and the kind of connectivity available in such places. A callback must be registered in order to be invoked when the connection available is the less expensive taking into account not only current and forecasted locations but also the deadline for data propagation. This callback would then be invoked asking the user to allow the connection establishment and the data to be sent/received. (Obviously, by configuration, the user could have previously instructed the system to send/received the data without asking for his authorization.)

Future location aware battery management

Power management can be adjusted taking into account the device's future location. The goal in this case is to ensure that either the device always has enough battery or, if it does not, the location is such that an electric plug is available for recharging. Furthermore, the system will be able to notify the user of the need for charging the battery's device taking into account future tasks and locations.

Similar to the previous case, this requires the system to be aware not only of its physical location but also where it will be in the future and the electrical plugs available in such places. A callback must be registered in order to be invoked when it is forecasted that the battery will be completely drained before getting to the next place where it can be recharged. This callback would then be invoked asking the user to plug his device and recharge its battery accordingly.

Connectivity aware sync'ing

Peer devices can be detected via both absolute location and network broadcast. This may allow the system to synchronize data marked for synchronization without requiring user attention. For example, a user may be in a meeting (with others also holding a PDA or similar device) in which some decision taken should be transmitted to someone not present at that meeting. If that person is then found to be nearby the location to where any of those who attended the meeting will be, the decision taken can be transmitted in an automatic way once the corresponding devices become closer (e.g. by means of Bluetooth).

From a system point of view, this requires setting a callback that will be invoked when some particular device is found to be nearby. In that case, the callback is invoked and asks the user for its confirmation concerning the data transfer (or the system makes the data transfer automatically).

5 Related Work

We witness today, and have for some time now, a trend to integrate several kinds of devices in an increasingly network-centric manner. Research works developed to deal with the issues raised by this shift have been divided in a number of fields: location-awareness, distributed event processing and distributed services integration.

Location awareness is the ability an application has to change its behaviour due to information about geographical positioning, globally or relative to buildings, rooms, etc. and proximity of other devices. In [TSENG01] routing efficiency and quality of service is maximized in ad-hoc wireless networks, i.e., networks composed of dynamically repositioning mobile hosts. In this work, location awareness is used at a lower level than in our work. It is used to improve routing algorithms and packet forwarding. Our approach aims at providing applications broader information about their computing environment, its host and its neighbourhood i.e., information about every capability their execution environment supports, and noticing them when these capabilities are subject to temporary or permanent changes.

In the Xerox ParcTab[WANT95] experience, broad work about location and context-awareness, for proximate selection and automatic reconfiguration, ranging from hardware design, user interface customisation based on context, and location information is also presented. Context information is based on information about neighbouring hosts. A specially developed predicate language is included for context-triggered actions programming. The system is highly dependent on outside information like responses to homing beacons and positioning devices.

Naturally, these notifications about the execution environment, in the case of network centric applications running in mobile hosts, must include information about nearby devices that may be consulted to obtain such information. This must be implemented with some kind of distributed event processing. In [BATES98], [JINI01], [RIO01], [UPNP00] there are several approaches to this issue. [BATES98] defends a framework for a federation of heterogeneous components connected, transparently, by distributed events in a publisher-subscriber model. There is an event taxonomy based on event sub-classing. This can be used to perform event filtering and avoid flooding with undesired events. Event delivery is performed by event-brokers that inject events in listening components by invoking actions. There is an event composition algebra that allows some degree of control over dependency checks between different sets of events and to enforce certain event sequences. Events are logged for future replay or querying.

In Jini[JINI01] the emphasis is put on resource and service discovery. There is only one event class so it lacks expressiveness. This is so as to avoid further class loading during event reception. However, there is a special field in the event class that can be used to hold an arbitrary object and this may be used to effectively ship further, richer information about the event and even supply code for any purpose, e.g., service proxy. This architecture is further refined and extended in Rio[RIO01] with extended event description, capabilities detection for proxy execution, operational strings to represent resources and services and quality of service matching between device capabilities and application requirements.

The Universal Plug and Play [UPNP00] approach aims at very similar goals than the previous two but is also centered on remote device and service detection, and data exchanging without any sort of code download. It is supported in a series of standards that makes it more platform independent though less flexible in the dynamic code download aspect. It supports dynamic IP addressing, device and service discovery; control actions are based on SOAP URL invocations and received events are implemented in XML messages defined in GENA.

Some of these technologies try do address different specific issues. They all try to take advantage of some form of awareness about the computing environment. None of them, though, comprehensively attend to all

the issues presented in this paper. They can, however, be integrated in different EPM's, composing a broad, expressive and uniform environment awareness API provided to environment-aware applications.

6 Conclusions

This paper presents an architecture for an integrated environment awareness system (EAS) that allows applications to assess and adjust to the device, the network and the physical environment where they are executed. The system is capable of anticipating future user behaviour so that applications may advice him in order to take full advantage of the resources available therefore increasing his productivity.

We believe that our system facilitates the development and execution of distributed applications that allow users to take full advantage of the potential that computer mobility brings. This is achieved by providing an infrastructure with a general API that can be easily extended and allows exploitation of significant synergies that can be drawn from combining and reasoning about sets of environment properties.

We also provide some examples such as location aware power management and complex personal organizer operations, and showed how these mechanisms are derived from simple reasoning about elementary environment characteristics.

7 Future Work

We are currently working on the implementation of the EAS detailed in this paper in order to demonstrate its usefulness. We will assess whether the evaluation of queries and callbacks can be efficiently implemented and expect to have some performance numbers in the near future.

We also want the system to be portable to several mobile platforms and we are trying to reuse as much software as possible. As already mentioned, there are several software packages for location/power/etc. management that we intend to reuse as much as possible. However, they have never been used together under a single architecture as the one we propose.

An important issue regarding portability is the mechanism that will be used to associate abstract environment properties with the physical sensors that detect them. Ideally this should be achieved solely based on standard API to access devices but that is not always possible.

Another important implementation aspect is the management of the log of events. Given that this system is aimed at all types of devices, and that memory occupation is a concern in many architectures, it is important to be able to restrict the dimension of the event log either by compressing or truncating it.

There are also additional design issues such defining authorization mechanisms and callback registration permissions for the remote invocation interface, namely deciding how queries and callback for surrounding machines can best be communicated.

8 References

[ACPI99] Advanced Configuration & Power Interface. http://www.acpi.info. February 1999.

[BATES98] John Bates, Jean Bacon, Ken Moody, Mark Spiteri. Using Events for the Scalable Federation of Heterogeneous Components. Proceedings of the 8th ACM SIGOPS European Workshop.pages 58-65. September 1998.

[DON01] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85. Oak Ridge National Laboratory, USA. October 2001.

[GUS95] John Gustafson, Quinn Snell. HINT: A New Way to Measure Computer Performance. Proceedings of the 28th Annual Hawaii International Conference on Systems Sciences. IEEE Computer Society Press, Vol. 2, pages 392-401. 1995.

[JINI01] Jini Specification. Sun Microsystems. 2001.

[KLA00] Alexander Klaiber. The Technology Behind Crusoe Processors. Technical Report. Transmeta Corporation. January 2000.

[MAK98] Spyros Makridakis, Steven Wheelwright, Rob Hyndman. Forecasting: methods and applications, 3rd Edition. John Wiley and Sons, New York. 1998

[RIO01] Rio Architecture Overview. Sun Microsystems. 2001.

[TSENG01] Yu-Chee Tseng, Shin-Lin Wu, Wen-Hwa Liao, Chih-Min Chao. Location Awareness in Ad Hoc Wireless Mobile Networks. IEEE Computer. June 2001.

[UPNP00] Universal Plug and Play Device Architecture. Microsoft Corporation. June 2000.

[WANT95] Roy Want, Bill Schilit, Norman Adams, Rich Gold, Karin Petersen, David Goldberg, John Ellis, Mark Weiser. The ParcTab Ubiquitous Computing Experiment. Technical Report CSL.95-1, Xerox Palo Alto research Center. March 1995.