A Utility-based Adaptivity Model for Mobile Applications

Mourad Alia
Simula Research Laboratory
Martin Linges v. 17, Fornebu
1325 Lysaker, Norway
mouradal@simula.no

Viktor S. Wold Eide
Simula Research Laboratory
Martin Linges v. 17, Fornebu
1325 Lysaker, Norway
viktore@simula.no

Nearchos Paspallis
Dept. of Computer Science
University of Cyprus
1678 Nicosia, Cyprus
nearchos@cs.ucy.ac.cy

Frank Eliassen
Dept. of Informatics
University of Oslo
P.O.Box 1080, Norway
frank@ifi.uio.no

Svein O. Hallsteinsen
Dept. of SE, Safety and Security
SINTEF ICT, Norway
S.P. Andersens 15 b, Trondheim
Svein.Hallsteinsen@sintef.no

George A. Papadopoulos
Dept. of Computer Science
University of Cyprus
1678 Nicosia, Cyprus
george@cs.ucy.ac.cy

Abstract

Mobile environments are characterized by resource fluctuations and limitations, and variations in user preferences. Therefore mobile applications need to be adaptive to retain usability, usefulness and reliability. In our approach to support adaptivity, we combine context awareness, reflection and component composition planning. The planning is done by generic middleware and supports dynamic discovery, utility-based and context-aware evaluation, and selection of the best implementation alternative of a given mobile application. In this paper we present a formal model of our approach and use this model to show the expressiveness of utility-based adaptation policies. To demonstrate the feasibility and expressiveness of our approach we include a case study based on a real adaptive application built using our model and middleware.

1. Introduction

With the proliferation of mobile and pervasive computing the typical operating environment for software is increasingly characterized by heterogeneity and unpredictable fluctuations in both computing resources, environmental contexts, and user needs and preferences. In this setting, adaptivity becomes crucial and there is an increasing need for development methods to construct adaptive systems able to continuously and autonomously ensure the safety of mobile applications. In our context, the term safety refers to the requirement of retaining the *usability*, *usefulness* and *reliability* of the user applications with regard to dynamic variations of the mobile environment.

Generally, adaptive systems are context-aware systems in the sense that adaptations are always performed in response to context changes. Typically, adaptation is handled by an adaptation control loop that senses the relevant context information, makes decisions on the adaptations to be made, and implements these decisions by dynamically reconfiguring the system. To overcome the complexity of building such adaptation loops, component-based approaches combined with reflection mechanisms have been widely adopted in many systems [2, 6, 11, 12, 14]. Using software components and reflection enables introspection of the structure, behaviour and context dependencies of the system and thereby supports dynamic reconfiguration.

However, the decision making component of the adaptation loop is still a challenge. The aim of decision-making is to choose carefully among possibly many alternative application configurations where only one or a limited number of them can be applied in a given context. The decision is made according to certain user-provided adaptation policies that allow the adaptation engine to automatically search and select the appropriate configuration in the whole search space of configurations. The implementation of such decisions may involve to dynamically replace components or compositions or to tune component parameters, resource reallocation, etc.

In our earlier work, we have mainly discussed the overall architectural issues related to the design of a planning based middleware that implements the adaptation control loop [11, 2] by using software components and by following the separation of concerns principle. We have also considered the scalability issue of the planning process that implements the decision-making [1]. In this paper, we rather focus on the expressiveness of the adaptation policies subsumed in our previous works. These policies are specified according to what we call the adaptivity model. Basically, this model combines component composition and context awareness and associates different application component composition variants to context states. It uses utility as a function of application level properties, user preferences and system level state to select the best application configuration. The viability and usability of our model as a generic approach to handle adaptivity in mobile environments has been validated through the experience gained by implementing several running adaptive applications in two research projects using planning-based middleware, namely the QuA[1] and MADAM[2] projects.

The rest of this paper is organized as follows. Section 2 discusses the context and the scope of adaptation targeted by our model in the mobile setting. Section 3 proposes and formalizes the adaptivity model subsumed by the planning process that selects the best application variant. Section 4 is devoted to validating our model through presenting a running application. Subsequently, section 5 provides an analysis of our model with regard to the state of the art before a conclusion is provided in section 6.

2. Mobility and adaptation scope

In this paper, we assume that the adaptation decisions are taken with the goal of maximizing the user benefit of the applications or services. The proposed taxonomy of adaptation aims at providing a more precise meaning to the adaptation term through a set of adaptation types that may potentially be composed to form different adaptations use cases and scenarios.

- *User interface delegation* — transferring (some of) the UI functionality to another device or peripheral. For example, a driver uses the car computer to establish hands-free interaction with her handheld device.
- *Functional richness* — extending the functionality of an application by providing access to new hardware or newly discovered services.
- *Network availability* — selecting between different network technologies, or switching to and from the offline mode.
- *User-Interface presentation* — tuning of the corresponding UI device, so that the experience delivered to the user is optimized.
- *User and application session redeployment* — redeploying components and/or applications to different devices to improve efficiency.
- *Data richness* — changing the quality of some data. As an example, when the bandwidth available for a video conference session varies, changing the video compression might be required to ensure that the best quality possible is preserved.
- *Security* — adjusting the security mode, for example to compensate for changes in the device's context and especially the networking infrastructure.
- *Software mode* — switching between different modes of operations that correspond to different software architecture alternatives in different contexts. It is assumed that the application developers make these modes available.

As the purpose of the adaptations is to maintain the highest level of user satisfaction, the decisions on which adaptations to perform are taken based on context.

Environmental and computing resource changes are the most important factors considered by the adaptation process. Computing resource changes include events such as fluctuations in their properties (e.g. network bandwidth or memory capacity), discovery of new resources (e.g. a new network) or existing resources becoming unavailable. Environmental context changes cover physical factors such as noise, light, location, etc.

Apparently, the perception of utility depends on user preferences. Therefore *user preferences* constituting the second important factor must also be considered when making adaptation decisions. User preferences are not static but may evolve either because of changes in user mood or occupation, or simply because of a change of mind.

Another important adaptation trigger is the *availability of component implementations*. When applications evolve, new components may be added in order to cover new functionality or improve existing ones. Also, when users move, new service providers may come into reach and others may disappear. These changes imply that additional component implementations, even ones not existing at development time, may become available while the system is running and thereby creating opportunities for improvement.

Finally, *starting and stopping applications* changes the demand for resources which obviously may have a serious effect on the behaviour of other running applications.

3. Adaptivity model

The objective of our work on adaptivity modelling is to provide a reusable approach and methodology for constructing adaptive systems based on software components and composition paradigms. The model provides a methodology for the designer (or developer) to express the adaptivity policies that drive the adaptation control loop. These policies embody the different parameters that influence

[1]Quality of Service Aware Component Architecture: http://www.simula.no/departments/networks/projects/QuA

[2]Mobility and ADaptation enAbling Middleware: http://www.ist-madam.org/consortium.html

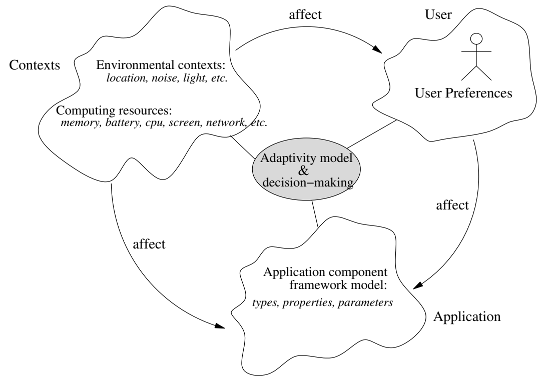
affect
User
Contexts
Environmental contexts:
location, noise, light, etc.
User Preferences
Computing resources:
memory, battery, cpu, screen, network, etc.
Adaptivity model & decision–making
affect
affect
Application component framework model:
types, properties, parameters
Application

Figure 1. Adaptivity Model: User, Applications and Contexts

the adaptation, namely the user requirements, the different computing and environmental context information and constraints for a given application. As depicted in figure 1, the adaptivity model should consider the complex inter relationship between the context elements that affect both the user (noise, light, etc.) and the application configurations, and the user needs that also affects the application configuration variants to be chosen during adaptation.

To be able to balance this multitude of concerns, we introduce utility functions that automate the decision-making process. Utility functions allow the automatic selection of an appropriate application configuration among all possible application configurations. Formally, the adaptivity model *AM* is defined as

$$AM = (A, X, Q, T, U)$$

where *A* represents the component framework model of the adaptive application, *X* represents context dimensions, *Q* represents the set of considered adaptivity QoS dimensions, *T* represents user preferences, and *U* the utility function.

3.1. Application component framework model

Adaptive applications are designed as component frameworks with explicit variability. A component framework specifies roles and interfaces, and regulates the interaction between components *plugged into* the framework. These frameworks are designed and implemented using minimal reflective and hierarchical component models, such as Fractal [4] and OpenCom [8], enriched with property annotations. The application is then decomposed into a set of composed component types where each type identifies a variation point. Obviously, this design depends on the application to be adapted and depends on a preliminary analysis of its adaptive behavior before specifying the different component types.

Each component type is associated with a set of required and provided properties. These properties cover varying aspects of the component behavior. It is distinguished between properties that are related to resources and those that are independent from resources consumption. E.g., a component that displays graphics could be annotated with a provided property related to the luminance of the image. The role of properties that are related to resources is twofold. On the one hand, they are exploited during the planning in order to select only the configuration variants that can be satisfied with the available amount of resources. On the other hand, they are used to derive and calculate the adaptivity QoS needs (reliability, load, etc.) as described in the next section.

Formally, the application component framework *A* for a given application is defined as:

$$A = (G, P)$$

where *G* represents the graph of components that compose the application (edges correspond to composition dependencies between components):

$$G = \{c_1, c_2, \ldots c_n\}$$

and *P* represents a set of properties:

$$P = \{p_1, p_2, \ldots p_m\}$$

The set of properties *P* is the union of all properties provided by each individual component of the composition.

For each application component framework there are associated *property predictor functions*. These functions are used to predict the property values resulting from the composition of component properties. For each property p_i of *P* is associated a function that takes as input the set of composed component properties and returns the value of p_i:

$$p_i = \xi_i(p_i^{c_1}, p_i^{c_2}, \ldots p_i^{c_n})$$

where: $p_i^{c_k}$ represents the property *i* of the component *k* of the composition. For example, the predictor function of the memory consumption of the application can be realized as the sum of memory consumption of each selected component. It is not an easy task to generalize this function because it depends on the target component programming model and application domain. However, models and algorithms such as those presented in [5], may be adopted in our case as it is also argued in [13].

Finally, regarding parameterized components, the designer defines *parameters to properties* functions that simply associate the different value configurations of the parameters $a_1, a_2, \ldots a_r$ of a parameterized component *k* to a corresponding set of property values.

$$\eta^{c_k}(a_1, a_2, \ldots a_r) = \{p_1, p_2, \ldots p_m\}$$

The different variants are then considered as different implementation alternatives of the same component type.

3.2. Context dimensions

By context, we refer to any information which relates to a particular interaction between a user and an application. *Computing context* refers to any information which describes the state of the hosting device, such as battery capacity and available memory. *Physical context* represents any information which describes the physical properties of the interactions, such as the geographical location, the temperature, and the weather.

Finally, *user context information* describes the state of the relevant user, such as whether she is sleeping, driving, attending a lecture, or even the mood of the user such as happy and sad. In our model, a context is modelled by a multi-dimensional space in which each dimension corresponds to an individual context aspect:

$$X = R \cup E \cup I$$

where *R* represents a set of resource properties of the execution platform, *E* is a set of environmental properties, and *I* represents a set of user context properties.

Under this assumption, points in such a space correspond to context states and a context change is defined as the transition from one context state to another. Assuming this abstraction, adaptive systems are designed so that the most suitable configuration is dynamically selected whenever there is a transition from one context point to another.

Within an adaptive system, the above context model is instantiated and monitored through a context management component which is tightly related to the decision-making process. Such a context management system would be responsible not only for sensing and encoding the context information, but also to further process it and to filter out any irrelevant data (i.e. noise). Furthermore, it is important that advanced context management mechanisms are in place with the aim of deriving more refined information, releasing the adaptation decision-making mechanism from the burden of ruling out irrelevant context change events.

Typically, the interaction between the adaptation decision-making component and the context manager is carried through an interface which allows the former to register for notifications of very specific events. In this way, one only needs to detect the critical context changes (i.e. those which can trigger an adaptation), and register for the corresponding context change events only.

3.3. Adaptivity QoS dimensions

Adaptivity QoS dimensions provide a first stage of mapping and an understandable level of adaptation reasoning for the user. The QoS dimensions represent higher level representative properties as perceived by the user. For a given adaptivity model, the adaptivity QoS dimensions *Q* are defined by a vector of *l* adaptivity property dimensions:

$$Q = [q_1, q_2, \ldots q_l]$$

Some of these properties are atomic in the sense that they are a subset of the different properties of the application component model (response time, network latency, etc.) or part of the context model dimensions (e.g. light and noise). Some others are composite properties in the sense that they define more high level QoS properties, such as availability, reliability, operability, etc. that are calculated from lower level properties. For example, the operability dimension of a mobile user depends on light and noise sensitivity, hands occupation, etc. Such dimensions may be quantified using dimensional utility functions (see section 3.5).

3.4. User preferences

User preferences are a collection of predicates or functions that express user constraints and needs that should be fulfilled by the adaptation. Generally, user preferences should be understood as wishes whose chance of satisfaction should be maximized, but which cannot always be completely fulfilled. Within the adaptivity scope targeted by this paper, the adaptivity model allows the user to express at least two kinds of preferences namely inter adaptivity QoS dimensions and inter application preferences: $T = (W^q, W^A)$

By considering the targeted set of adaptivity QoS dimensions *Q*, the *inter adaptivity QoS dimensions* preferences allows the user to express the level of importance of each individual adaptivity QoS dimension, as well as tradeoffs among these dimensions. User satisfaction of a given QoS dimension is usually expressed as a utility function that maps the value or the ranges of values of this QoS dimension to a value that represents the degree of satisfaction of the user for different QoS values in this dimension. The tradeoff between the different adaptivity QoS dimensions is expressed using weights that the user associates relatively to each dimension. For each adaptivity QoS dimension from Q respectively $q_1, q_2, \ldots q_l$ is associated a corresponding weight $W^{q_1}, W^{q_2}, \ldots W^{q_l}$. For example, these preferences allow expressing the tradeoff between the choice of video quality and the network cost, assuming that networks with high bandwidth are more expensive than those with low bandwidth. Richer examples are given in section 4.

The *inter applications* preferences allow the user to express his preferences among the different running applications. As in the case of inter adaptivity QoS dimension preferences, the user can simply assign relative weights for each application. For *t* applications $A_1, A_2, \ldots A_t$ are assigned respectively the following weights $W^{A_1}, W^{A_2}, \ldots W^{A_t}$.

Note that additional preferences models such as those related to the user choices and constraints upon requested services may be needed to express inter services user's requirements. This kind of preferences may be handled earlier during component discovery and querying before the planning process and therefore is out of the scope of our model.

3.5. Aggregated utility functions

The aim of the utility function is to express the quality of the adaptation for a given user. It takes as input adaptivity QoS dimensions and user preferences and selects the application variants that best satisfy the user within the current context state including the available resources.

Utility functions are in general n-dimensional functions taking values from an n-dimensional QoS space from the vector *Q* as argument. In our model, we adopt a simpler approach, also adopted in [3], which is to define overall utility as a weighted sum of a set of dimensional utility functions. A dimensional utility function measures user satisfaction in one QoS-dimension only. The weights of the overall utility functions usually correspond to the relative level of importance of each QoS-dimension as preferred by the user. Utility functions normally map the degree of satisfaction into a real number, often in a given bound range $[a, b]$, where a indicates that the corresponding quality of the service is below the minimum required by the user, while b indicates that the corresponding quality is at or above the required maximum quality level.

The dimensional utility function associated to the QoS dimension q_i which depends on *j* properties is:

$$F_(q_i) = f(x_1, \ldots x_j) \ \ where \ \ x_i \in P \cup X$$

Then, the utility function related to a given application *A* qualified by the Adaptivity QoS dimensions *Q* is:

$$U(A) = \sum_{i=1}^{l} W^{q_i}.F(q_i) \tag{1}$$

This function is an objective function that should be maximized during component variants selections. In the presence of many applications, this utility function *F* is

$$U = \sum_{i=1}^{t} W^{A_i}.U(A_i) \tag{2}$$

In both cases, the amount of resources needed by the selected variants must not exceed the available computing resources:

$$\sum_{i=1}^{t} R(A_i) \leq R \tag{3}$$

In order to improve the performance of the planning process, the utility function evaluation is the ultimate stage of the decision making process after an early filtering. The objective of this filtering is to reduce the search space. This includes:

- Discarding the non recommended component variants (or services) following the user *inter services* preferences. This may for example exclude some service providers based on user preferences (e.g language choices).

- Discarding all the unfeasible application variant compositions. This includes component composition constraints related to the semantics of the application. For example, a given encoding component should match the decoding component within an application variant. These composition constraints cover also the versioning problem.

- Discarding the variants that could not be satisfied with the amount of available resources. The planner checks through the needed properties of a given component variant alternative if its required resource properties could be fulfilled.

4. Validation

The adaptivity model has been validated through different applications uses cases that include commercial products, in the context of the QuA and the MADAM projects. Roughly, both MADAM and QuA middleware architectures follow the IBM autonomic element architecture[3] applied to components and component frameworks. This architecture implements the adaptation control loop in which the context manager senses the different relevant context information and notifies the adaptation manager to select the best composition variant which in turn communicates the new configuration to the configurator component which is responsible to derive the reconfiguration tasks that transform the system to the new adapted state. In the following, we have selected the QuA *PMS* (Personal Media Service) case study. For more different adaptation use cases, we invite the reader to consult the MADAM deliverable D1.2.

4.1 The Personal Media Service

The Personal Media Service (PMS) can be viewed as an in-house personal proxy service for the delivery of multimedia content. Any media content accessible from home is made by the PMS accessible from anywhere, assuming Internet connectivity. The PMS adapts the content of media streams to the capabilities of the client device and the quality of the network connection, while taking user preferences into account. The PMS application was developed to support a user watching a live video while on the move.

[3]An Architectural Blueprint for Autonomic Computing: http://www.ibm.com/autonomic

Being self-adaptive, the PMS was designed to continuously monitor variations in *network availability* and to adapt itself in order to maintain the best possible user experience.

With respect to *data richness*, the PMS may manipulate the video quality in a number of different quality dimensions before sending the stream to a client device, exploiting the data richness to accommodate the current bandwidth availability. While on the move, a user may choose to change the device used for accessing media streams. E.g., a user may take advantage of a home cinema display to receive a stream from the PMS in high quality. When leaving home, the user may prefer the session to continue uninterrupted and therefore to hand over the ongoing session to a mobile device. This illustrates the need for *user interface delegation*, where the streaming session is transferred from a client device to another. Such handover may also require *user and application session redeployment*. The PMS is designed to support this kind of adaptation, even in the middle of a session. Data richness is also necessary for supporting client devices ranging from PDAs (tiny displays and scarce bandwidth resources) to workstations (huge displays and high speed networks). With respect to *software modes*, the PMS switches between different architectures at runtime in response to contextual changes, while trying to meet the user requirements.

4.1.1 Adaptivity QoS dimensions

The PMS software takes advantage of multi-dimensional scalable video streaming [10]. Scalability is realized by a layering scheme. A base layer provides the lowest quality. Several enhancement layers, each building on the layers below, allow for fine grained selectivity of the quality received and the bandwidth needed to carry the video data.

The QoS dimensions supported are *temporal quality*, *luminance quality*, and *chrominance quality*, denoted by t, y, and c respectively. The PMS also extends the adaptive behavior provided by the scalable video coding scheme by introducing so-called *time-shift* capability.

The *time-shift* capability allows the PMS to cover the case when the available bandwidth is insufficient for streaming even the lowest quality acceptable to the user by pausing until the network becomes usable. During a pause, the frames are buffered by the PMS in order to resume streaming once the connection is reestablished. Such buffering introduces delay, a *time-shift*. However, the delay may get steadily reduced by streaming the video slightly faster from the buffer compared to the buffer arrival rate. Such *time scaling* results in somewhat increased playout rate at the client device, while catching up with the stream arriving at the PMS.

Users may have different preferences regarding such *time-shift* behavior. In order to handle the behavior in a user specific way, we introduce two additional QoS dimensions, namely *temporal displacement* and *time scale ratio*. The temporal displacement dimension represents the number of seconds introduced by time-shifting, while the time scale ratio is the speed of the presented video relative to the speed of the original stream, denoted by d and r respectively.

Different time-shift configurations are supported, some having time scale ratio less than 1.0 and others greater than 1.0. This allows streaming to continue in situations where bandwidth availability drops below what is necessary for live streaming, by using a time scale ratio less than 1.0. Some users may prefer such a slowdown for a shorter period of time, instead of a pause.

Table 1. Dimensional weights

User 1				User 2				User 3			
W^t	W^y	W^c	$W^{d,r}$	W^t	W^y	W^c	$W^{d,r}$	W^t	W^y	W^c	$W^{d,r}$
0.6	0.2	0.1	0.1	0.1	0.5	0.3	0.1	0.25	0.25	0.25	0.25

4.1.2 Utility functions and application variants

In the PMS case, utility is defined as a weighted sum of dimensional utility functions:

$$U(t,y,c,d,r) = W^tF(t) + W^yF(y) + W^cF(c) + W^{d,r}F(d,r)$$

Table 1 illustrates how different users may specify the relative importance of the different quality dimensions by assigning weights to each dimension. E.g., the first user regards temporal quality as most important. The second user perceives luminance quality most important, while the third user regards the different dimensions as equally important.

Table 2 illustrates the dimensional utility functions for the three different users, defined as a set of coefficient values. Each coefficient specifies the utility value for a quality layer of a QoS-dimension. Both the first and the second user accept quality degradation in the *temporal*-, *luminance*-, and *chrominance quality* dimensions. In contrast, the third user does not accept reduced frame rate and only slightly reduced luminance and chrominance quality. Note that the temporal dimension cannot be omitted, while either the luminance or chrominance part of the video signal can. Table 3 illustrates the dimensional utility for the *temporal displacement* and the *time scale ratio* dimensions. As can be seen from the table, these dimensions are dependent. Users may specify that they prefer to increase the *time scale ratio* if they fall behind the original stream. E.g, the first user prefers no time scaling in the live streaming case and speedup in the time-shift case, while the second user does not accept any slow down or speed up. The third user accepts a slow down in order to reduce the risk of pauses. If such a pause is introduced due to insufficient network bandwidth, the third user prefers a speed up in order to catch up with the original stream (and even more so as the temporal displacement increases).

Table 2. Dimensional utility functions (t, y, c)

	User 1			User 2			User 3		
Layers	$F(t)$	$F(y)$	$F(c)$	$F(t)$	$F(y)$	$F(c)$	$F(t)$	$F(y)$	$F(c)$
+3	0.9	0.7	0.4	0.4	0.9	0.7	0.9	0.7	0.7
+2	0.8	0.6	0.3	0.3	0.8	0.6	0.0	0.6	0.6
+1	0.7	0.5	0.2	0.2	0.7	0.5	0.0	0.0	0.0
base	0.6	0.4	0.1	0.1	0.6	0.4	0.0	0.0	0.0
null		0.0	0.0		0.0	0.0		0.0	0.0

Table 3. Dimensional utility functions (r, d)

d	User 1			User 2			User 3		
	r			r			r		
	0.8	1.0	1.2	0.8	1.0	1.2	0.8	1.0	1.2
0	0.0	1.0	0.0	0.0	1.0	0.0	0.4	1.0	0.0
$< 0 - 10]$	0.0	0.0	1.0	0.0	1.0	0.0	0.3	0.7	0.9
$< 10 - 60]$	0.0	0.0	1.0	0.0	1.0	0.0	0.2	0.6	0.9
$< 60-$	0.0	0.0	1.0	0.0	1.0	0.0	0.1	0.5	1.0

The PMS is realized by means of three different architectures, live-streaming, storage, and time-shifted streaming, illustrated in Figures 2, 3, and 4.

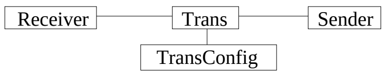

Figure 2. PMS live streaming configuration

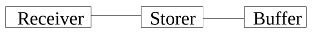

Figure 3. PMS storage configuration

5. Discussion and related work

By adopting a general architectural approach, we have proposed a domain independent adaptivity model in the sense that it copes with different adaptation types presented in section 2. This is true since we suppose that applications are entirely designed using component frameworks and the planning generates the whole application configuration that fits the required adaptation (performance, disconnections, ...). Furthermore, our approach does not disturb the modularity and reusability properties when building adaptive applications. The scope of reuse here is not only from a functional viewpoint but also from a quality viewpoint. Indeed, some property predictor functions are application dependent, while others are general and therefore may be reused into different adaptation models (e.g. resource consumption properties). Component parameter mapping functions, however, are usually application logic dependent.

With regard to the state of the art, [16] distinguishes three types of adaptation approaches namely *action based*, *goal based* and *utility function based*. Since goal based approaches are generally considered to be less advanced and are less popular than the other two, we concentrate our discussion on action based and utility functions based approaches.

In action-based policies, rules expressed as condition-action pairs are used to declare the system's behavior and establish the actions to be executed in response to context changes. In most of policy-based middlewares [14, 6], it is common to use action based approaches to express and manage the dynamic reconfiguration of adaptive systems. The management of the set of policies present in the system, includes the resolution of conflict and incoherencies before deriving the actions that adapt the application.

As in our approach, policy-based middlewares tend to provide general models that facilitate the development of adaptive applications following the separation of concerns principle using software components. However, while the adopted action based policy is natural and simple to use, it presents some drawbacks compared to our utility function-based policy approach [16]. Indeed, this policy does not consider mapping between different adaptivity QoS dimensions and therefore does not offer any methodology that allows reasoning on adaptivity dimensions composition. Consequently, it provides less transparency to the developer by reasoning in terms of reconfiguration (i.e. actions on low-level details of system functions) instead of architectural design (i.e. variability-based design).

Utility function-based approaches aim at delivering the best possible decision rather than just a feasible one as is the case of the goal-based adaptation. In this respect, [15] and [13] are the closest approaches to ours in the sense that the subsumed adaptivity model also considers the component composition selection problem within a constrained environment. The main differences between [15] and our approach is that the proposed adaptivity model does not consider the composition of component properties when composing components and it also does not consider contextual information dimensions related to the mobile environment. Concerning the QoS-oriented composition analysis model presented in [13], the main difference is that it considers only a class of adaptation types that are related to embedded systems scope and therefore it does not take into account either user needs and preferences or contextual information dimensions.

While utility functions provide a rich model to express the adaptation policies, it is generally complex and challenging to define and compute general utility functions [7]. Rather we believe a system architect should define these functions for a given application. We think that some results from the multi-attribute utility theory developed by the AI research community may be exploited to overcome this difficulty, such as the usage of languages for composing utility measures and construction of libraries of standard forms for utility functions [9].

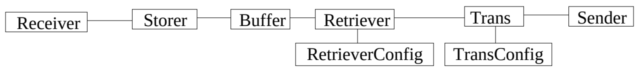

Figure 4. PMS time-shift configuration

6. Conclusions

This paper has reported on results from two related projects both addressing adaptation middleware, focusing on how to express adaptive behavior of an application in terms of an adaptivity model. This model alleviates the complex task of developing adaptive applications for a mobile environment by clearly separating the adaptation concern from the application logic, as well as separating different aspects of adaptivity, such as adaptation decision making, adaptation polices, context dependencies and user preferences. Furthermore, it makes it possible to leave much of the complexity of adaptivity to generic middleware leveraging the reflection principle. The backbone of the model that binds the different aspects together, is a component oriented view of the application architecture defining component types and how components are connected to create the application.

The proposed approach has been analyzed and validated experimentally through the development of adaptation middleware leveraging the adaptation model to perform adaptations in response to context changes and several adaptive applications intended for mobile use. Our experience from developing and testing these applications indicate that our model is both viable and usable as a generic approach to handle adaptivity in mobile environments.

Currently, we are extending our model so that to cover distribution by considering the placement of applications components on different available computing nodes as part of the planning process. Both centralized and decentralized adaptation alternatives are considered depending on the target environment properties and the scalability issue.

7 Acknowledgements

The authors would like to thank Eli Gjørven, J. A. Michaelsen and the involved partners of the QuA and MADAM projects, and acknowledge the partial funding of the EU for the MADAM project, no. 4169.

References

[1] M. Alia, G. Horn, F. Eliassen, M. U. Khan, R. Fricke, and R. Reichle. A component-based planning framework for adaptive systems. In *8th International Symposium on Distributed Objects and Applications (DOA)*. Springer Verlag, 2006.

[2] S. L. Amundsen, K. Lund, and F. Eliassen. Utilising alternative application configurations in context- and QoS-aware mobile middleware. In *6th International Conference on Distributed Applications and Interoperable Systems, Bologna, Italy*. Springer Verlag, 2006.

[3] S. Bowers, L. Delcambre, D. Maier, C. Cowan, P. Wagle, D. McNamee, A.-F. L. Meur, and H. Hinton. Applying adaptation spaces to support quality of service and survivability. In *DISCEX '00*, volume 2, pages 271–283, Jan. 2000.

[4] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, and J. B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(1112):1257–1284, 2006.

[5] J. T. Buck. Scheduling dynamic dataflow graphs with bounded memory. Technical report, Berkeley, CA, USA, 1993.

[6] L. Carpa, W. Emmerich, and al. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.

[7] S.-F. Chang and A. Vetro. Video Adaptation: Concepts, Technologies, and Open Issues. In *Proceedings of the IEEE*, volume 93, pages 148–158, January 2005.

[8] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama. A component model for building systems software. In *Proceedings of IASTED Software Engineering and Applications (SEA'04), Cambridge MA, USA*.

[9] J. Doyle and R. H. Thomason. Background to qualitative decision theory. *AI Magazine*, 20(2):55–68, 1999.

[10] V. S. W. Eide, F. Eliassen, and J. A. Michaelsen. Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming. In S. Chandra and N. Venkatasubramanian, editors, *Proceedings of the Twelfth Annual Multimedia Computing and Networking (MMCN '05), San Jose, California, USA*, volume 5680, pages 155–166, January 2005.

[11] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjørven. Beyond design time: using architecture models for runtime adaptability. *IEEE Software*, 2006.

[12] A.-C. Huang and P. Steenkiste. Building services using service-specific knowledge. In *proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing, July 2005*, 2006.

[13] H. Ma, I.-L. Yen, J. Zhou, and K. Cooper. Qos analysis for component-based embedded software: model and methodology. *J. Syst. Softw.*, 79(6):859–870, 2006.

[14] P.-G. Raverdy and R. Lea. DART: A distributed adaptive run-time. In *In IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, 1998.

[15] V. Poladian, J. Sousa, D. Garlan, and M. Shaw. Dynamic configuration of resource-aware services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, 2004.

[16] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the First International Conference on Autonomic Computing (ICAC'04*, 2004.