A Multi-dimensional Model Enabling Autonomic Reasoning for Context-aware Pervasive Applications

Nearchos Paspallis
Department of Computer Science, University of Cyprus
20537-1678 Nicosia, Cyprus
+357-2289-2672
nearchos@cs.ucy.ac.cy

Konstantinos Kakousis
Department of Computer Science, University of Cyprus
20537-1678 Nicosia, Cyprus
+357-2289-2684
kakousis@cs.ucy.ac.cy

George A. Papadopoulos
Department of Computer Science, University of Cyprus
20537-1678 Nicosia, Cyprus
+357-2289-2693
george@cs.ucy.ac.cy

ABSTRACT
A fundamental requirement for autonomic computing is to be able to automatically infer how human users react in similar contextual conditions. This paper examines the problem of autonomic reasoning for adapting context-aware applications in mobile and pervasive computing environments. In this type of systems, both the context and the adaptation possibilities must be modeled appropriately to enable the adaptation reasoning engine to infer decisions on which adaptations to perform. It is assumed that multiple cross-cutting concerns affect such decisions, and thus we introduce a multi-dimensional, utility-based model which attempts to simulate the user's reasoning mechanisms. The proposed model is applied to component-based mobile and pervasive applications, and is being evaluated through a detailed scenario. It is argued that the proposed model provides a novel and promising approach for designing context-aware, self-adaptive systems, in particular with respect to mapping the adaptive behavior to the system.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]: Object-oriented design methods

General Terms
Algorithms, Design, Human Factors

Keywords
Context-aware, Self-adaptive, Utility functions, Modeling

1. INTRODUCTION
With the advent of mobile computing and the increasing importance of ubiquitous computing, one can easily realize the potential of context-aware, self adaptive systems. Such systems are commonly expected to provide autonomic behavior, utilizing their knowledge on the context to adapt their functioning. For all these, 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 service 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 not trivial. The human reasoning is complex and it has not been sufficiently understood yet. Furthermore, different users exhibit different behavior and consequently, different choices. Even if users were interviewed, many would not be able to detail their decision process in the form of an algorithm. Many users are not even explicitly aware of the factors which affect their decision, when faced with a choice.

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. Finally, it is assumed that the developed applications are component-based and are dynamically composed at runtime.

The rest of this paper is structured as follows: Section 2 provides the required foundations regarding context awareness and self-adaptive behavior in mobile and pervasive computing systems. Then, the basic multi-dimensional reasoning model is presented in Section 3. A case-study scenario is introduced in Section 4 to evaluate the proposed multi-dimensional utility-based approach. This 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 and they involve both direct and indirect user interaction. In both cases it is assumed that the user perceives the service and has a personal opinion 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 a system configuration over another

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848

one, then it is assumed that the former has a *higher utility*. A more formal definition of utility is provided further on.

In mobile and ubiquitous computing environments, the context changes frequently. For this reason, systems targeting this type of environments feature multiple configurations and modes (referred to as *variants* in this paper), which are designed to optimize the utility for different subsets of the context space. 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 maximized.

In order to enable a more rigorous study of the problem, we propose a set of definitions for *context*, *variants* and *utility*. These definitions provide the foundation for the proposed approach.

2.1 Context

In this text we follow *Dey*'s definition for context, which is one of the most frequently cited [1]: "*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. Some of these are infinite (*e.g.* time) while some others are limited in value (*e.g.* the user's gender can only be "*male*" or "*female*"). In this perspective, the context might be modeled as a multidimensional space in which each relevant context type defines a dimension (in the case of types with finite value-domains, each value is assigned a range in the dimension). Similar approaches exist in the literature such as the one in [2].

Assuming that each context type can be abstracted by a real number (*i.e.* $\Re$), then a context space of d types can be abstracted as a d-dimensional space $\Re^d$. Then, 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 $(c_t^1, c_t^2, \ldots, c_t^d)$, where c_t^i indicates the context value at dimension i for the time instance t). We refer to these points c_t as *context instances*. More formally, these two terms can be defined as follows:

Context instance

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

Context space

$$\textit{context space} \equiv \Re^d \quad (2)$$

In these definitions, the *context space* (or *context*) is defined as a d-dimensional geometric space, and a context instance is a point (or vector) of the space. Although a geometric analogy is used to characterize the context, no assumptions are made concerning the relation between points, especially their geometric distance. Since discrete states of context can be mapped to (arbitrary) real values, no relation can be guaranteed for neighboring points.

2.2 Variants

In software engineering, there are two main approaches for software adaptation: *parameter adaptation* and *compositional adaptation*. Several aspects of adaptations have been extensively studied, such as *where*, *when*, and *how* they are applied [3].

In the context of mobile and ubiquitous computing environments, adaptivity is required to overcome the variability of these environments. In this respect, systems are designed with adaptive properties so that a system can be configured in different modes (*i.e.* combinations of component compositions and parameter settings), each one of which is designed to offer maximum utility for different conditions of the context. The main characteristic of alternative variants is that they maintain the *functional properties* of the software system, while possibly varying their extra-functional characteristics. In this case, the purpose of the context-aware, self-adaptive system can be seen as the adjustment of the extra-functional properties of the system with the aim of optimizing the perceived utility [4].

Assume that the system supports a finite set of N variants: $variant_1$, $variant_2$, …, $variant_N$. In practice, the set of variants can be infinite. For example, in the case of parameter adaptation the value domain can be infinite (*e.g.* consider a component which can be adapted dynamically by setting an interval parameter to any "positive integer"). In order to simplify the analysis of such systems, it is important to assume that in such cases, 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 values). For instance continuing with the previous example, the "positive integer" parameter can be reduced to {"0", "greater than zero and less than 10", "10 or more"}. In this way, it can be assumed that the number of variants is always finite.

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, the variants are often required to be constructed dynamically. For instance, in component-based systems the variants are constructed by examining the provided and required services (*i.e.* interfaces) of each component [5]. Thus, the exact set of available variants fluctuates according to the availability of components and services and also according to the contextual conditions. The definitions of variants are summarized below:

Variant

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

Variants

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

In this paper, we are primarily concerned with component-based applications, and thus we assume that the system comprises 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.

2.3 Utility Functions

Here, we refer to *utility functions* as mathematical artifacts which map combinations of *context* states and *variants* to scalar values, typically in the range [0, 1] where 0 indicates minimum (worst) utility and 1 indicates maximum (best) utility (*i.e.* quite similar to the notion used in micro-economic where utilities represent user happiness). The choice of the [0, 1] bounds provides the

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848

convenience of allowing the multiplication of different utilities without exceeding the original bounds.

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

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 $f(c_t, v_x) > f(c_t, v_y)$ if and only if the user prefers variant V_x to V_y from her or his 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 section.

3. MULTI-DIMENSIONAL 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 tune and also by setting the vibration off. This implies that when the "default" profile is selected, the user is alarmed for incoming calls with the selected tune and the smart-phone does not vibrate. Different profiles, such as the "meeting", can have different settings such as sound-off and vibration on.

This example is a scenario where the adaptation affects multiple dimensions. For instance, one such dimension is whether there will be a tune played when the phone receives a call or not, and another dimension is whether the vibration will be activated or not. A third dimension, which is not completely cross-cutting, is which tune is played for incoming calls. 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 it can provide the foundation for specifying context-aware, self-adaptive applications, as it will be described later on.

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 possible to define a perfect utility function, because generally users are not 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, without explicitly knowing why and which contextual factors affect their opinion. Furthermore, it is possible that the user's perceived utility depends on factors that cannot be explicitly measured or abstracted, such as their emotional state.

$$\mathtt{U}_{\mathtt{perceived}}(\mathtt{variant}_{\mathtt{X}}) \equiv \sum_{i} \mathtt{U}^{i}_{\mathtt{perceived}}(\mathtt{variant}_{\mathtt{X}}) \quad (6)$$

$$\mathtt{U}_{\mathtt{computed}}(\mathtt{variant}_{\mathtt{X}}) \equiv \sum_{i}^{K} \mathtt{U}^{i}_{\mathtt{perceived}}(\mathtt{variant}_{\mathtt{X}}) \quad (7)$$

$$\mathtt{U}_{\mathtt{computed}}(\cdot) \cong \mathtt{U}_{\mathtt{perceived}}(\cdot) \quad (8)$$

In this text, 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 by formula (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, we define the *computed utility* which is an approximation of the *perceived utility* as shown by formula (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 could be simulated by examining his perception over the video clarity and latency only. It is argued that this approach results to a computed utility which approximates the user perceived utility, as shown by formula (8). Furthermore, it is argued that this approximation provides a reasonable and realistic approach 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 expressed as the *weighted sum* of the dimensional utilities, as shown below:

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

The weights w_i can be adjusted to reflect the importance of each of the monitored dimensions for the targeted user. In this paper, it is assumed that the weights are manually adjusted by the users, but in future work we plan to provide methods and mechanisms that automate this (for example by taking into consideration user feedback that is collected at runtime).

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 when needed. This approach is evaluated in the following section.

4. CASE STUDY-BASED EVALUATION

To illustrate the functioning of the proposed approach, we present a case-study scenario, along with explanations of how the utility functions are evaluated along the scenario. The scenario is about an on-site technician who uses a smart-phone device to assist her in her everyday assignments. In particular, the smart-phone runs a specialized, context-aware application which updates her of any

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848

upcoming tasks even while out of the office and, also, it allows her to interact with her colleagues when she needs to do so.

For simplicity, we consider four dimensions only. Based on these dimensions, the system tries to optimize the user's perception. These dimensions are the *hands-free operation*, the *audio volume*, the *system response* and the *video quality*. The hands-free operation dimension 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. The system response dimension measures the capacity of the system to quickly respond to the user input. The user's need for responsiveness may vary based on her activities (*e.g.* relaxing at home or working on a difficult and stressful task) and its provision can depend on factors such as CPU load and network latency and bandwidth. Finally, the video quality dimension measures the video stream quality (*i.e.* in terms of resolution, refresh rate, and colors), which is relevant when the user uses to a video conferencing application.

A typical day of this user is as follows: The user wakes up and starts preparing for work. She then takes her car to work, at which time she instructs her smart-phone to sync with the enterprise server. When the user arrives at office, she has a video conference with her colleagues to plan the day's activities. Next, she drives to a client's site and her agenda is updated dynamically while she drives. The updates are spoken to her by a text-to-speech system. The user arrives on-site and uses the device to get information in the specifications of the machinery she needs to maintain, and also in order to contact her colleagues by voice when needed.

To enable this sort of scenario, a smart-phone device is assumed, running an appropriate context-aware, adaptation supporting middleware. Such middleware would provide functionality for automatic context management (*i.e.* sensing and access of context data), and for deployment and adaptation of applications.

4.1 Experimental Setup

In the following paragraphs, we describe the experimental setup which is based on the case study scenario and which attempts to demonstrate how the multi-dimensional utilities approach is used.

Table 1: Case study scenario - Possible variants

#	Short name	Description
1	Offline-Visual	No net connection with visual UI
2	Offline-Audio	No net connection with audio UI
3	Offline-Audio-Loud	No net connection with audio UI and loud volume
4	Online-Visual	Online operation with visual UI
5	Online-Audio	Online operation with audio UI
6	Online-Audio-Loud	Online operation with audio UI and loud volume
7	VideoConf-HQ	Video conference mode with high quality streaming
8	VideoConf-LQ	Video conference mode with low quality streaming

First, based on the possible component compositions and the parameter settings, the middleware dynamically constructs the set of possible variants (*i.e.* possible system configurations). These variants specify compositions of the application, as well as values of their configurable parameters. For simplicity, the numerical parameters are quantized to a few values only (*e.g.* the volume is set for either noisy or quiet environments, as opposed to allowing arbitrary values in a range of 0 to 1 for example). A limited and fixed set of variants is defined as shown in Table 1.

Throughout the execution of this scenario, the middleware monitors the context (including the user's occupation, status and anticipated needs) and tries to autonomously select the most suitable variant. This sort of adaptation is decided by means of a feedback control loop, which continuously estimates the utility of each possible configuration through a set of utility functions (in this case the feedback consists of the sensed context values). Applying the four detected dimensions to the utility function of formula (9) results to the equation shown below:

$$U(V_j, C_m) = \frac{w_1 \cdot f_1(\cdot) + w_2 \cdot f_2(\cdot) + w_3 \cdot f_3(\cdot) + w_4 \cdot f_4(\cdot)}{w_1 + w_2 + w_3 + w_4} \qquad (10)$$

In this case, the utility of each variant V_j for some given context conditions C_m is computed as the weighted sum of the *fitness* of the given variant for the specified context, over the four detected dimensions. In this case, the weights are assumed to be values in the range [0, 1] reflecting the importance of each aspect to the targeted user. In this example, it is assumed that all weights are equal (for example set to 1).

Next, the fitness functions of formula (10) are defined, in relation to the context and to the variants' properties. Generally, the fitness functions attempt to evaluate the relevant context types and the corresponding properties of the variant into question. Their goal is to return higher values for better matches of the given context with regards to the dimension into consideration. An example of how the functions of the case study example would be constructed with pseudo-code is depicted in the following:

$f_1(\cdot) \equiv \text{Utility}_{\text{hands-free}}$:
  if(context.user.state is driving OR manual_work)
  then return variant.offered_hands-free
  else return 1 - variant.offered_hands-free

$f_2(\cdot) \equiv \text{Utility}_{\text{audio-volume}}$:
  1 – diff(context.env.noise_level, variant.offered_audio-volume)
diff(x, y): a function which compares two values x, y and returns a higher value the more different they are (min 0 and max 1)

$f_3(\cdot) \equiv \text{Utility}_{\text{system-response}}$:
  if(context.user.state is manual_work)
  then (0.7*variant.offered_sr + 0.3*context.resources.cpu_avail)
  else (0.5*variant.offered_sr + 0.5*context.resources.cpu_avail)

$f_4(\cdot) \equiv \text{Utility}_{\text{video-quality}}$:
  if(context.user.state is video_conferencing)
  then (0.6*variant.offered_vq + 0.4*context.resources.net_bw)
  else 0

Figure 1: Fitness functions of the case study scenario

Figure 1 shows how the fitness functions are generally expressed in terms of context values (*e.g. context.user.state*) and variant properties (*e.g. variant.offered_hands-free*). Based on the body of these fitness functions, the context needs can also be computed. To better understand the nature of the dimensional (fitness) utility functions, consider that had the user being interested in just one

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848

dimension of the adaptation domain (*e.g.* hands-free option), then that fitness function (*e.g.* f_1) would correctly rank the variants for a context state, based on their offered hands-free properties only.

As mentioned already, it is assumed that the middleware provides automatic management of context sensing, and asynchronously informs the adaptation reasoning engine whenever some relevant context change occurs (*i.e.* a change in the above context types). The relevant context types for this scenario are shown in Table 2.

Table 2: Case study scenario - Needed context types

Context type	Description & Values
user.state	Describes the state of the user (*i.e.* whether she or he is working, resting, sleeping, etc)
	Values: driving, manual_work, video_conf, resting, sleeping
env. noise_level	Describes the level ambient noise (*i.e.* noise in the environment of the device)
	Values: lower, low, medium, high, higher
resources. cpu_avail	Describes the availability of CPU based on the work load and the running applications
	Values: 0:0.1:1, where smaller values indicate lower availability and higher values indicate higher availability
resources. net_bw	Describes the bandwidth of the network, as a percentage of the total available (*e.g.* assuming a 54Mbits 802.11G interface)
	Values: 0:0.1:1, where 0 indicates no network connection, 1 indicates 54Mbps bandwidth availability, etc

The other constituent of fitness functions are the variant properties. These properties describe the variants in terms of specific characteristics, such as for example their ability to operate in hands-free mode. These are shown in Table 3.

Table 3: Case study scenario – Variant properties

Variant	ohf	oav	osr	ovq
Offline-Visual	0	0	0.2	0
Offline-Audio	0.8	0.5	0.2	0
Offline-Audio-Loud	0.9	1	0.2	0
Online-Visual	0	0	0.8	0
Online-Audio	0.8	0.5	0.8	0
Online-Audio-Loud	0.9	1	0.8	0
VideoConf-HQ	0.5	0.5	0.7	1
VideoConf-LQ	0.5	0.5	0.7	0.7

4.2 Experiment Scenes

Based on the experimental setup of Section 4.1, we define a detailed scenario, which comprises a number of individual scenes. For each one of these scenes, we define the context values for the relevant context types (As mentioned already, it is assumed that the middleware provides automatic management of context sensing, and asynchronously informs the adaptation reasoning engine whenever some relevant context change occurs (i.e. a change in the above context types). The relevant context types for this scenario are shown in Table 2.

Table 2) and we compute the utilities for each dimension independently, as well as the overall utility. Based on the scenario described at the beginning of this Section, we define the scenes as shown in Table 4.

Based on these settings and their specified context values, the adaptation reasoning engine evaluates the dimensional utilities for each of the four aspects, as well as the overall utility as the weighted sum of these utilities. Based on these, an individual ranking of the variants is inferred for each of the examined context states. These results are illustrated in Table 5.

Table 4: Case study scenario - Experiment scenes

Scene #	Description & Context			
Context	**user.state**	**env.noise**	**res.cpu**	**res.net_bw**
Scene #1	The user is still at home, and syncs her smart-phone's agenda over the slow home network.			
	resting	low (0.1)	hi (0.9)	med (0.5)
Scene #2	The user enters her car and drives to work. She wants to continue receiving updates in her agenda.			
	driving	med (0.5)	hi (0.8)	low (0.2)
Scene #3	The user sits at her office and has a video conf with her colleagues to plan the day activities			
	video_conf	med (0.5)	hi (0.7)	hi (0.9)
Scene #4	The user arrives to the client's site and performs manual maintenance on the installed equipment			
	man_work	hi (0.9)	lo (0.3)	med (0.6)

As illustrated in Table 5, a different variant is selected for each scene (*i.e.* context conditions). For the first scene, the typical online variant with visual UI is selected. This is a reasonable choice as the home environment is characterized by low ambient noise, medium network bandwidth and the user is capable of using the preferred visual UI mode of interaction. The overall (average) utility is computed to be 0.64, with the corresponding computed utilities for *hands-free*, *audio-volume*, *system-response* and *video-quality* set to 1, 0.9, 0.67 and 0 respectively. The rest of the selections are also argued to be reasonable for the contextual conditions of each scene.

Table 5: Adaptation decisions based on dimensional utilities

Scene #	Selected Variant	Score overall (dimensional)
Scene #1	Online-Visual	0.64 (1, 0.9, 0.67, 0)
Scene #2	Online-Audio	0.61 (0.8, 1, 0.64, 0)
Scene #3	VideoConf-HQ	0.76 (0.5, 1, 0.56, 0.96)
Scene #4	Online-Audio-LD	0.61 (0.9, 0.9, 0.65, 0)

The complete set of computed utilities (both dimensional and overall) can be computed with a small script implementing the dimensional utility functions. This can be especially useful for the developers of the context-aware applications, as it allows them to fine tune both the variant properties (as shown in Table 3) and the dimensional utility functions (*i.e.* the fitness functions as shown in Figure 1). This kind of matrices provides important insight to the developers concerning the correctness of the utility functions as well as the functioning of the context-aware system early on at development time. In the next section we compare the proposed multi-dimensional model to the related work, and we argue on its advantages and limitations.

5. RELATED WORK

The current state of the art refers to three main types of adaptation approaches, namely action-based, goal-based and utility-function based [6]. In this paper we are concerned with utility function-based approaches, which assign values (utilities) to adaptation

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848

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 proposed a middleware which uses a utility-based, architectural approach to adaptation [7]. 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 [8], 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 [7] 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.

As it is stated in [9], interfacing with humans is one of the main challenges in designing and implementing autonomic computing systems. Quoting the author, "*the difficulty with utility functions is that humans find them difficult and awkward to specify*". The results of this paper target primarily developers of context-aware, self adaptive systems. It is argued that the proposed model can be of significant help as it adopts the Separation of Concerns (SoC) approach to allow the developers to concentrate on an individual aspect of the adaptive behavior of the system at a time. Further on, it is argued that the proposed model can facilitate reusability of the context-aware and adaptation properties of the components (and applications), which is a significant gain for the developers.

Finally, while this approach appears to have some potential in the field of context-aware, self-adaptive applications, it naturally comes with a few limitations as well. For instance, the tuning of the utility functions can be quite cumbersome and difficult to be performed. In this respect, we are working on a mechanism which tries to automatically perform that task with minimum user-intervention, using results from the control theory field. Another limitation concerns the case where a large number of variants is deployed which poses scalability issues. However, as we are primarily concerned with small mobile and embedded devices running only a few context-aware applications, it is argued that this problem will not arise too frequently.

6. CONCLUSIONS

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 introduced a novel, multi-dimensional utility model which mitigates the complexity inherent in the development of such systems.

The improvement is achieved by introducing a utility function-based approach that allows the developers to focus on a specific aspect of the context-aware, self-adaptive behavior at a time. The adaptations are decided by matching the offered properties of each variant to the contextual conditions, and then repeating this for each relevant dimension. Furthermore, this approach offers high reusability as both the adaptation properties of the variants and the utility functions of the system are highly reusable.

7. ACKNOWLEDGMENTS

This work was financially supported by the EU as part of the IST-MUSIC project (6th Framework Programme, contract no. 35166).

8. REFERENCES

[1] Dey, A. K. 2001. Understanding and Using Context. Personal Ubiquitous Computing, Vol. 5, No. 1, pp. 4-7.

[2] Padovitz, A., S. W. Loke, A. Zaslavsky. 2004. Towards a Theory of Context Spaces, 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04), IEEE Computer Society Press, pp. 38-42.

[3] McKinley, P. K., S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. 2004. Composing adaptive software. IEEE Computer, Vol. 37, No. 7, pp. 56-64.

[4] Paspallis, N., and G. A. Papadopoulos. 2006. An approach for developing adaptive, mobile applications with separation of concerns. 30th International Computer Software and Applications Conference (COMPSAC 2006), Chicago, USA, IEEE Computer Society Press, Vol. 1, pp. 299-306.

[5] Cervantes, H., and R.S. Hall. 2004. Autonomous adaptation to dynamic availability using a service-oriented component model. 26th International Conference on Software Engineering, (ICSE 2004), Edinburg, Scotland, UK, pp. 614-623.

[6] Walsh, W. E., G. Tesauro, J. O. Kephart, and R. Das. 2004. Utility functions in autonomic systems, International Conference on Autonomic Computing (ICAC), New York, NY, USA, 17-18 May 2004, IEEE Computer Society Press, pp.70-77.

[7] Floch, J., S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, E. Gjorven. 2006. Using architecture models for runtime adaptability. IEEE Software, Vol.23, No.2, March-April 2006, pp. 62-70.

[8] Alia, M., V. S. W. Eide, N. Paspallis, F. Eliassen, S. Hallsteinsen, G. A. Papadopoulos. 2007. A Utility-based Adaptivity Model for Mobile Applications, 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), Niagara Falls, Ontario, Canada, May 21-23, 2007, IEEE Computer Society Press, pp. 556-563.

[9] Kephart, J. O. 2005. Research challenges of autonomic computing, 27th International Conference on Software Engineering (ICSE 2005), St. Louis, MO, USA, pp. 15-22.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3848
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3848