Optimizing the Utility Function-Based Self-adaptive Behavior of Context-Aware Systems Using User Feedback

Konstantinos Kakousis, Nearchos Paspallis, and George A. Papadopoulos

Department of Computer Science, University of Cyprus
P.O. Box 20537, 1678 Nicosia, Cyprus
{kakousis,nearchos,george}@cs.ucy.ac.cy

Abstract. The vision of ubiquitous computing is about numerous devices embedded in our every-day environment, designed to serve humans in a non-obtrusive manner while minimizing the required user attention. These devices are expected to seamlessly monitor context changes and adapt their behavior and functionality to maximize the user benefit. However, designing the self-adaptive logic of such systems is far from trivial. This paper discusses a utility function-based approach for specifying the adaptive behavior of component-based context-aware systems. Although this approach allows for completely autonomous self-adaptive behavior, it also leverages potential user feedback by adjusting and optimizing its behavior. It is argued that this approach provides significant improvement to the adaptive behavior of a system while maintaining the required user attention to a minimum. At the same time, it keeps the complexity involved in the development of such context-aware, self-adaptive applications to a reasonably low level.

Keywords: Context-aware, Self-adaptation, Utility functions, User feedback.

1 Introduction

With the proliferation of smart-phones and the increasing importance of ubiquitous computing, it is difficult to overestimate the potential of context-aware, self adaptive systems. In mobile computing settings, handheld devices are expected to be aware of their ever changing context and adapt their behavior accordingly. Furthermore, in ubiquitous computing environments many devices are embedded in the environment and are expected to provide autonomic behavior by utilizing their knowledge on the context to adapt their functioning. For all these actions, the main driving and guiding force is the optimization of the user experience. In other words, the context is sensed and the adaptations are decided with the purpose of improving the provided utility as it is perceived by the user in the mobile or ubiquitous computing environment.

However, building systems which can be configured to anticipate and react on the user needs and wishes is far from trivial. This paper advocates the use of a multi-dimensional, utility function-based approach for enabling automatic decision-making in the event of context changes. In this way, several aspects of the decision-making

R. Meersman and Z. Tari (Eds.): OTM 2008, Part I, LNCS 5331, pp. 657–674, 2008.

process are examined independently, and then combined to reach a decision. We argue that the way these aspects are combined can be automatically adjusted by means of asking the user to confirm or reject an adaptation decision and then use the feedback to tune the way the decisions are made.

In this respect, we propose an approach which attempts to take into consideration as many choice-affecting aspects as possible. These aspects form a multidimensional space, and the choice is automatically made based on the overall matching across these dimensions. It is argued that this approach can offer a reasonable approximation of the user's reasoning process, while at the same time requiring only a reasonable amount of work from the developers. The utility of each aspect contributes to the overall utility of the system by a factor which is dynamically adjusted based on the received user feedback. We show that this adjustment can automatically optimize the context-aware, self-adaptive behavior of the system. Finally, an underlying middleware system such as MADAM [1] or MUSIC [2] is assumed, in which the developed applications are component-based and are dynamically planned and composed at runtime.

The current paper builds upon our previous work on a multi-dimensional adaptation model originally presented in [3]. That work introduced a method for interfacing humans with context-aware, self-adaptive applications. This paper extends and refines the results of its successor but, mainly, it also proposes a novel, control theory-based approach for automatically optimizing the utility functions used to provide the self-adaptive behavior of the system. This mechanism incorporates user feedback into the adaptation reasoning loop and uses that input to automatically optimize the multi-dimensional utility functions which are used to enable its self-adaptive behavior.

The rest of this paper is structured as follows: Section 2 discusses the foundations regarding context awareness and self-adaptive behavior in mobile and pervasive computing systems. The same section also presents the basic, multi-dimensional reasoning model. The main ideas of the proposed optimization are then introduced in section 3, which is followed by a case-study based evaluation of the approach in section 4. The evaluation of the proposed optimization is performed by means of a simulated experiment presented in the same section. This optimization approach is compared with related work in section 5 and, finally, section 6 concludes the paper by summarizing its main contributions and by pointing to our plans for future work.

2 Foundations

Consider a user in a mobile or pervasive computing environment. Such environments are generally designed to offer services to the users. Such services involve both direct and indirect user interaction. In both cases it is assumed that the user perceives the service and has an impression about its *utility, i.e.* different users might perceive the utility of the same service differently. In this discussion, the utility refers to a quality metric, broader than *Quality of Service* (QoS), which aims to capture the general user satisfaction with the functioning of a system in a given context. For instance, if a user prefers one system configuration over another one, then it is assumed that the former has a *higher utility*. A more formal definition of the *utility* is provided in section 2.3.

In mobile and ubiquitous computing environments, the context changes frequently. For this reason, systems targeting this type of environments are designed so that they can be operated in a set of varying configurations and modes (referred to as *variants* in this paper). These variants are designed with the goal of optimizing the utility for at least a subset of the context space (evidently, when a variant provides no advantages for any of the context, then it is unnecessary). In the scope of context-aware, self-adaptive systems the main goal is to provide mechanisms which dynamically and automatically find and apply the optimal configuration as the context changes. In this case, we assume that the optimality is computed by means of the user-perceived utility, which must be monitored and evaluated.

In order to enable a more rigorous study of the problem, we present a few definitions for the *context*, the *variants* and the *utility*. These definitions were originally introduced in [4] and provide the foundation for the proposed optimization.

2.1 Context

In this paper we use Dey's definition for context, which is one of the most frequently cited [5]: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves*".

In practice, context can be divided into several cross-cutting types, or dimensions. These dimensions can be enumerated (e.g., gender can only be "male" or "female") or unbound (e.g., time)".In this perspective, the context might be modeled as a multidimensional space in which each relevant context type defines a dimension. Similar approaches exist in the literature, such as the one presented by Padovitz *et al* [6]. Analogous to our approach, this work treats context as a cross-application artifact which needs to be handled at a lower (than the application) level, such as in middleware. The framework is based on the idea, introduced by Zaslavsky [6], which suggests representing a context as an object in a multidimensional Euclidean space, termed as *situation subspace*. Each situation subspace is a collection of *N* attribute-values, each one of them restricted in an acceptable region.

Contextual information is distinguished between *numerical* and *non-numerical* attribute values. Additionally, numerical values can be further categorized as *required* and *optional, or to values* with greater or lesser influence over the context feasibility.

For the manipulation of non-numerical attributes two approaches are found in the literature. The first one suggests mapping semantic values into numeric values and treating all kind of contextual information uniformly. The second approach adopts the relational database model for specifying the values of the attributes and their domains. To avoid loss of expressiveness, in our description we adopted the first approach. Therefore, we assume that each context type can be abstracted by a real number, *i.e.* $\Re$, and a context space of d types can be abstracted as a d-dimensional space $\Re^d$.

Thus, at any time *t*, the context can be abstracted by a point c_t, which defines a value for each of the *d* dimensions, *i.e.* the c_t is defined as $(c_{t1}, c_{t2}, \ldots, c_{td})$, where c_{ti} indicates the context value at dimension *i* for the time instance *t*. We refer to these points c_t as context instances. Furthermore, an acceptable region of values for each dimension restricts the contextual space that a context instance apprehends. Assuming

that the context space is comprised by the infinite points in $\Re^d$, the above are more formally expressed by the definitions (1) and (2) shown below:

Context instance

$c_t \equiv (c_t^1, c_t^2, \ldots, c_t^d) \in \Re^d$, where $c_t^i \in \Re \; \forall$ i in [1..d]

restricted by $A_t \equiv (a_t^1, a_t^2, \ldots, a_t^d) \in \Re^d$, where a_t^i a set of elements that satisfy a *condition* (*e.g.,* $0<a_t^i<1$). (1)

Context space

context space $\equiv \Re^d$, where d represents the number of dimensions of the context space. (2)

2.2 Variants

In the literature, there are mainly two approaches for software adaptation: *parameter-based* and *compositional-based*. Furthermore, several aspects of adaptation have been extensively studied, such as *where*, *when*, and *how* they are applied [7]. In the context of mobile and ubiquitous computing environments, adaptivity is required to overcome the variability of these environments. For this reason, systems are designed with adaptive properties so that a system can be configured in different ways, *i.e.* different component compositions or parameter settings, resulting to different variants. Each one of these variants is designed to offer maximum utility for specific context conditions. The main characteristic of alternative variants is that they maintain the *functional properties* of the software system, while possibly varying its extra-functional characteristics. In this case, the purpose of the context-aware, self-adaptive system can be seen as the adjustment of its extra-functional properties with the aim of optimizing the user-perceived utility [8].

In order to simplify the analysis of such systems, it is important to assume that (if needed) the adaptation domain can be transformed to a finite set of configurations by quantizing their value range, *i.e.* by mapping ranges of the infinite domain to a finite number of subsets.

In some cases, these variants are defined *a priori* by the software developers. However, in order to provide maximum flexibility and to meet the requirements of such dynamic environments, variants are often required to be composed dynamically. For instance, in component-based systems the variants are constructed by examining the provided and required services, *i.e.* interfaces, of each component [1]. The exact set of available variants can thus fluctuate based on the availability of components and services and also based on the contextual conditions.

Variant

A variant is any parameter-based or compositional-based configuration of the application, maintaining its original functional properties (3)

Variants

$variants \equiv \{variant_1, variant_2, \ldots, variant_N\}$ (4)

In this paper we are primarily concerned with component-based applications and thus, it can be assumed that the system is comprised of either a single application or a set of applications. However, we consider the utility of each possible application individually, partly based on the user preferences for each one of them, as it will be discussed in the next section. The definition of variants is summarized in (3) and (4). It is worth noting however, that variants can vary dynamically over time, depending on the availability of components and services, and also based on the status of the required resources.

2.3 Utility Functions

In this work, utility functions refer to mathematical artifacts mapping the combination of every *context* state and *variant* to a scalar value, typically in the range [0, 1], where 0 indicates minimum, *i.e.* worst, utility, and 1 indicates maximum, *i.e.* best, utility. This approach is quite similar to the notion used in micro-economics where utilities represent user happiness. The choice of the [0, 1] bounds provides the convenience of allowing the multiplication of different utilities without exceeding the original bounds and without having to normalize the result.

The purpose of a utility function is to provide a formal, mathematical method for computing the *utility* of an application, as it is perceived by the end-user. In this respect, the utility function is defined as follows:

Perceived utility function

A perceived utility function ($U_{perceived}$) is a function that for any context point C_t and any two variants V_x and V_y, it computes arithmetic values (e.g., in the range of [0, 1]) so that $f(c_t, v_x) \geq f(c_t, v_y)$ if and only if the user prefers variant V_x to V_y from her point of perception (5)

Given this definition, the problem of decision in self-adaptive context-aware systems becomes the formation of a computed utility function ($U_{computed}$) which can *approximate* the perceived utility. This approximation is the topic of the following subsection.

2.4 Multi-dimensional, Utility-Based Model

Most modern mobile phones provide personalization, and manual adaptation, through profiles, which are user-customizable. For example, a user can configure the "default" profile of his smart-phone with a custom ring-tone and also by setting the vibration on and off. This example shows a scenario where the adaptation affects multiple dimensions. For instance, one such dimension is whether there will be a specific ring-tone played when the phone receives a call or not, and another dimension is whether the vibration will be activated or not.

In this paper, we extend this model, to arbitrary numbers and types of dimensions. We refer to these as *adaptation dimensions*, and we argue that they can provide the foundation for specifying context-aware, self-adaptive applications, as it will be showcased later on. It is however, an essential prerequisite that the adaptation dimensions are high level adaptation aspects, as independent as possible from the contextual

situation. In other words, the preference of a user in favor of an adaptation dimension should not change radically between *nearby* context conditions.

To enable this kind of adaptation reasoning, the utility of each application is computed independently for each dimension, and the overall utility is computed as their weighted sum. Regardless of whether the subject under discussion is an application or an individual component, its utility over a specific dimension can be more easily computed in terms of a *fitness function*. Such functions measure the fitness of particular variants for specific context conditions. For example, considering the dimension of the mobile phone sound alert, the fitness function would examine if the variant into consideration (*e.g.,* sound off) is a good *fit* for a given context (*e.g.,* in a meeting). Fitness functions are essentially utility functions covering only a specific aspect of the adaptation.

In practice, it is not easy to define a perfect utility function, simply because users are not generally completely aware of how they perceive the optimality of a service, nor can they describe it. For instance, it is possible for a user to sense that she or he prefers one variant over another under certain context conditions, without explicitly knowing why, or even which contextual factors affect their opinion. Furthermore, it is possible that the user's perceived utility depends on factors that cannot be explicitly measured, such as their emotional state.

In this regard, we propose the formalization of utility functions which try to approximate the functioning of the users' internal reasoning process. In practice, users evaluate the utility of a service over numerous aspects. This can be expressed by an equation as shown in (6), where the fitness function over dimension i is expressed as U^i. However, in order to implement a realistic adaptation reasoning algorithm which imitates the user's perception, we define the *computed utility* which is an approximation of the *perceived utility* as shown in equation (7), and which is computed over a subset of the dimensions of the *perceived utility*. For example, a user perceives the overall utility offered by a video-conference system as a combination of many factors, but that is simulated by examining his perception over *video clarity* and *latency* only, as defined by equation (7). It is argued that this approach results in a computed utility which approximates the user perceived one, as defined by equation (8). Furthermore, it is argued that this approach provides a reasonable and realistic method for enabling context-aware, self-adaptive behavior.

Finally, it is worth noting that this elementary approach enables adaptation reasoning over multiple dimensions, but it is limited in terms of customization. Most notably, it is expected that different users have different perception for the importance of each of the examined dimensions, compared to other users. For this reason, the overall utility of an application is refined to express the *weighted sum* of the dimensional utilities, as shown by equation (9).

As the weights w_i are expected to be dependent on the particular user, *i.e.* because different users assign different levels of importance on the individual aspects, they can be adjusted to reflect the preferences of the targeted user. One of the main contributions of this paper is that we demonstrate an automated method for adjusting these weights, taking into consideration the user feedback at runtime.

$$U_{perceived}(variant_X) \equiv \sum_i U^i{}_{perceived}(variant_X) \tag{6}$$

$$U_{computed}(variant_X) \equiv \sum_i^K U^i{}_{perceived}(variant_X) \tag{7}$$

$$U_{computed}(\cdot) \cong U_{perceived}(\cdot) \tag{8}$$

$$Utility\ (V_j, C_m) \equiv \frac{\sum_{i=1}^{K} (w_i \cdot fitness_{\ i}(V_j, C_m))}{\sum_{i=1}^{K} w_i} \tag{9}$$

Given this mathematical method for computing utilities, a context-aware, self-adaptive system can be constructed by means of evaluating the *computed utility* of each variant whenever the context changes, and by adapting to the optimal variant as needed. This approach is further described and evaluated in the following sections.

3 Leveraging User Feedback to Adjust the Utility Functions

Following up on the promise of pervasive computing, we envision a mechanism that allows the users of context-aware, self-adaptive systems to intervene in the adaptation control loop when they wish to, with the goal of optimizing its self-adaptive behavior. As the utility function is essentially the weighted sum of the dimensional utilities of certain aspects of the adaptation, an optimization can be achieved by means of adjusting these weights. In detail, it is argued that different users have different preferences over the individual aspects measured by the fitness functions. Thus the effectiveness of the utility function can be optimized by adjusting the corresponding weights according to the individual user needs.

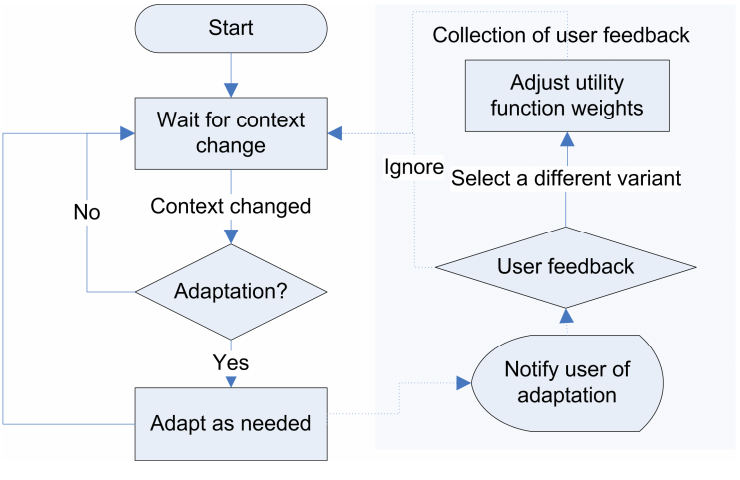

Fig. 1. Adaptation control loop and collection of user feedback

In this section we present a solution which collects user feedback while it causes minor distraction, and uses that input to adjust the utility function weights and, consequently, optimize its self-adaptive behavior. A high level view of this approach is illustrated in the workflow of Fig. 1.

Throughout this paper, it is assumed that the middleware provides automatic management of context sensing, and asynchronously informs the adaptation reasoning engine whenever a relevant context change occurs (for an example see [9]). As soon as such a context change is detected, the adaptation reasoner computes the utility of all its variants under the new context and decides if an adaptation is required by checking if the current variant is still the one offering the highest utility.

Initially, the system sets the weights of the utility functions to a default value (*e.g.,* the median value which is 0.5 for the used range of [0, 1]). The variant with the highest utility score is considered to be the best fit for the given context and is selected by the system for implementing the upcoming adaptation. This is illustrated in Fig. 1 (left side), where context changes trigger the adaptation reasoning loop, the result of which might be an adaptation or not.

At the same time though, the users have the ability to intervene and express their disagreement with the adaptation decision. We envision this to be implemented by means of a non-intrusive notification appearing in the status bar of the used device (*e.g.,* smart-phone or PDA) when an adaptation is decided, allowing the users to get involved if they wish to, but not requiring them to do so. In the case where a user mediates and selects a different variant, an event is generated and forwarded to the adaptation reasoner which uses this input to adjust the utility functions (in a way that will be explained later on). Eventually, the control loop resumes its normal flow and continues monitoring for new context changes. This is illustrated by the right (shaded) side of Fig. 1. Please note that the user is free to completely ignore the notification, which should expire and disappear after some time.

Mathematically, if we assume that the new context is $Context_i$ and the system selects $Variant_s$ as the optimal one, then it is implied that the selected variant has a higher (or at least equal) utility, compared to any other variant (see inequality (10)). Furthermore, if the user expresses his or her disagreement regarding the system's choice and selects an alternative variant, *e.g.,* $Variant_u$, their choice is translated into the inequality shown in (11), which combined with equation (9) produces the inequality depicted in (12). D is the dimensionality of the utility function and w_d is the weight of the corresponding dimension d.

$$\text{Utility}(Variant_s, Context_i) \geq \text{Utility}(Variant_x, Context_i),\ \forall x \in V,\ x \neq s \tag{10}$$

$$\text{Utility}(Variant_u, Context_i) > \text{Utility}(Variant_s, Context_i) \tag{11}$$

$$\sum_{d=1}^{D} (w_d \cdot (fitness_d(V_S) - fitness_d(V_U))) < 0 \tag{12}$$

Thus, in principle, the user's feedback regarding a particular adaptation decision can be transformed into an inequality constraint. Extending the same reasoning for all the decisions of the system, it can be inferred that each time a $Variant_u$ is selected among N variants for the context conditions $Context_i$, $N-1$ constraints can be defined.

Given this, it is argued that the problem of incorporating the user's preferences into the adaptation reasoning process can be transformed to a *constrained nonlinear*

optimization problem [10]. In this kind of problems, the input is a set of equalities and inequalities formed over a set of unknown real variables. The goal is then to assign values to the set of unknown variables so that the equalities and the inequalities are satisfied while at the same time a cost function is maximized or minimized (in this case the cost functions are assumed to be quadratic).

The transformation is achieved as follows: For each input received by the user concerning an adaptation decision, we form an inequality as shown in (12). In this case, the difference of dimensional utility of variants s and u is a known, constant value and the weights w_d are unknown, real variables (which we restrict to [0, 1]). This inequality provides an additional constraint which in combination with the other collected constraints define the conditions that the weights must satisfy in order for the utility functions to produce the desired adaptations when the context changes.

Thus, the problem of optimizing the utility functions to better match the user preferences essentially becomes the computation of a new set of weights w_i, so that the utility function (shown in equality 9) is adjusted accordingly. As the original setting of weights is considered to be *almost* optimal, we define a cost (or objective) function which enables the optimization of the weight variables with the minimum variation of the values in each optimization step.

In a d-dimensional space, where d is the number of adaptation dimensions, consider w and w' as two vectors representing two different d-dimensional points of the space. Minimizing the *Euclidean distance* between vectors w and w' provides us with the ability to fulfill our goal. The vector w defines the current weights for the adaptation dimensions, while w' represents the set of the unknown weight variables which we intend to compute. To achieve the minimum fluctuation possible, the objective function is formulated as the Euclidian distance between the two multi-dimensional points w and w', as shown by formula (13).

Since the fitness of a particular variant over a specific dimension d is constant for a given context state, the difference $fitness_d(V_S)$-$fitness_d(V_U)$ is also constant, and for simplicity it is denoted as $C_{d,S\prec U}$ (where $S\prec U$ implies that the user prefers $Variant_U$ to $Variant_S$ for the given context). Thus, the mathematical formulation for the optimization problem is transformed to the one given in (14).

$$\left\|w - w'\right\|_2 \equiv \sqrt[2]{\sum_{d=1}^{D}((w_d - w'_d)^2)} \tag{13}$$

$$\text{minimize } \left\|w - w'\right\|_2 \text{ so that } \sum_{d=1}^{D}(w'_d \cdot C_{d,S\prec U}) < 0 \;\forall \textit{ constraint } S\prec U \tag{14}$$

Mathematical optimization problems, such as the one specified above can be solved by the use of *sequential quadratic Programming* methods. In particular we have used the *fmincon* function of Matlab's optimization toolbox which computes a constrained minimum of a scalar function by starting at an initial approximation and updating an estimation of the *Hessian of the Lagrangian* at each iteration [11].

As with any optimization problem, there is no guarantee that a feasible solution exists for the given constraint set. In this case, a failure to generate weight values that satisfy the inequalities of the constraints set might be due to a contradiction between the user's preferences in favor of a dimension in different context scenes. For this kind of situations, the system fulfills the user's wish for the given input, *i.e.* applies the selected variant, without however changing the weights.

If however, the optimization process succeeds, a new adaptation event is generated and forwarded to the adaptation controller which initiates a new run of the adaptation control loop using the modified weights.

4 Case Study-Based Evaluation of the Approach

To illustrate the functionality of our approach and also to evaluate its efficiency in real-life examples, we demonstrate the approach through a case study example. This case study specifies a scenario describing a typical day of an on-site technician who uses a smart-phone to assist her with her everyday routine. Her smart-phone is assumed to run an intelligent agenda manager application, exhibiting context-aware, self-adaptive behavior. These adaptive features of the application include switching between offline and online synchronization modes, switching between user interaction modes and adjusting the volume level based on the ambient noise.

4.1 Experimental Approach and Setup

In order to experimentally evaluate the proposed approach, we describe a scenario which defines an adaptive application and a set of pre-defined context conditions. Furthermore, we define fitness functions to capture the utility of each variant over the individual dimensions, for any given set of context values. The utility functions are used to compute the utility of each variant throughout the scenario, and thus to select the best matching variant. Later on, we incorporate hypothetical user feedback, and we demonstrate how it is used to adjust the weights used in the utility function and how it results to automatic accommodation of the user preferences.

Since our approach is based on the *multi-dimensional model* presented in section 2.4, we start by specifying the relevant adaptation dimensions which the adaptation reasoning mechanism is based on. For the scope of this scenario, we consider three adaptation dimensions: *Interaction Mode*, *Audio Volume*, and *Synchronization Mode*. The first one examines the user's need for hands-free operation, *i.e.* when she is driving, and the system's ability to provide her this mode. The audio volume dimension controls the audio volume, for which the user's need may vary based, for example, on the ambient noise. Finally, the synchronization mode dimension reflects the user's need for synchronizing the agenda with new tasks and events and for downloading software updates, which has an impact on the battery and network use.

The scenario is comprised of seven scenes, describing different phases in the day of the technician. In the first scene, the user is at home preparing for the day and uses the agenda manager to update her of the day's tasks. New events are downloaded through the home Wi-Fi network. During the second scene, the user drives towards her office and the device switches to using the GSM network for updating the agenda. At the same time the hands free mode is enabled and the agenda's contents are converted to speech and then spoken through the car speakers. To reach her office, the engineer drives through a tunnel where GSM connectivity is broken. The manager adapts accordingly by switching to offline mode, thus deferring the synchronization of the agenda events. Once the user arrives at office, in the fourth scene, the agenda utilizes the available high speed WiFi network for downloading events and software

updates if needed. Later on, during a local technical meeting, the device automatically enters the silent mode while continuing the task updates through the slower WiFi network available in the meeting room. Then in the sixth scene, the user visits a client site where manual work is needed. At that point, the ambient noise is very high and the device's volume is adjusted accordingly. Furthermore, the hands free operation is enabled freeing the engineer's hands for manual work. Finally the engineer returns to the office and, as the battery is running low, the smart-phone switches to a mode for low consumption performing necessary synchronization only when needed.

To allow the experimental evaluation of the variants' utility for each of these scenes, we collect the relevant context information, *i.e.* types and values, in Table 1.

Table 1. Context values for the seven scenes of the scenario. The acceptable range for all the monitored context types, except the user.state, is [0, 1] with intermediate step 0.25.

Scene	user.state	env.noise	res.net_bw	res.net_cost	res.batt
1	home	0.25	0.75	0.25	1.0
2	driving	0.5	0.25	0.75	1.0
3	driving	0.5	0.0	0.0	0.75
4	office	0.25	1.0	0.0	0.75
5	meeting	0.0	0.75	0.0	0.5
6	manual_work	1.0	0.75	0.0	0.5
7	office	0.25	1.0	0.0	0.25

For the purposes of this scenario we consider five main context types: the user's state, the environment noise level, the battery level of the device, the network availability and the network cost.

As it was discussed earlier, we are concerned with component-based applications and compositional adaptation and thus, it is assumed that the middleware constructs the set of all the possible variants dynamically, taking as input the set of available components. The resulting variants specify special compositions of the application, as well as values of their configurable parameters. To avoid cluttering, we do not discuss the detailed composition of the variants but rather, we just describe them and present their properties.

In more detail, the first variant specifies that the agenda is configured to use the maximum bandwidth available for updates and for synchronization. This mode results in high battery consumption. Additionally, the hands-free operation is disabled. In the second variant, the agenda continues to use high bandwidth for updates and synchronization, but the hands-free mode is enabled. Then, in the third variant, the agenda uses GSM for synchronization, while maintaining the hands-free mode enabled and the volume adjusted for high ambient noise. Next, in the fourth variant, the agenda manager operates in the offline mode with the hands-free mode enabled and the volume adjusted for the car speakers. Further on, in the fifth variant the agenda manager operates in a low battery consumption mode featuring minimum synchronization and not providing any hands-free support. Following that is variant six in which the agenda operates in the minimum synchronization mode again, but configured for noisy environments. The hands-free operation mode is enabled. Finally, in the last variant the agenda uses high speed bandwidth for synchronization and provides no hands free operation. In this case, the battery consumption is high.

Table 2. The relevant properties of the application variants

Variant #	net_sup	vol_supp	hfree_supp	batt_cons	GSM_req
1	1.0	0.3	0.0	0.75	false
2	0.75	0.3	0.75	0.75	false
3	0.25	0.4	1.0	0.5	true
4	0.0	0.5	1.0	0.25	false
5	0.25	0.3	0.0	0.0	false
6	0.25	0.8	0.75	0.5	false
7	0.75	0.0	0.0	0.75	false

In order to be able to evaluate the dimensional fitness of the variants dynamically, each one of these variants explicitly defines a set of properties. In this scenario, we have identified 5 relevant properties: the *network support*, the *volume support,* the *hands free support*, the *battery consumption* and the *GSM required* property. Note that in the case where the variants are dynamically formulated as component compositions, these properties can be dynamically computed by means of combining the individual component properties. Based on the description of the variants, we have assigned them the properties illustrated in Table 2.

Table 3. Fitness functions for the three adaptation dimensions

```
Utility_interaction-mode =
    1 - |diff(userNeededHandsfree, variant.hfree_supp)|
```

```
Utility_audio-volume =
    1 - |diff(variant.vol_supp, env.noise)|
```

```
Utility_synchronization-mode =
    if (battery_level not critical) then
        0.6*(1 - |diff(variant.net_supp, res.net_bw)|)
        + 0.2*(1 - |diff(variant.batt_cons, res.batt)|)
        + 0.2*(1 - res.net_cost)
    else
        0.2*(1 - |diff(variant.net_supp, res.net_bw)|)
        + 0.7*(1 - |diff(variant.batt_cons, res.batt)|)
        + 0.1*(1 - res.net_cost)
```

Finally, given the context and the variants domains, we need to define the dimensional utility functions, *i.e.* the fitness functions, in order to be able to complete the experimental evaluation of the approach. These functions are responsible for expressing the *fitness* of each variant for a given set of context conditions over each one of the adaptation dimensions detected at the beginning of Section 4.1. A fundamental assumption of this approach is that the individual fitness functions are *correct, i.e.* given any context conditions, they provide a valid ranking of the studied variants for the corresponding dimension). The fitness functions are expressed in a mixed form of mathematical notations and pseudo code, as shown in Table 3.

In this case, the *diff* function measures the difference between the provided and the required degree for the particular feature. In practice, this method measures the

suitability of the respective variant for a particular aspect as a function of the context. In the first fitness function, the *userNeededHandsfree* corresponds to a value denoting the importance of the hands-free mode according to the user state.

$$U(V_j, C_m) = \frac{w_1 \cdot f_1(V_j, C_m) + w_2 \cdot f_2(V_j, C_m) + w_3 \cdot f_3(V_j, C_m)}{w_1 + w_2 + w_3} \quad (15)$$

Finally, as described in Section 3, the middleware is responsible for autonomously selecting the most suitable variant based on the monitored context. This sort of decision is made by means of a feedback control loop, as shown in Fig. 1. The utility of any variant for a given context state is computed by applying the equation of formula (9) to the three adaptation dimensions identified earlier in Section 4.1. In this case, the utility of variant V_j under context conditions C_m is computed using the above equation (15).

4.2 Experimental Results

Given this experimental setup, the evaluation of the dimensional utilities of the seven variants, for some specific assignments of context values, is straightforward. The overall utility is computed by estimating the weighted sum of the dimensional utilities, as shown in formula (15).

We aim to include user's preferences in the adaptation reasoning mechanism by optimizing the weight values of the adaptation aspects as explained in Section 3. In order to reveal the potential of the presented solution the same seven-scene scenario is repeated twice. We consider the first trial as a training session for the system during which the user has the opportunity to express her disagreement over the system's adaptation decisions. These disagreements are internally translated into mathematical constraints which are then used in the optimization phase. Eventually, the user feedback is reflected in the utility function through the adjusted weight values. When the second trial is executed, the adjusted weights obtained during the previous day's run, *i.e.* training phase, are used in the adaptation reasoning and thus the system decisions are expected to reflect the user's preferences.

During the first context scene, while the user is at home preparing for the day's activities, the adaptation reasoner picks a variant that uses maximum bandwidth for updates and synchronization. Since battery consumption is not an issue and the hands-free operation is not needed, variant 1 is found to be the best choice. However, the user expresses her disagreement as she would rather have lower utilization of the available bandwidth and simple task synchronization without the overhead of the software updates. She selects variant 7 as she thinks that it is a better choice for her.

The system satisfies the user's wish and at the same time the optimization method attempts to modify the weight values appropriately. Given the constraint that variant 7 should get a higher utility score than variant 1 in the first context scene, the optimization component adjusts the weights w_1, w_2 and w_3 to 0.5, 0.4435 and 0.5459 respectively. The progress of these three weights, which correspond to the *interaction mode*, *audio volume* and *synchronization mode* respectively, is depicted in Table 4.

Table 4. Illustrating the automatic selection of the optimal variant, and the adjustment of the weights as a result of the user feedback

Scene #	Weights			Day 1	
	w_1	w_2	w_3	System selection	User feedback
1	0.5	0.5	0.5	variant 1	variant 7
2	0.5	0.44	0.55	variant 3	
3	0.5	0.44	0.55	variant 4	
4	0.5	0.44	0.55	variant 1	
5	0.5	0.44	0.55	variant 7	
6	0.5	0.44	0.55	variant 6	variant 2
7	0.5	0.3	0.6	variant 5	
Scene #	Weights			Day 2	
	w_1	w_2	w_3	System selection	User feedback
1	0.5	0.3	0.6	variant 7	
2	0.5	0.3	0.6	variant 3	
3	0.5	0.3	0.6	variant 4	
4	0.5	0.3	0.6	variant 1	
5	0.5	0.3	0.6	variant 7	
6	0.5	0.3	0.6	variant 2	
7	0.5	0.3	0.6	variant 5	

Later on in scene 2, the user takes her car and drives to the office. During that time, a GPRS network is available and the system uses it for synchronization. At the same time the hands free operation is enabled and the volume is adjusted for the car environment. Variant 3 is the system's choice for the second context scene, which is approved by the user. In this case, the user accepts the system's decision implicitly, *i.e.* by not intervening in the adaptation loop. The same applies in the third scene, where both the system and the user accept variant 4 as the optimal selection.

In the next two scenes, the user does not intervene to the system's selection, which is implicitly interpreted as agreement. While at office, the user enjoys the full utilization of the available bandwidth which accelerates the synchronization and update of the device (variant 1), while during the local technical meeting the silent mode and the high speed synchronization of variant 7 is agreed to be the best choice.

Conversely, during her visit to the client's site, the technician disagrees with the system's choice. Variant 6 was originally selected due to its great support for noisy environments and due to its hands-free provision. However, the user is proved to be mainly interested in maintaining the high speed synchronization rather than adjusting the volume level perfectly. Thus, variant 2 is selected instead of variant 6. The same optimization process is followed and the weights of the *interaction mode*, the *audio volume* and the *synchronization mode* are adjusted to 0.5, 0.3 and 0.6 respectively. Finally the user agrees with the system's decision in selecting the low battery consumption mode for the last context scene (variant 5), since the device battery level is remarkably decreased.

The same seven-scene scenario is repeated a second time, *i.e.* day 2, but this time the optimized weight values computed during the first iteration, *i.e.* day 1, are used. It turns out that this time, the user agrees with the system's decisions completely. This is reasonable as the weights for the adaptation dimensions have been adjusted according to her preferences already. The user feedback, along with the evolution of the weight values are depicted in Table 4.

5 Related Work

Kokar *et al* [12] propose a paradigm of software engineering which maps the concepts of control theory to software engineering of self controlling systems. In this adaptive software model a QoS subsystem is introduced and is assumed to generate feedback information. In some cases the feedback is the result of a function which generates output variables from internal values, while in some other cases external input is taken (*e.g.,* from the user) and feedback information is generated based on all the available relevant information. Although control theory-based models can produce impressive results in software adaptation and reconfiguration, they inherit some main constraints of control theory. For example in dynamic environments the controller might not be able to steer the systems towards the desired goal (controllability problem) or small changes in the input might cause large modifications to the system's behavior (stability problem). In addition to that, using traditional formalisms of control theory, such as differential equations, for controlling software systems is not an easy task. In an attempt to overcome this complexity, intelligent controllers which use soft computing techniques, *i.e.* fuzzy logic, neural-networks, genetic algorithms and machine learning, have been proposed in the literature but have been applied mainly to other disciplines than software engineering, such as artificial vision, target identification, etc [13].

Ranganathan and Campbell in [14], describe various techniques for the construction of context-aware agents in ubiquitous computing as well as numerous approaches for defining their behavior in varying conditions. Among other methods (mainly rule-based), machine learning approaches [15] are presented enabling agents to train according to the environmental conditions or the users' mood. Although machine learning techniques can monitor the users' preferences and give valuable feedback to the system, they require training which in some cases can be quite long in period. This is against one of the main goals of ubiquitous computing: non-obtrusive user feedback collection. Additionally, machine learning techniques on smart-phones and other hand-held devices could be too resource demanding to be accommodated, thus causing unacceptable slowdowns.

Finally, machine learning techniques are also used by O' Connor *et al* in [16] but this time an application optimization is attempted through the dynamic definition of self-adapting context. In particular, an application can use feedback from the environment to evaluate its contexts and adapt them where necessary in order to eliminate issues arising with the developer-dependent context definitions (*e.g.*, inaccurate context data that lead to incorrect behavior, or fixed relationships between developer-defined contexts and sensor data). The Q-Learning algorithm from the reinforcement learning area is used to learn the optimal action for each context state. Though a powerful approach suitable for releasing developers from the complicated task of defining the monitored context types, it is application dependent and a new learning process needs to be processed each time a new application is installed. On the contrary, we argue that our approach is user centric and optimizes the adaptation behavior of the system according to the user's preferences over high-level dimensions uniformly for all the applications.

Concerning software adaptation, the current state of the art refers to three main approaches, namely *action-based*, *goal-based* and *utility-function based* [17]. In this

paper we are concerned with utility function-based approaches, which assign values (utilities) to adaptation alternatives and which provide higher levels of abstraction by enabling dynamic determination of the optimal adaptation alternative (variant), typically the one with the highest utility.

The use of utility functions for enabling context-aware, self-adaptive systems is a rather novel approach, receiving increasing interest from the software engineering community. For example, the MADAM project proposes a middleware which uses a utility-based, architectural approach to adaptation [9]. In this case, the utility functions are also expressed as functions on context, using intermediate *property predictor* artifacts. The latter are used to compute reusable parts of the utility function. Furthermore, a similar multi-dimensional utility approach is also described in [18], but which is however limited to four QoS-specific dimensions.

Unlike the state of the art, our approach breaks the computation of the utility for a variant into several aspects, covering different cross-cutting dimensions of the adaptation. For instance, the MADAM approach [9] uses a static approach where the context-aware properties of applications are fixed into the composition plans, making the reuse of individual components significantly harder. Contrary to this, our approach does not depend on any hard-coded properties in the plans, but rather it dynamically acquires the relevant properties of each variant at deployment time by accessing its relevant metadata. This has the significant advantage of facilitating reusability. In addition to this, one of the main contributions of the presented approach, when compared to similar adaptation methodologies [17, 9 and 18], is the incorporation of the user's feedback in the adaptation reasoning mechanism, at runtime rather than statically in the form of predefined user preferences.

As it is stated in [19], interfacing with humans is one of the main challenges in designing and implementing autonomic computing systems. The results of this paper target primarily developers of context-aware, self adaptive systems. The proposed model can also be of significant help as it adopts the Separation of Concerns (SoC) approach and facilitates reusability of the context-aware and adaptation properties of the components (and applications), which is a significant gain for the developers.

6 Conclusions and Future Work

Mobile and pervasive computing introduces new and important challenges to the software developers. Especially with respect to the interaction with users, context-aware applications are expected to automatically and autonomously adapt to maximize the overall user satisfaction. In this respect, we have refined a multi-dimensional adaptation reasoning approach originally introduced in [3], and also we have introduced an optimization technique which allows the automatic adjustment of the self-adaptive behavior of context-aware systems. This optimization is based on the aforementioned utility function-based approach, where the weight of each dimension of the adaptation reasoning is adjusted automatically, based on user feedback.

Defining the context-aware, self-adaptive behavior of a system, while allowing it to adjust and improve its functioning, is still an open research problem. The initial evaluation of this work was based on a theoretical case study example which has illustrated the potential of this approach. Future research is also planned in order to

evaluate this approach in the context of more complex and realistic scenarios, using actual devices and real user feedback. Furthermore, we are investigating the possibility of applying additional techniques, such as reinforcement learning from the field of artificial intelligence, with the purpose of allowing a context-aware, self-adaptive system to optimize their adaptation logic even after deployment.

Acknowledgments. This work received partial financial support by the EU as part of the IST-MUSIC project (6th Framework Programme, contract no. 35166).

References

1. Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., Hallsteinsen, S., Horn, G., Khan, M.U., Mamelli, A., Papadopoulos, G.A., Paspallis, N., Reichle, R., Stav, E.: A Comprehensive Solution for Application-Level Adaptation. Software – Practice and Experience Journal (to appear, 2008)
2. Self-adapting applications for Mobile Users in Ubiquitous Computing environments (MUSIC) (accessed on Wednesday) (August 20, 2008), http://www.ist-music.eu/
3. Paspallis, N., Kakousis, K., Papadopoulos, G.A.: A Multi-dimensional Model Enabling Autonomic Reasoning for Context-aware Pervasive Applications. In: The Workshop for Human Control of Ubiquitous Systems (HUCUBIS 2008) in conjunction with the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2008), Trinity College Dublin, Ireland. ACM Press, New York (to appear, 2008) (accepted for publication)
4. Dey, A.K.: Understanding and Using Context. Personal Ubiquitous Computing 5(1), 4–7 (2001)
5. Padovitz, A., Loke, S.W., Zaslavsky, A.: Towards a theory of context spaces. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004, March 14-17, pp. 38–42 (2004)
6. Zaslavsky, A.: Adaptability and Interfaces: Key to Efficient Pervasive Computing. In: NSF Workshop series on Context-Aware Mobile Database Management, Brown University, Providence, January 24-25 (2002)
7. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing Adaptive Software. IEEE Computer 37(7), 56–64 (2004)
8. Paspallis, N., Papadopoulos, G.A.: An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns. In: 30th International Computer Software and Applications Conference (COMPSAC 2006), Chicago, USA, vol. 1, pp. 299–306. IEEE Computer Society Press, Los Alamitos (2006)
9. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using Architecture Models for Runtime Adaptability. IEEE Software 23(2), 62–70 (2006)
10. Bertsekas, D.P.: Nonlinear Programming, 2nd edn. Athena Scientific (1999) ISBN 1-886529-00-0
11. Chong, E.K.P., Zak, S.H.: An Introduction to Optimization, 2nd edn. John Wiley & Sons Pvt. Ltd, Chichester (August 2001)
12. Kokar, M.M., Baclawski, K., Eracar, Y.A.: Control Theory-based Foundations of Self-Controlling Software. IEEE Intelligent Systems and Their Applications 14(3), 37–45 (1999)

13. Aksit, M., Choukair, Z.: Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision. In: 23rd International Conference on Distributed Computing Systems Workshops, pp. 84–89 (2003)
14. Ranganathan, A., Campbell, R.H.: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, p. 998. Springer, Heidelberg (2003)
15. Alpaydın, E.: Introduction to Machine Learning (Adaptive Computation and Machine Learning). MIT Press, Cambridge (2004) ISBN 0262012111
16. O'Connor, N., Cunningham, R., Cahill, V.: Self-Adapting Context Definition. In: First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, July 9-11, pp. 336–339 (2007)
17. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility functions in Autonomic Systems. In: International Conference on Autonomic Computing (ICAC), New York, NY, USA, May 17-18, pp. 70–77. IEEE Computer Society Press, Los Alamitos (2004)
18. Alia, M., Eide, V.S.W., Paspallis, N., Eliassen, F., Hallsteinsen, S., Papadopoulos, G.A.: A Utility-based Adaptivity Model for Mobile Applications. In: 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW 2007), Niagara Falls, Ontario, Canada, May 21-23, pp. 556–563. IEEE Computer Society Press, Los Alamitos (2007)
19. Kephart, J.O.: Research Challenges of Autonomic Computing. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 15–22. Springer, Heidelberg (2006)