

Quality Considerations on Ubiquitous Systems

Andres Flores and Alejandra Cechich

Department of Informatics - University of Comahue

Buenos Aires 1400, 8300 Neuquén, Argentina

E-mail: aflores@uncoma.edu.ar, acechich@uncoma.edu.ar

Abstract. When building ubiquitous architectures some generic quality attributes become fundamental. Whilst applying quality aspects in common architectures is quite difficult, for an environment becoming pervasive is even harder. Mobility, Adaptability, Resource Awareness are features that define such an environment. For their support some qualities like Portability, Scalability, Availability must be reconsidered. Since maximize one usually affects the degree of other, design decisions become highly risky in such a construction. This paper proposes a multidimensional scheme where the space of possibilities for quality properties is depicted. In the scheme are identified those areas that highlight the conditions to reach the required ubiquitous features and illustrated by samples of current approaches for better understanding. We expect to use this scheme for future classification of adequate features for pervasive environments on architectural approaches.

Key words : Software Engineering, Software Architecture, Ubiquitous Computing, Software Quality.

1 Introduction

The world of software development and the context in which software is being used are changing in significant ways. One of emerging trends that promises to have a major impact on software development is that of ubiquitous, or pervasive, computing [1]. Pervasive computing is about a future in which computation becomes part of the environment, and many different protocols and operating systems are interrelated to allow accessing information anywhere, anytime in a secure manner [2]. Its comprised computing universe is populated by a rich variety of heterogeneous computing devices: smart cars, entertainment systems, appliances systems, etc. The ambition is to accomplish data to be delivered efficiently, effectively, and economically to any device, no matter the time or distance of user's location. This is not only related to largely distributed systems and applications, but about highly dynamic and mobile sets – clusters of participants, interacting with each other and storing data individually on mobile devices, as well as in remote facilities [3]. The major goal is to achieve an implicit human-computer interaction. That is to render computing devices and their technology invisible so the user remains focused on the matter at hand: receiving updates and tasks. The traditional nature of user's relationship to computation changes in a pervasive environment. The difference is the explicitness of the computational task, where people think in terms of performing explicit tasks “on the computer” – creating documents, sending e-mail, and so on. This model of interaction responds to a low level abstraction: individual applications and devices. When computation is part of the environment this explicitness disappears as individuals behave as they normally do: moving around, using objects, seeing and talking to each other. The computation in the environment is in-charge of facilitating these actions, and individuals may come to expect certain services to achieve high-level goals – a high-level task interaction [2,4].

Developing software for such an environment becomes quite challenging. There are numerous and conflicting requirements to be considered. For example, supporting particular connections with likely reconnections from possible different locations. Also the bridge between broadband content, distributed systems, and mobile systems with limited capacity and bandwidth, which lead us to multiple protocols and devices [3]. It is not only to allow users to move their computational tasks easily from one *environment* to another, but also let users to be able to take full advantage of local capabilities and resources within a given environment, even as other users and devices enter and leave that environment, and as resources (like available bandwidth) change [5,6]. All these requirements must be considered when describing the skills that components in a pervasive environment should possess: Mobility, Adaptability, and Resource Awareness [1]. However, as mentioned before, there is no simplicity for accommodating these important features into such disparate components.

Several researchers have proposed different approaches for ubiquitous computing. Despite the success that have been achieved when focusing on specific low-level requirements, the need for solving the conflict among those necessary features guide us to move our focus to a higher vision [7,8,9]. From an architectural viewpoint the scenery for this conflict may be better considered at analysing other perspectives. Architectural issues are closely related to the

quality, which is accomplished for the various structures comprising a system. There is not easy to reach a good appliance of every required quality and the difficulty varies depending on the attribute. Some of them based its importance on providing great benefits in a development stage, like Modifiability or Reusability. Others highly improve user interaction as we refer to Usability. However there is a group of qualities, which cannot be missed when the architecture involves pervasive computing. Portability, Scalability, Availability are some of those qualities, which become fundamental when building a ubiquitous architecture. Its importance comprises the fact of providing support for those ubiquitous features that every component should include when conforming such architecture. Maximize one of them usually implies injecting to the architecture a negatively side effect, descending the level of other of those essential quality attributes [10,11]. That is why design decisions appear as a subject of considerably matters when developing that kind of architecture.

In this paper we propose a 3-dimensional scheme where the space of possibilities for quality properties is depicted. The importance of each attribute is described when considering a disparate environment. That is, for every generic quality attribute that is considered essential, its relation with every ubiquitous feature is explained. All the conditions that are adequate for achieving a ubiquitous architecture can be identified as scheme areas, once is stated the quality scheme. For a better understanding some scheme areas are illustrated by samples of current approaches where a significant effort to fulfil a degree of quality has been made. The quality scheme provides with a manner to analyse important but conflicting characteristics that should be incorporated in a pervasive computing architecture. This work contemplates the Quality Model proposed by ISO/IEC 9126-1.2 [12]. Though we have slightly change our focus from general software product to specific architectural perspective, which can help analyse skills for such systems under development. That is to provide ingredients to achieve architecture-based analyses that make risks and costs explicit, and allow performing tradeoffs analysis within the space of alternatives [13]. The next section introduces the 3-dimensional scheme where ubiquitous features as well as quality properties are distilled. Then section 3 presents insights from the scheme illustrated with samples of applications of ubiquitous qualities. Finally section 4 presents conclusions and future work.

2 Scheme for Quality on Ubiquitous Architecture

The space for alternatives on architectural qualities upon pervasive computing properties has adopted the form of a 3-dimentional scheme. The first dimension involves main characteristics related to a pervasive environment, called *ubiquitous features*. The second comprises generic *quality attributes* (its name), which should be accommodated in architecture to accomplish the requirements demanded by such disparate environment. The third dimension called *degree* provides a range of values: low, medium, high; that might describe a particular aspect achieved on building a ubiquitous architecture. These values do not state a discrete range instead they express a classification where they may fall, facilitating analysis over architecture. Exact values over this dimension should be obtained through applying adequate metrics. Following are introduced the concepts considered as values for each scheme dimension. First the features for ubiquitous architectures are presented. Then the essential qualities are detailed, highlighting their importance and relation with each other and across dimensions. Dimensions are shown in figure 1.

2.1 Ubiquitous Features Dimension

In order to successfully fulfil the distinctive of pervasive computing, components should exhibit the following characteristics, which were extracted from [1].

- ♦ **Mobility:** refers to both users and their tasks moving across environments. It is required a refocusing of software systems toward collections of components cooperating to achieve a user's task. Besides, users expect the computing environment to take advantage of resources in different environments. Mobility needs to be achieved seamlessly and transparently, with little or no intervention from the user.
- ♦ **Adaptability:** in a ubiquitous environment, it is highly likely that the resources available to a user in an environment will change as resources and users move in and out of that environment. The tasks and software should adapt to take advantage of these new resources.
- ♦ **Resource Awareness:** to effectively achieve mobility and adaptability, a ubiquitous environment needs to make optimal use of available resources to support user tasks. For this, components need not only to publish their interfaces and protocols for interaction, but also make their resource requirements (such as required bandwidth, computing power, memory and battery consumption) known. In addition to the environment being able to ascertain components' resource requirements, components should also be aware of the resources they offer, and adapt their performance accordingly. Components should therefore be able to offer multi-fidelity services.

2.2 Quality Attributes Dimension

Following, the importance of each of the essential quality attributes closely related to the previous features is explained. Table 1 shows a summary of the relation between generic quality attributes and ubiquitous features expressing with a (+) symbol where the relation has been found stronger. Before expressing the relevance of each quality into a pervasive environment, brief but consistent definitions from [10,12] are presented.

- ♦ **Portability:** *the ability of (part of) the system to run under different computing environments. A (part of) system is portable to the extent that all of the assumptions about any particular computing environment are confined to one component (or at worst, a small number of easily changed components).* The main feature supported here is *mobility*. It clearly expresses how crucial become this quality when it has to be fulfilled by many components, as is this case where many components in the presence of many devices come in and out. There is also a relation with the other values on the *ubiquitous features* dimension. Changes in a device usage may imply to run adapting processes after proper device-skill analysis, which is just a system's ability of being aware of available resources.
- ♦ **Scalability:** *the ability of adding components to a system with a minimum of integration concerns. Those components are very low or not coupled with each other, and so this independence provides the opportunity to easily integrate them to the system.* Lots of devices should be easily integrated into a pervasive environment but this may implicate an adjusting process in the meanwhile. Thus, not only *Mobility* is involved but also *Adaptability*, and from above we can see that *Resource Awareness* is the feature that assists in adapting processes.
- ♦ **Usability:** *is highly related to make a system's user interface (UI) clear and easy to use. This mainly implies to get the details of a user's interaction correct: matching the interface to the user's mental model of the system; using conventions, standards and familiar metaphors; providing adequate performance; and allowing a user to feel in control. In addition the right information should be available to the user at the right time and the user's instructions should get to the right destination in the system; that is the information must flow to and across the appropriate components via the connectors.* This quality should be present in every architecture but in a pervasive environment where multiple devices are "in the game" its presence becomes highly recommended. Clearly usability is a matter of *Adaptability* and *Resource Awareness*.
- ♦ **Availability:** *measures the proportion of time the system is up and running. It is measured by the length of time between failures as well as by how quickly the system is able to resume operation in the event of failure. The steady state availability of a system is the proportion of time that the system is functioning correctly.* With no doubt, this is the most precious quality to be achieved in a ubiquitous architecture. As users moves across environments, they will expect to continue accessing to the desire software services. Through managing a dynamic range of computing devices they will want to use the resources to perform computing tasks, such as writing documents, reading email, getting reminders and accessing information. The illusion of "continuity" across environments is important in the presence of long-lasting and highly mobile users. When a mobile user comes into a new environment, he wants to simply resume his tasks and use currently available resources to keep working [4]. This covers the three of the ubiquitous features aforementioned.
- ♦ **Integrability:** *the ability to make the separately developed system components work correctly together. This depends on the external complexity of the components, their interaction mechanisms and protocols, and the degree to which responsibilities have been cleanly partitioned. Integrability also depends upon how well and completely the interfaces to the components have been specified.* This quality is highly related to *Adaptability* and *Resource Awareness* since some adjustment might be required for a component (from the device that offers it) to be integrated to the system.

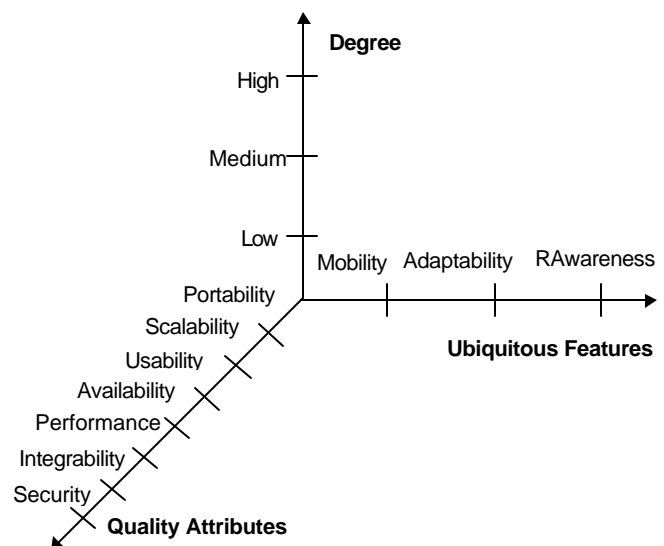


Figure 1: Space for Ubiquitous Architecture Qualities

- ♦ **Security:** is a measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users. If indeed this quality is not immediately recognized, its importance increases when many users are involved in a pervasive environment. Some of them may not be wanted to reach certain aspects of the system. Security is always an important quality that should be maintained in every environment. The three features are involved when the system requires a degree of security.
- ♦ **Performance:** refers to the responsiveness of the system – the time required to respond to stimuli (events) or the number of events processed in some interval of time. Since communication usually takes longer than computation, performance is often a function of how much communication and interaction exist between the components of the system. This is especially true if the components reside on different computing elements, such as in a distributed system. As is explained in [10], performance may be negatively affected by almost every other quality. An example is particularly given concerning portability because of the additional computation involved in execution due to the incorporated isolation to the components. However, portability has been recognized as one of the most essential qualities in a pervasive environment, so there is much consideration to be done in here. The previous definition visibly explains the additional difficulty when in presence of an environment where the components are distributed across disparate devices. Performance mainly concerns Resource Awareness, involving capabilities related to analysing best places where locate functionality.

Ubiquitous Features	Quality Attributes						
	Portability	Scalability	Usability	Availability	Integrability	Security	Performance
Mobility	+	+	+	+			
Adaptability		+	+	+	+	+	+
Resource Awareness		+	+	+	+	+	+

Table 1: Relation between Quality attributes and Ubiquitous features

3 Insights from the Quality Scheme

Despite facilities and capabilities to access and create valuable information into this unlimited infrastructure that freely and transparently allows communication between people, there is still a need for discerning what good architectures for pervasive computing are. This trend has a number of consequences. It forces us to consider architectures for open systems that scale to the size and variability of the Internet. Systems in which resource usage is a critical issue with the need to handle reconfiguration dynamically while guaranteeing uninterrupted processing of parts that do not change. When mobility is actually applied there is a negative influence injected in the architecture. While isolating components to be platform-independent, communication overhead is produced. Moreover, the level of mobility might be in conflict with concurrency and level of resource conflict. In essence, *the higher the mobility, the lower the concurrency*. To sum up, from an architectural perspective, mobile computing arises the problem of finding ways to reconfigure high-level tasks to match local resources, which requires architectures that can dynamically self-adjust to a changing computing base. However the need for supporting persistent storage and mobile computations, allowing devices to work even under limited connectivity, indicates that a pervasive computing architecture ought to be kept simple to facilitate its deployment across multiple devices [3,13].

Currently we are inadequately prepared to design and analyse systems whose overall utility is based on such tradeoffs. Some effort has been done in this matter with a focus on other but architectural quality aspects [14,15,16]. A key ingredient to improve the situation would be to have architecture-based analyses that make risks and costs explicit, and thus allowing to carry out tradeoffs analysis within the space of alternatives. That is, to characterize “approximations to correctness” and to tune utility functions over these values. Our approach is intent towards being able to discriminate whether the construction of a ubiquitous architecture achieves its required quality features. This work contemplates the Quality Model proposed by ISO/IEC 9126-1.2 [12]. However our focus is over a specific architectural perspective rather than a general software product viewpoint in order to help analyse skills for such systems under development. To provide a better understanding of the usage of the proposed schema in architecture-based analysis, next section gives explanations of some feasible circumstances. Notice that along next section, two levels could be distinguished when relating dimensions. Some combinations may initially appear regarding only system level while others concerning its comprised components. However for a system level, consider that some or all of its internal components should exhibit the discussed requirements. Thus insights are mainly related to components. Table 2 summarizes main concerns emerged from the quality scheme. By means of real samples some situations are properly illustrated.

3.1 Using the Ubiquitous Quality Scheme

In order to present the explanations of the combined dimension across the quality scheme we will move through the *quality attributes* dimension describing the alternatives for every *ubiquitous features* dimension values.

Portability is the first issue, which concerns provide independence to the system.

- ♦ **Mobility.** Forms to include a virtual machine (VM) layer to allow communication with numerous and disparate devices must be considered. This means, to give the faculty of being platform-independent to components, which will be located into portable devices or will communicate with them. However as a portability layer is defined, its services should be appropriately specified to get flexibility in the system. For example, for *Jini* is proposed to equip each device with a Java Virtual Machine (JVM) [17,18]. In addition, as many disparate appliances enter in scene, the system must be capable of communication through different protocols. *Personal Digital Assistants* (PDA), *PalmTops* (integrated with cellular phones) and *Mobile Phones* may use the *Wireless Application Protocol* (WAP). For some tasks HTTP can be used for accessing to the WWW. Besides FTP can be used for transporting big-sized files. In [18] *W@PNotes* uses an integrated web-browser for mobile phones upon WAP (WAP-phones). Figure 2 shows the optimal situation about these two aspects. For a pervasive environment the degree in this relation should be at least around a *medium* value. That is the set of protocols should not be too reduced. Set members must provide a sufficiently wide-range usage, e.g. FTP may be discarded for being of usage-specific. Systems with poor or no independence but still providing mobility may be found in a low degree.
- ♦ **Adaptability.** There is a need to adjust components to be located in portable devices that may involve an interface update. Further, there may be adaptation of the Human-Computer Interaction (HCI) to be used. It means adjust components for communication according to the transport protocol the device uses. Another subject must also be considered. *A portability layer only provides true portability if usage conventions are enforced: application software must be constrained to use the portability layer and not directly access the environment that it is charged with abstracting* [10]. This is clearly a policy concern. Every device-integrated application or component migrated to a device must respect the communication policy. Every component is involved in a contract for service in order to enforce the system integrity too (see *security*). However this contract may change according to device movements likely involving updates on the amount or quality of services (*renegotiation*) [16]. This tuple might be relaxed to be fairly above the *low* degree. Contract renegotiation can be avoided for a fixed policy.
- ♦ **Resource awareness.** As can be seen for portability is necessary the ability to make analysis for different surging protocols and appliances capabilities. Thus the best channel for communication can be selected. Besides, there should be monitoring processes to maintain contractual policies. As the last consideration, this tuple should be above the *low* degree so as to assure analyses of protocol and device communication capabilities and run the appropriate adaptation processes.

Scalability may also be architecturally analysed even for components that were decided to be located in a fixed place like a server. However this aspect is not of interest for composing this scheme.

- ♦ **Mobility.** For its achievement is essential not to design context-specific functionality. The system interacts with many disparate devices that need to be easily integrated. However most of them may not be powerful enough and functions need to be accommodated to do their required work. Functionality design should prepare the system for dynamic changes. Generalization is a proper technique but also to prepare strategies for dynamic changes. Through this the system also allows users not to feel discontinuity when a certain function is not available due low-skill devices. Client components may then be designed for using generic services that may be properly adjusted as needed. Design patterns may help making the system more flexible, appealing to a broader customer base. *BEACH* follows this approach defining the system functionality in various levels of abstraction [21,23]. Once an analysis of the current situation is obtained, functions are appropriately adjusted. This tuple has a big importance and should not be depreciated, taking as value upper a *medium* degree.
- ♦ **Adaptability.** It comes to fulfil those processes of adjusting functions for adequate execution on different appliances. That is, selecting specialized functions among the collection that has been designed. But also there may be adaptations of component interface in the meanwhile, when a mismatching is discovered. See *BEACH* in [21] for an example of adapting processes. As before this combination cannot be reduced in degree below the medium value whether the system wants to achieve a proper level of integrability.
- ♦ **Resource awareness.** Its association to scalability has been explained in previous issues, and its importance relies on providing the necessary protocol-device analysis. From this, adapting operations can be run after proper selection. Thus, this relation should be kept as higher as possible not descending from a *medium* value.

Usability mainly concerns UI and in a pervasive environment, users interact with disparate devices whose interfaces should be standard-way designed.

- ♦ **Mobility.** The system must be prepared for interacting with many different forms of HCI according to current tasks and portable devices utilized. In [21], *BEACH* offers UIs that fits also to the needs of devices that have no mouse or keyboard which require new forms of human-computer and team-computer interaction, and through its layered model an interface may be adapted from its conceptual definition to the physical usage in a particular real device. Further a system should allow the usage of different applications for a communication. Thus a web-browser may be adequate while in other cases simply e-mail communication can be enough (or be forced to its usage). For word processors as example, there are many options, from those complex as *MS-Word* to those simpler as *MS-NotePad* or *Vi for X*. The choice among these applications not only depends on device capacity but also on the underlying Display Service, as we have *X*, *X-Windows*, *MS-Windows*, *MS-WinCE*. This combination requests to be higher and an important effort on issues explained for *scalability* should be done. When starting a ubiquitous project, this relation can be relaxed for providing particular alternatives of applications and HCI forms, which means a value around a *low* level. However for a full operational system this combination must be enhanced.
- ♦ **Adaptability.** The system must allow function operation according to user tasks and device skills. In addition, when quality services differ from contract specification (see policy on portability) proper adjustments must be applied. For example synchronization must be carried out when audio and video are involved. Also video quality may be altered along with going away from the signal server. Audio and Video quality can also be affected by other air-frequencies inserting “noise” in the signal. Furthermore, HCI customisation must be properly achieved, which means not only selection a form of UI, but also respecting standards for uniform HCIs. Besides the way a task is carried out depends on UI. There can be many alternatives that influence on making a task complex or simpler. This involves the way functionality is joined or services are composed for a user task. Here we are concerned with high-level user tasks, what surely involves many simpler and common low-level tasks [4]. Usability is the most related to this, since it is user-oriented and so may properly understand user-requirements. That way users feel not affected in their tasks but greatly benefited instead. In [18] the service composition concept is applied. For a system to be considered ubiquitous this combination should not be less than *medium* value.
- ♦ **Resource awareness.** To select an adequate HCI is required to apply device-skill analysis to discover platform, display server, and communication-integrated channel. Further there must be monitoring procedures to maintain quality service (as mentioned previously). Since usability concerns “user-doing” not only analysis for proper service composition is needed, but also its coordination among devices where functionality is located. Also the system should prevent the need for future usage of a certain function, that is, to predict future system behaviour. In [7] is presented *Spectra* for learning functions from applications and predicts operation resource usage. Thus, the more an operation is executed, the more accurately its resource usage is predicted. In [24] a notification server optimised for distribution of awareness information is provided. Users are enabled to subscribe a kind of information and specify awareness tools to be sent to the information (customisation). For this tuple the consideration should be as the preceding issue.

Availability must be enhanced in a pervasive computing environment. It clearly expresses the essence of such an environment, the necessity to provide for users the feeling of being “followed” by tasks across environment changes that brings the idea of system continuity.

- ♦ **Mobility.** There is a need of an appropriate underlying infrastructure to enhance the capacity of being “ubiquitous”. That is, no matter where the user moves, it is always possible to feel the presence of the system. Since a user moves from one environment to another many connection/disconnection situations occur. However users expect to resume their tasks without losing or discontinuity effects. For this, powerful distributed networks must be supplied. In order to construct the whole infrastructure fixed wired networks (*Ethernet LAN*, *Modem via dial-up telephone*), wireless networks (*Wireless LAN*, *DECT*, *CDPD*, *GSM*), Infra-Red (inside a room) can be used along with many connectivity devices. Other powerful but costly possibilities are satellite systems [15]. As for any construction, design decisions are also based upon economic considerations, which may in many cases disarrange an initial and well-configured project. If there is no possibility to achieve this combination with a higher degree, at least a wireless network must be provided to allow mobility into the system.
- ♦ **Adaptability.** Since wireless networks are less reliable and involve limited bandwidth, traditional transaction models, transaction-processing techniques, concurrency-control mechanisms must be reconsidered. For example concurrency is typically based on locking, which does not suit for highly failure prone. However another approach called *Optimistic Two Phase Locking* [25] has been proposed. Further advanced algorithms to conserve energy are

critical components as well as those in charge to adjust variations in communication channels, which may involve re-routing processes. Figure 2 shows a possibility for this combination: a *medium* degree that represents a limit situation considering how the system may be affected.

♦ **Resource awareness.** Since transaction might involve a division of their computation into sets of operations to be executed on mobile or stationary hosts proper alternative examination must be allowed. Different mechanisms are required if users want to continue their transactions while move across environments [26]. Monitoring tasks for channel communication as well as appliances capabilities must be carried out. In [7] the *Spectra* component includes several and specialized monitors like cache state monitor, battery monitor, and others. The last monitor takes advantage of “smart” batteries: chips that report battery levels and power drain. The collected information is used to make alternative elections according to user metrics. For a user moving away from a mobile server constant location-awareness should be applied, using for example a *GPS* [15]. Thus the election for a suitable server might be analysed while executing right adjusting processes. A consideration here is whether the user wants to be informed about server-changes [25], what deforms the transparency of feeling a complete system support. With the system architecture *one.world* in [19] is proposed to let the components to know how their operating environment is changing so that they can best adapt their computation, communication and UI. That is, instead of developing transparent solutions, they find more likely that decisions will need to be application specific. As the previous issue, this tuple assumes an importance that makes the whole system comes to a critical situation when its level is not kept higher. The sensation of “continuity” while allowing mobility is what makes a pervasive environment so admired.

Performance is highly affected by portability and also influenced by scalability concerns as will be seen in following issues.

♦ **Mobility.** As was explained in section 2.2 there is a compromised relation here. We have two opposite considerations. First, distributing functionality among disparate components with the aim of taking advantage of better execution upon powerful devices. Parallelise computation highly improve performance through mobility. Nevertheless when those servers do not provide good enough resources, communication among these computation units becomes a large problem. Since there is a need for *portability*, devices must include a VM. However this additional component makes devices having less space for requested applications and computation must be divided between the application and the VM [3]. Thus performance is negatively affected and the solution might be prioritising functionality. That is, selecting among applications, those that may provide the required service with a likely less complex functionality. The worst case would imply making the user doing more simple tasks to achieve a complex one. In future with less costly and more powerful devices the last consideration may become obsolete and it will be solved the current *portability* & *performance* conflict. However now it must be carefully studied since totally influences the efficiency of the system.

♦ **Adaptability.** Those issues explained for scalability influence its considerations. From the preceding concerns prioritise functions may imply adapting the component interface or adjust its functionality where specialize is a recommended technique. Additionally, there must be adjustments for the channel communication either for re-routing or minimize the amount of transmissions or avoid coalitions. All this influence decisions while dealing with locating functionality on portable devices. In [8] there is a basis of architectural models. Here a translator interprets *repair scripts* as low-level operators, which include routines for creating new request queues, activating and deactivating servers, and moving client communications to a new queue. Operations at the *Model Layer* are translated into calls on the operations in the *Runtime Layer* to effect the actual change in the system.

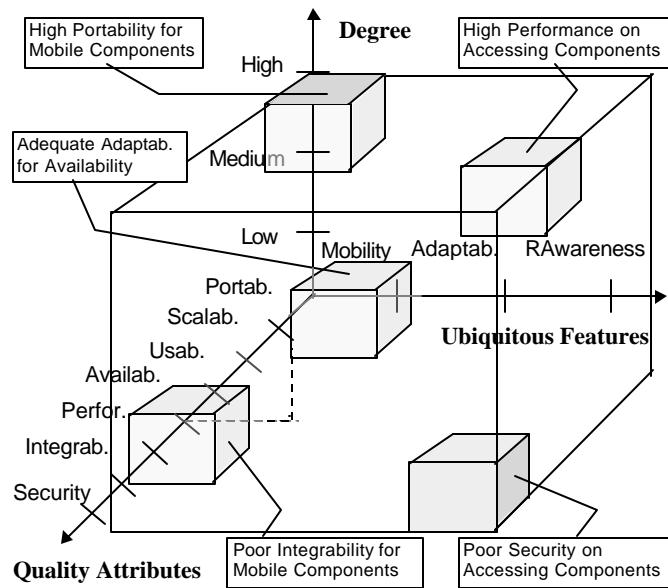


Figure 2: Quality Alternatives on Ubiquitous Architecture

♦ **Resource awareness.** The need is for adequate components whose main behaviour is to find best alternatives for easy system-appliances communication. As we have seen the current negative influence of portability requires to evaluate options before migrate components to portable devices for computation. Thus location of functionality is a significant concern and proper techniques must be applied to achieve this evaluation. In [7] *Spectra* evaluates alternatives by their impact on user metrics, which measure performance or quality perceptible to the end user. Its calculations take to an instance of the well-known concept of “resource goodness mapping”. In [9] the architectural models permit analytical methods to derive sound repair strategies, e.g. a queuing theoretic analysis of performance can indicate possible points of adaptation for a performance-driven application. Figure 2 shows the optimal situation for performance when related to resource awareness.

Integrability is closely related with scalability, and in a ubiquitous architecture is highly influenced by portability. This implies that components must be designed with appropriate interfaces to easy their integration to the pool of components, which is dynamically restructured while devices come to scene as environment changes.

♦ **Mobility.** As was explained for portability-adaptability, portable components must only interact with the environment through specified portability-layer services. Even though there can be many different communication protocols that components need to be easily integrated. This implicates a flexible underlying infrastructure accordingly developed with integration facilities. Examples are found in [18,17] (*Jini*), [20] (*Bluetooth*), [28] (*UPnP*). Thus the level of this tuple ought to be maintained higher than a *low* degree. Figure 2 depicts a situation that must be avoided, a poor grade of integrability in relation with mobility.

♦ **Adaptability.** Many mechanisms should be incorporated to the system to allow integration. Though components interact through specified service contract, adjustments in component interface might seem necessary due variations on device skills or changes of appliances. In [21] (*BEACH*) there is a mechanism called *physical interaction* where adaptations made by users (setting of devices) trigger actions on the software. This is useful for changes of current collaboration mode. Thus for a flexible infrastructure this combination becomes important. Its value should not descend the *low* degree.

♦ **Resource awareness.** The system must be incorporated with functionality to allow recognising appliances skills when they start to be operable into the environment. This is important to facilitate its full integration. *BEACH* in [21] achieves the integration of devices and software within a working environment or a working context by means of *context awareness*. It uses sensors (distributed all over the environment) to collect context information. So, changes can be detected to inform the application (through a likely pre-processing). Thus, the previous consideration for the degree of this relation is applicable here.

Security for architectures also embraces integrity and privacy. Since many portable devices may come into the range of a wireless network, there should be a mechanism for recognizing those who are enable from those who are not – *integrity*. Moreover among the empowered users of a pervasive system, there can be those who want to maintain their information confidential from others – *privacy*.

♦ **Mobility.** Many devices might access to the environment. It is needed a mode to identify users as well as portable devices. Some models for user-identification involves *encryption keys*, *cryptographic signatures*, *encrypted messages* [15,27]. Also some devices allow identification like cell phones, PDAs, PalmTops since they have a telephone number. Existing security infrastructures deal with authentication and access control, which may involve security risks. In [27] there is a support for *strong cryptographic security and authentication*. A suggestion to improve security in [8,22] is basically by the addition of *trust* similarly to the way security is handled in human societies. A common trust model assumes servers as physically secure and trusted, and client-server communication channels encrypted for privacy. In [29] there is a strong security infrastructure comprising various mechanisms: *location authentication*, *beacon and tag authentication*, *secure WebTunnel*.

♦ **Adaptability.** The need is centred in mechanisms to adjust security aspects. It can involve techniques to update the confidentiality level or adjust the frequency in the communication channel. The latter becomes important into a wireless network that may allow unauthorized intruders. In [6] tasks are treated at higher level and its description includes constraints that also involves privacy requirements. When these constraints are not met the *Prism* component may suspend the executing task, or at least adjust the parts that are affected by the context change; for instance hiding the display of sensitive data when someone else comes into the user’s office. In [29] the *reverse proxy* handles URLs containing capabilities-opaque strings to obtain a record about the client’s ability to access users’ services. That access record may be set up to restrict use of the URL according to local policies. For example, it may restrict the number of uses or the time span of use. It may allow a user to access certain resources even after it has gone home but preventing accesses after a specific time, such as the expiration of a work-for-hire contract.

♦ **Resource awareness.** Mechanisms for allowing secure access and monitoring functions to maintain integrity and privacy in the system are required. Traditional access preventing are authentication, firewalls, proxies. Some of these approaches have been redesigned for a pervasive environment. In [27] is the Auto-session application that employs SSH key-based authentication to securely validate the user's access to a desktop machine without the need for typed passwords. In [8] is explained that a user is allowed to access a service or information, if the user has the access right to do so, or if the user has been *delegated* the ability by a trusted authority. Here *trust management* is viewed as developing of security policies, the assignment of *credentials* to entities, checking if the credentials fulfil the policy and the delegation of trust to third parties. In addition, in [22] is proposed an approach for the *Aura* architecture [6] where is suggested using the caching technique for achieving trust. For *caching trust* is used a surrogate of the desired data into the mobile device. The challenge is to preserve the system's safety properties even when the surrogate is physically compromised, what can be achieved either using private or public key encryption. Figure 2 presents an alternative that should be avoided or at least considered for future architectural extension, which is a poor grade of security-resource awareness, where both integrity and privacy are at risk as we have seen.

Quality Attributes	Ubiquitous Features		
	Mobility	Adaptability	Resource Awareness
Portability	Ways to isolate components; Proper portability-layer services (JVM); Many protocols (WAP, FTP, HTTP).	Adjust interface component, HCI; Contract renegotiation.	Recognize protocol communication; Analyse device communication capabilities; Hold contract policies.
Scalability	Generalize and strategize functionality for uniform, transparent and low-coupled components interaction.	Adjust component interface; Specialize functionality;	Analyse protocol communication, device skills.
Usability	Support for many forms of HCI; Many communicative applications (e-mail, fax, web-browser).	Adjust function operation, video/audio quality and synchronization; Customize HCI; Service composition.	Recognize protocol, device abilities; Analyse proper HCI; Monitor device-skill variation; Predict likely required functions; Analyse "user-tasks" for proper composition and coordination.
Availability	Infrastructure support: distributed network (wireless, wired), sensors. Mechanisms to sustain "continuity".	Adjust portable device and channel communication capabilities, operational functions; Transaction processing techniques.	Monitor portable and communication devices; Find and analyse alternatives for computation; Location-awareness (GPS).
Integrability	Many communication protocols; Easy connectivity (Jini, UPnP).	Adjust communication protocol, component interface.	Recognize protocol, interface component, device skills
Security	Ways of identification: encryption keys, signatures; infrastructure support	Adjust confidentiality, communication frequency; hide information;	Authentication of users, portable devices; Maintain user, appliances list.
Performance	Parallelise computation upon powerful devices; Prioritise functions for users (weak devices).	Adjust channel communication, functionality (interface, specialize)	Find best alternatives for easy system-appliances communication; Proper functionality location.

Table 2: Issues from the Quality Scheme

4 Conclusions

In this paper we have proposed a space for alternatives on architectural qualities and ubiquitous computing properties that have adopted the form of a 3-dimensional scheme. The first dimension involves the main characteristics related to a pervasive environment, called *ubiquitous features*. The second comprises generic *quality attributes* (its name), which should be accommodated in architecture to accomplish the requirements demanded by such disparate environment. The third dimension called *degree* provides a range of non-discrete values, describing a particular aspect achieved on building a ubiquitous architecture. Through applying adequate metrics exact values over this dimension might be obtained. For every dimension the importance of each property was explained when considering a disparate environment. In addition, every combination of generic quality attributes and ubiquitous features was analysed. Thus once the quality scheme is stated, all the conditions that are found adequate for achieving a ubiquitous architecture can be identified as scheme areas. For a better understanding some scheme areas were illustrated by samples of current approaches where a large effort for a grade of quality has been achieved.

There is a need for improving the way conflicting requirements – as those found in a pervasive environment – can be analysed and coherently accommodated. Some effort has been done in this matter with a focus on other but architectural quality aspects. A key ingredient for improving the situation will be to have architecture-based analyses

that may allow to carry out tradeoffs analysis within the space of alternatives which help adjust utility functions while characterizing “approximations to correctness”. Our approach is intent towards providing with a mode for analysing important but conflicting characteristics when building a ubiquitous architecture. We believe that an architectural viewpoint is an appropriate perspective for more opportunities of being able to discriminate whether the construction of a ubiquitous system can reach its desired requirements. We expect to use this scheme for future classification of pervasive environments’ adequate features on current and future developed architectural approaches.

References

1. Garlan D. *Component-Based Software Engineering in Pervasive Computing Environments*. The 4th ICSE, Wksp on Component-Based Soft. Eng.: Component Certification and Sys. Prediction, Toronto, Canada, May 2001.
2. Mark W. *Turning Pervasive Computing Into Mediated Spaces*. IBM Systems Journal-Pervasive Computing Issue, Vol. 38, No. 4, <http://www.research.ibm.com/journal/sj38-4.html>. 1999.
3. Vichr R., Malhotra V. *Build it to move: The Architecture of Pervasive Computing*. IBM developerWorks: Wireless articles. <http://www-106.ibm.com/developerworks/wireless/library/wi-archpvc/index.html>. May 2002.
4. Wang Z., Garlan D. *Task-Driven Computing*. Carnegie Mellon University, School of Computer Science, Tech. Rep. CMU-CS-00-154. <http://reports-archive.adm.cs.cmu.edu/anon/2000/abstracts/00-154.html>. May 2000.
5. Weiser M. *The Computer for the 21st Century*. Scientific American, 265(3): 94-104, September 1991.
6. Sousa J., Garlan D. *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments*. The 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, 2002.
7. Flinn J., Narayanan D., Satyanarayanan M. *Self-Tuned Remote Execution for Pervasive Computing*. The 8th Workshop on Hot Topics in Operating Systems, Oberbayen, Germany, May 2001.
8. Kagal L., et al. *Moving from Security to Distr. Trust in Ubiquitous Computing Envs*. IEEE Computer, Dec. 2001.
9. Garlan D., et al. *Software Architecture-based Adaptation for Pervasive Systems*. In Proc. of ARCS’02, 2002.
10. Bass L., Clements P., Kazman R. *Software Architecture in Practice*. Addison-Wesley, 1998.
11. Shaw M., Garlan D. *Software Architectures: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
12. ISO/IEC FCD 9126-1.2: 1998, Information Technology – Software Product Quality – Part 1: Quality Model.
13. Garlan D. *Pervasive Computing and the Future of CSCW Systems*. Proc. of CSCW’00, Philadelphia, Dec. 2000.
14. Castro P., Chen A., Kremenek T., Muntz R. *Evaluating Distributed Query Processing Systems for Ubiquitous Computing*. Department of Computer Science, University of California, Los Angeles.
15. Chalmers D., Sloman M. *A Survey of QoS in Mobile Computing Environments*. IEEE Communications Surveys, <http://www.comsoc.org/pubs/surveys>, 2nd Quarter 1999.
16. Chalmers D., Sloman M. *QoS and Context Awareness for Mobile Computing*. Computing Department of Computing, Imperial College of Science, Technology & Medicine, London, U.K.
17. Jini Connection Technology, available at <http://www.sun.com/jini>
18. Zeidler A. *User Interface Design for Ubiquitous Computing: W@PNotes, anExample*. Databases and Distributed Systems Research Group Wilhelminenstr, Darmstadt.
19. Grimm R., et al. *System-Level Programming Abstractions for Ubiquitous Computing*. University of Washington.
20. Blue tooth website. *The Official Bluetooth Website*, at <http://www.bluetooth.com/>, 2001.
21. Tandler P. *Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices*. In UbiComp’01. Heidelberg Springer LNCS 2201, pp.96-115, 2001.
22. Satyanarayanan M. *Caching Trust Rather Than Content*. Operating Systems Review, Vol.34, No.4, Oct. 2000.
23. Tandler P. *Modeling Groupware Supporting Synchronous Collaboration with Heterogeneous Single- and Multi-User Devices*. In CRIWG’01, IEEE CS Press, Darmstadt, Sep. 6–8, 2001.
24. Kantor M., Redmiles D. *Creating an Infrastructure for Ubiquitous Awareness*. Inf.&Comp.Sc., Univ. California.
25. Seydim A. *An Overview of Transaction Models in Mobile Environments*. Department of Computer Science and Engineering, Southern Methodist University, Dallas.
26. Madria S. *Transaction Models for Mobile Computing*. Centre for Advanced Information Systems, School of Applied Science, Nanyang Technological University, Singapore.
27. DeVaul R., Pentland A. *The Ektara Architecture: The Right Framework for Context-Aware Wearable and Ubiquitous Computing Applications*. The 4th Int’l Symp. on Wearable Computer IEEE Computer Society, 2000.
28. Universal Plug & Play, <http://www.upnp.org>
29. Debaty P., Caswell D. *Uniform Web Presence Architecture for People*. Internet & Mobile Systems Lab.