Leveraging State-Based User Preferences in Context-Aware Reconfigurations for Self-Adaptive Systems

Marco Mori[1], Fei Li[2], Christoph Dorn[3],
Paola Inverardi[4], and Schahram Dustdar[2]

[1] IMT Institute for Advanced Studies Lucca
marco.mori@imtlucca.it
[2] Distributed System Group, Vienna University of Technology
{li,dustdar}@infosys.tuwien.ac
[3] Institute for Software Research, University of California, Irvine
cdorn@uci.edu
[4] Dip. di Informatica, Università dell'Aquila
paola.inverardi@di.univaq.it

Abstract. Applications in ubiquitous environments need to adapt to a range of fluid factors, like user preferences, context, and various system configurations. In this paper, we address the problem of system adaptation in order to continuously achieve high user benefit while keeping reconfiguration costs low. To this end, the presented approach leverages not only the immediate context but also future transitions. In contrast to existing approaches that either maximize benefit or minimize reconfiguration costs, our proposed decision support mechanism achieves a trade-off between those factors. Considering user preferences, deployment constraints, and probabilistic context state transitions, we propose a multi-objective utility function to determine the best reconfiguration choices. Experimental results show that the proposed approach achieves high user benefit while keeping reconfigurations costs low.

1 Introduction

As ubiquitous computing is becoming more and more widespread, software engineers have to deal with different variability dimensions including the system context, users, and the system configuration itself. Changes are not always predictable since they are beyond the control of the system and they may require human intervention. To reduce maintenance cost it is desirable to achieve automatic self-adaptations in response to various kinds of changes. Adaptations should meet the desired quality requirements according to user preferences and they should be performed at reasonable cost and in a timely manner.

Self-adaptive systems are able to adjust their run-time behavior in face of changing external circumstances [17,4,1]. Software engineers define a set of alternative behaviors at design time while the actual adaptation decisions are postponed to run-time. Context plays a key role for adaptation since it determines

G. Barthe, A. Pardo, and G. Schneider (Eds.): SEFM 2011, LNCS 7041, pp. 286–301, 2011.

the variation of user preferences and the space for the admissible adaptation alternatives. Context, thus, needs to be explicitly modeled in order to account for the run-time adaptation as proposed in research of context-aware systems [3,10].

This paper addresses the problem of how to achieve simultaneous adaptation to system execution context and user preferences. The execution environment determines the space of admissible system configurations whereas the user determines the benefit of each available configuration. Switching between different configurations comes at some cost. Consequently, predictive information promises a significant cost saving potential by making adaptation decisions aware of probable future context changes and thereby anticipates upcoming reconfiguration needs. When determining system configurations, the challenge lies in finding a suitable trade-off between two opposite objective functions: maximize user benefit while minimize reconfiguration costs. Pure user benefit-driven selection comes with high costs due to frequent reconfigurations. In contrast, pure cost-driven adaption neglects user preferences and invariably prefers the current configuration, thus it changes the system configuration only when absolutely necessary. For balancing the two factors, we define our solution as a multi-criteria selection problem among different system alternatives and we evaluate each of them through an aggregating objective function that combines cost and user benefit. Experiments based on a case study and simulation demonstrate that our approach successfully determines Pareto-optimal configurations of high user benefit and low reconfiguration costs. Our contributions in this paper are:

- a concrete methodology to characterize system resources and configuration eligibility while
- defining user preferences on non-functional properties specific to distinct user contexts (situations).
- applying probabilistic information on future context states to predict upcoming context changes
- defining the configuration selection as a multi-criteria optimization problem.

The remainder of this paper is structured as follows: Section 2 presents a motivating scenario followed by a description of our approach in Section 3. Section 4 introduces the optimization framework in terms of their main components, while Section 5 formalizes the optimization problem. Section 6 provides evaluation and validation results based on our case study and a simulation. Section 7 discusses related work before conclusions and future work round up our paper in Section 8.

2 Motivating Scenario

An e-Health application supports doctors' activities by providing the most relevant services to visualize per-patient case history. Patient information is available at three levels of granularity: (i) a complete case history that includes textual reports and medical images, (ii) a compact version with only the recent history of reports and images, and (iii) only a textual case history. In addition images are displayed either as black and white images, in low color (256 colors), or as fully colored images (4096 colors).

Doctors need to receive aggregated per-patient information to support their activities at different locations. These activities include patient consulting, check-up, and medical procedures such as operations. Moreover they may be involved in emergency situations. These activities are performed at different locations such as common visiting rooms, surgery rooms, patient home or outside the hospital when an emergency arises. The doctor is able to visualize per-patient information through an accessible device inside or outside the hospital. Devices differ in their hardware resources such as bandwidth availability ($netB$), number of screen colors (SC), CPU speed ($CPUClockRate$) and available memory (Mem). Hardware has an impact on the available services: e.g., low bandwidth and 8 bit colors restrict the responsiveness to retrieve the patient's medical history and available image quality.

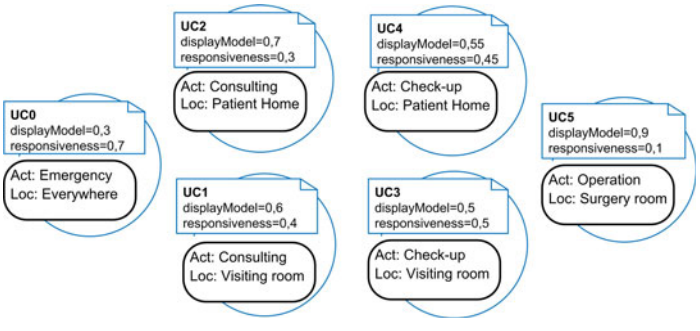


Fig. 1. User Preferences Example

Activity and location influence the doctor's preference for displaying the case history and image quality, see Fig. 1. The doctor might prefer a responsive system in case of an emergency activity. In another case, immediate retrieval of per-patient information is not as important as a detailed history for consulting activities. Upon context changes, the e-Health application needs reconfiguration based on the underlying hardware resources and the doctor's (context-dependent) preferences.

3 Approach

From a feature engineering perspective, features are the basic unit of behavior [6]. Breaking the system logic into feature components enables us to reduce the impact that any change might have on the system. Thus, we represent each alternative system variant as a distinct configuration of features. In [9] we defined a methodology to create the space of admissible configurations for a self-adaptive application starting from the set of basic features. In this paper we are going to define a decision making mechanism that is suitable for feature-based systems having the cost of deploying a feature independent from the running configuration. Figure 2 visualizes the conceptual aspects of our work. We consider for each

configuration a set of deployment constraints to assess the configuration admissibility. These constraints are evaluated against the current underlying context (resources) to establish whether the environment can support the execution of that particular configuration. On the other hand, we map a feature configuration to non-functional properties to represent the configuration's quality. This quality becomes a configuration utility (i.e., user benefit) when matched with user preferences. For each user, we assume the availability of historical transitions between the various context states. We also assume the time required for system adaptation upon a state transition to be negligible compared to the frequency of user context changes.

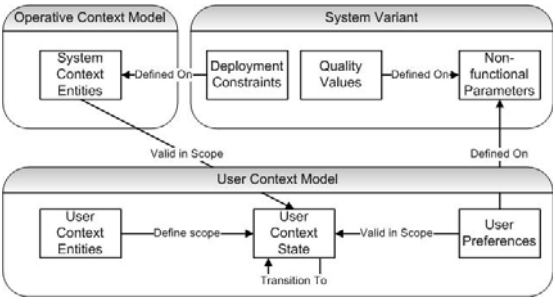


Fig. 2. Conceptual Model

4 Basic Models

In this section we introduce the basic definitions and models we use to formulate the context-aware reconfiguration problem. System reconfiguration aims at satisfying two objectives: user benefit and cost which arise due to system reconfiguration. We represent each *system variant* in terms of a *deployment constraint* and the provided *non functional properties*. We propose a definition of *transition cost* to penalize reconfigurations. In our framework, there are two types of context models. On one hand, features express their deployment constraint as conditions on the *operative context model*. In contrast, user preferences are not static but depend on the underlying *user context entities* instances. Thus, the *user context model* provides mean to map particular user preferences (over the *non-functional properties*) to specific context instances. We exploit the operative context model to evaluate which alternatives are admissible at a given point in time, and the user context model to deal with the current and probable future user preferences.

4.1 Operative Context Model

The system variants describe which resources they require for execution and thus need to be represented in the operative context model. Each system context entity is identified through a tag within the set $TagId = \{TagId_1, ..., TagId_n\}$ and it can assume one among its admissible values contained in the corresponding

finite domain $\mathbb{D}_1, ..., \mathbb{D}_n$. We adopt the modeling approach proposed in [13] to represent our definition of *operative context space* and *operative context scope*.

The *operative context space* for the system context entities is defined as the Cartesian product of their admissible values: $O = \bigotimes \mathbb{D}_i \;\; s.t. \;\; i = 1, ..., n$. Each element in O is a vector r expressing a different assignment of values, e.g. $r = (netB(100Kbps), Mem(10MB), SC(256colors), CPUClockRate(100Mhz))$.

An *operative context scope os* is a subset of the operative context space O, $os \in 2^O$, e.g. $os = (netB(100-200Kbps), Mem(10-50MB), SC(10-20colors), CPUClockRate(100-150Mhz))$.

4.2 System and System Variants

We have adopted the feature engineering perspective to express the system variants. Each basic unit of behavior is expressed as a feature, that is the smallest unit of behavior that can be perceived by the user [11]. System variants are expressed as configurations; each one obtained by combining subsets of features. We define each configuration in terms of deployment constraints and the fitness values for the non-functional properties.

Deployment Constraints. It is a predicate which expresses the demand to the operative context entities for a configuration c, e.g. $netB(5kbps) \wedge Mem(0, 1MB)$ is true only with a sufficient level of bandwidth and memory. An operative context scope os_c entails the elements in O that make the predicate true. Then, we evaluate if a configuration c is eligible in a context scope os with the function f_c which is equal to 1 only if $os \subseteq os_c$ and 0 otherwise. In our problem we also exploit the function $Eligible(r)$ to evaluate which configurations are eligible with the context values in r.

Non-functional Properties. Each configuration for an adaptive application provides different qualities to the user. These non-functional properties $NFP = \{nfp_1, nfp_2, ..., nfp_s\}$ can be quantitatively measured to drive the adaptation and to guarantee the user benefit. We map the properties values (defined over finite domains) to normalized fitness values in the real range [0,1]. For each configuration c, the vector fv_c contains the fitness values.

Transition Cost. An important factor to consider during the reconfiguration process is the penalty of switching from the source configuration to the target configuration. Since in our approach system configurations are made by features, we characterize this penalty based on the distance between the two configurations as $Dist_{y,z} = [NToDeploy \; NToUnDeploy]$ expressing the number of features to deploy and un-deploy switching from y to z. The vector $FCost = [CDeploying_f \; CUnDeploying_f]$ contains the same cost of deploying and un-deploying a feature. Based on the two vectors we define the transition cost of switching from y to z as:

$$TC(y, z) = (Dist_{y,z} \cdot FCost^T)/MaxCost \tag{1}$$

This cost is normalized to the maximum theoretical cost, which depends by the maximum number of features to deploy and un-deploy:

$$MaxCost = [MaxToDeploy\ MaxToUnDeploy] \cdot FCost^T \tag{2}$$

This simplified cost model is sufficient for our purpose since we do not address the problem of executing the actual system reconfiguration at the implementation level.

4.3 User Context Model

User context entities characterize the user's situation. As they are beyond the control of the application, they play a key role in the adaptation process. As mentioned in Section 2, the user's preferences change when switching from one user context state to another. Note that our approach is independent from the actual user context entities and how they change as long as there is a mapping of the various observable user context states to user preferences.

We define a mapping between the user context state UC — as defined by a set of user context entities — and the associated user preferences. User preferences express the importance (i.e., weight) of the various non-functional properties in a given context state. Higher weights express higher importance applied in the mapping functions $w : NFP \rightarrow [0, 1]$. Furthermore, we introduce a probabilistic automaton to represent the changing user preferences as induced by the underlying transitions between context states.

This automaton is defined as $A = (UC, P, E)$ where:

- $UC = \{UC_0, ..., UC_t\}$ is the set of states expressing the space of the user preferences. Each state is represented as a different combination of weights upon the non-functional parameters: $UC_j = [w_1(nfp_1)...w_s(nfp_s)]\ j = 1, .., t$; at each state the weights are defined as: $\sum_{i=1}^{s} w_i(nfp_i) = 1$
- P is the set of transition probabilities
- $E : UC \times P \rightarrow UC$ is the probabilistic transition function

This probabilistic state-based model shows how the preferences reflect the changes of user context entities. Historical data collected during system execution allows us to determine the actual transition probabilities between user context states. We continuously sample user context data at fixed intervals of time so that the probability to have two or more preference changes (i.e., context changes) within one interval is negligibly low. This process, however, is beyond the scope of this paper. Nevertheless techniques like [12] show the possibility to get preferences from user context, whereas methodologies like [14] define how to build a probabilistic model and maintain it updated with current system execution.

We expect that the various user context states come with changes in the operative context space. For example, bandwidth will not be the same in every location. Consequently, we consider also if a particular system variant is admissible in the observed user context state, independent from user preferences. We define

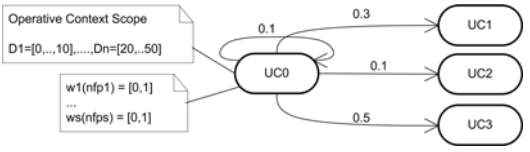


Fig. 3. Probabilistic automaton excerpt

a mapping function to associate each state in UC with an operative context scope within the set OS ($UCR : UC \rightarrow OS$). This models the correspondence between the user preferences and the observed system context entities. Fig. 3 provides an excerpt of a probabilistic automaton, detailing the mapping of user preferences and operative context scopes to a user context state.

5 Problem Formalization

Two events trigger the optimization problem and subsequent reconfiguration. Either the user moves into a new user context state characterized by a changing preference or the operative context cannot support the execution of the current system variant anymore. The best configuration to deploy depends on the achievable user benefit and the associated costs for reconfiguring the system. A strategy that maximizes the user benefit after each transition possibly requires many system reconfigurations. On the other hand choosing a fixed configuration which is always eligible throughout all states may result in possibly sub-optimal user benefit or may not exist at all. As a consequence we have to consider a trade-off analysis between two conflicting criteria, i.e. user benefit and reconfiguration costs. In the following we formalize the user benefit, the reconfiguration cost, and describe their combination in a single utility function.

As shown in Eq.3 the component B_{curr} evaluates how well a certain configuration c fits the current user context state. The user benefit at each state is the product of the corresponding user preferences vector with the quality attribute fv_c offered by the configuration.

$$B_{curr} = UC_{curr} \cdot fv_c^T \tag{3}$$

A configuration that gives optimal user benefit for a certain state may be sub-optimal if we consider the probable future states. Therefore we introduce an equation component that evaluates the expected user benefit in the future as given by the probabilistic context transitions. The cost component B_F shown in Eq.4 computes the future benefit of a configuration. We limit the calculation of future benefit to a single hop in the transition graph. Considering additional states (i.e., multiple hops) is expected to yield little additional benefit as each of the reachable states will have very small probability and thus hardly any impact.

$$B_F = \sum_{j=1}^{\#OutLink(UC_{curr})} p(UC_{curr}, UC_j) \cdot \left[UC_j \cdot fv_c^T\right] \cdot f_c(os_j) \tag{4}$$

B_F aggregates the user benefit for each subsequent user context state weighted according to the respective transition probability. A configuration yields user benefit only if it is eligible in the corresponding operative context scope ($f_c(os_j) = 1$). Ultimately, the overall user benefit equation is obtained by combining the current and future user benefits as follows:

$$B_{Agg} = h \cdot B_{curr} + (1 - h) \cdot B_F \tag{5}$$

The horizon h regulates the importance of the current user benefit compared to the future user benefit. The horizon close to 1 expresses a preference for the current state, whereas for h close to 0 we deem the future more relevant. Thus for environments where the user is expected to rapidly switch between states, the horizon configuration parameter should be closer to 0 as he/she will leave the current state soon.

The reconfiguration cost TC represents the cost of switching from the current deployed configuration to the configuration c (Eq. 1). The problem of selecting the best configuration in a operative context model state r, given a predefined user context model, is formalized as a max optimization problem combining the expressions defined in Eq. 3, 4, 1:

$$\max_{c \in Eligible(r)} \alpha \cdot [h \cdot B_{curr} + (1 - h) \cdot B_F] - (1 - \alpha) \cdot TC(c_{curr}, c) \tag{6}$$

The parameter α regulates the trade-off between user benefit and reconfiguration cost. Setting α closer to 1 makes the optimization more likely to meet the user benefit in spite of a high cost of reconfiguration. When setting this parameter closer to 0, we reduce the reconfiguration cost by selecting general purpose system variants that may be sub-optimal on the user benefit. The parameter h enables to tune the interest between the current user preferences and the probable future user preferences as explained above.

By introducing the variables α and h we make our optimization process customizable to various environments. The horizon h enables tuning to self-transitions in the context automata. If the resulting self-transitions are very high but still we are interested in optimizing future preferences we need to decrease the value of h. On the other hand if we end up with low self-transitions but we want to better match current preferences we have to increase the value of h. In addition, by setting $h = 1$ we enable comparison to existing approaches that are future-unaware.

6 Evaluation

This section presents two ways to evaluate our contribution. Firstly, we model the motivating scenario in Section 2, and apply our approach to find an optimal solution to the scenario. Secondly, we simulate the reconfiguration process at large scale in order to provide general guidelines of parameter settings to extensive application scenarios.

6.1 Case Study

Applying the feature engineering perspective the e-Health scenario yields following alternative features to view the per-patient case history: $S = \{f_{viewAllIm}, f_{viewLastIm}, f_{viewAllRep}, f_{viewLastRep}, f_{viewSum}, f_{paintBW}, f_{paintCol}, f_{paintFCol}\}$.

Table 1. System Configurations

Configuration	Deployment Constraint	Non-Functional Properties
$c_1 = \{f_{viewSum}\}$	$netB(5kbps) \wedge Mem(0,1MB)$	$displayModel = summary$ $responsiveness = high$
$c_2 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintBW}\}$	$netB(20kbps) \wedge Mem(2,5MB) \wedge$ $CPUClockRate(40Mhz)$	$displayModel = lastHistory$ $responsiveness = mediumHigh$
$c_3 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintCol}\}$	$netB(20kbps) \wedge Mem(2,5MB) \wedge$ $CPUClockRate(50Mhz)$	$displayModel = lastHistory$ $responsiveness = mediumHigh$
$c_4 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintFCol}\}$	$netB(200kbps) \wedge Mem(40MB) \wedge$ $CPUClockRate(10Mhz) \wedge SC(4096colors)$	$displayModel = lastHistory$ $responsiveness = mediumLow$
$c_5 = \{f_{viewAllIm}, f_{viewAllRep}, f_{paintBW}\}$	$netB(40kbps) \wedge Mem(10MB) \wedge$ $CPUClockRate(40Mhz)$	$displayModel = completeHistory$ $responsiveness = medium$
$c_6 = \{f_{viewAllIm}, f_{viewAllRep}, f_{paintCol}\}$	$netB(40kbps) \wedge Mem(10MB) \wedge$ $CPUClockRate(50Mhz) \wedge SC(256colors)$	$displayModel = completeHistory$ $responsiveness = medium$
$c_7 = \{f_{viewAllIm}, f_{viewAllRep}, f_{paintFCol}\}$	$netB(800kbps) \wedge Mem(160MB) \wedge$ $CPUClockRate(100Mhz) \wedge SC(4096colors)$	$displayModel = completeHistory$ $responsiveness = low$

Table 1 lists the 7 configurations built from these proposed features. Configurations c_1 provides only a textual representation of the patient's case history ($summary$). The next three configurations display only the very recent case entries ($lastHistory$) by means of textual reports and medical images which may be colored following the three different modes (BW, $Fullycolored$, $Colored$). The last three configurations display the complete case history ($completeHistory$) with a different coloring modality. The feature combination in each configuration determines the responsiveness level (ranging from Low to $High$). In the following we describe how the reconfiguration process takes place whenever the user switches context. We potentially observe a change of the user preferences when the doctor moves to a different location or engages in a different task. If this is the case, we then have to evaluate which configuration maximizes Eq. 6. We select the best configuration starting from the following inputs: the set of eligible configurations in the current operative context, the user context automata and the reconfiguration costs. In this case study, we obtain a user context automaton with the transitions probabilities shown in Fig.4 by analyzing historical user data detailing the movements and the doctor's working timetable. Each state is characterized by different weights for each quality attribute (UC_0 = [0.3 0.7], UC_1 = [0.6 0.4], UC_2 = [0.7 0.3], UC_3 = [0.5 0.5], UC_4 = [0.45 0.55], UC_5 = [0.9 0.1]). The first component of each vector indicates how important the *displayModel* property is, while the second expresses the weight for *responsiveness*. In addition each user context state is associated to a different operative context scope $OS_0, .., OS_5$.

Suppose the doctor changes from an emergency activity to a check-up activity within the hospital visiting room. As a consequence the user context switches from UC_0 to UC_3 and the reconfiguration process commences. Note that the user is free to switch between context states which exhibit no corresponding transition in the automaton. Let us suppose that the running configuration is c_2 and the

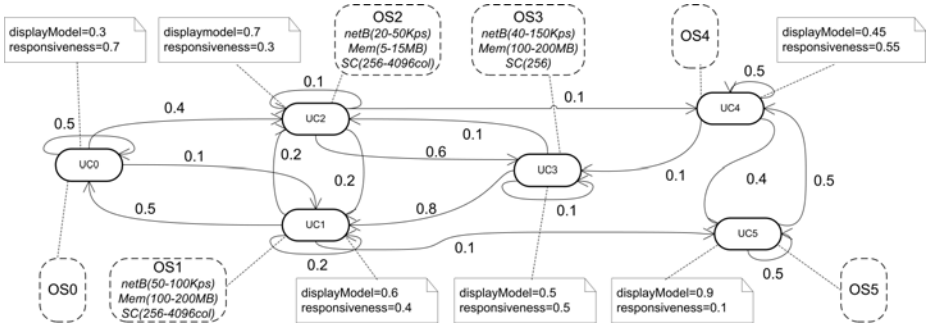


Fig. 4. User Context Automata

operative context state is $r_{curr} = (netB(50Kbps), CPUClockRate(100Mhz), Mem(20MB), SC(256colors))$. We check the deployment constraints for the configurations in Table 1 against the state r_{curr}. Thus we compute the set of eligible configuration as $Eligible(r_{curr}) = \{c_1, c_2, c_3, c_5, c_6\}$. Each configuration provides two non-functional properties $NFP = \{displayModel, responsiveness\}$. The first assumes one value among *summary*, *lastHistory* and *completeHistory* whereas the second assumes one value among *low*, *mediumLow*, *mediumHigh*, *medium* and *high*. Starting from the qualities offered by each configuration we evaluate the parallel fitness vectors exploiting a possible normalization:

$$\begin{aligned}
c1: &\ [summary\ high] &&\Rightarrow fv_{c1} = [0.1\ 0.8]\\
c2: &\ [lastHistory\ mediumHigh] &&\Rightarrow fv_{c2} = [0.5\ 0.65]\\
c3: &\ [lastHistory\ mediumHigh] &&\Rightarrow fv_{c3} = [0.5\ 0.65]\\
c5: &\ [completeHistory\ medium] &&\Rightarrow fv_{c5} = [0.9\ 0.5]\\
c6: &\ [completeHistory\ medium] &&\Rightarrow fv_{c6} = [0.9\ 0.5]
\end{aligned}$$

For purpose of demonstrating, we assume the cost of deploying and un-deploying any feature is $FCost = [2\ 1]$. The distance between each admissible configuration and the current one ($c_2 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintBW}\}$) in terms of features to deploy and un-deploy is given in Table 2. The normalized costs of switching from the current configuration to each possible target one are: $TC(c_2, c_1) = 0.555$, $TC(c_2, c_2) = 0$, $TC(c_2, c_3) = 0.333$, $TC(c_2, c_5) = 0.667$, $TC(c_2, c_6) = 1$. The maximum theoretical cost we exploit for the normalization is evaluated as $MaxCost = [3\ 3] \cdot [1\ 1]^T = 6$ (Eq. 2). We solve the optimization problem at Eq.6 considering the new user context state UC_3, the set of eligible configurations at the operative state r_{curr}, and the costs. For demonstrating our approach we set α to 0.7 to express that the user benefit is more important than

Table 2. Distance evaluation

Dist./Conf.	c_1	c_2	c_3	c_5	c_6
ToDeploy	1	0	1	2	3
ToUnDeploy	3	0	1	2	3

costs. We also set the variable h to 0.5 to consider equally current and future user preferences.

Our proposed methodology enables selecting the configuration which fits better the current preferences while considering the future user preferences. Future preferences are determined by the probable future task and location in which the doctor will be involved. In addition also the costs of switching configuration are taken into account.

At the current user context state ($UC3$, the one where the user just arrived), the doctor is performing a check-up activity at the visiting room where the $responsiveness$ and $displayModel$ properties are equally ranked (Figure 1). By looking at the automata in Fig.4 we reason that with very high probability the doctor will thereafter switch to another state (UC_1). This probable subsequent state comes with different weights for $responsiveness$ and $displayModel$ (UC_1). As a consequence we anticipate this future transition by selecting a system configuration which provides already better display modality now, even if it does not strictly meet the current user preferences. Nevertheless, in this example the top ranked configuration maximizes also the current preferences.

Table 3 presents the overall utility value for the eligible configurations obtained by combining the user benefit component (Eq.5) and the cost (Eq. 1). User benefit components do not need normalization since they are evaluated exploiting normalized user preferences and normalized quality vector. In this illustrative example the best configuration is c_6 since it corresponds to the best trade-off between user benefit and costs with given h and α.

Table 3. Configurations evaluation

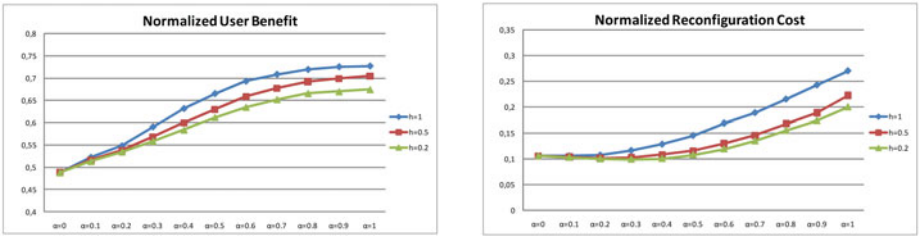
Configuration	B_{curr}	B_F	Cost	**Overall utility**
$c_1 = \{f_{viewSum}\}$	0.45	0.38	0.555	0,457
$c_2 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintBW}\}$	0.575	0.56	0	0,397
$c_3 = \{f_{viewLastIm}, f_{viewLastRep}, f_{paintCol}\}$	0.575	0.56	0.333	0,497
$c_5 = \{f_{viewAllIm}, f_{viewAllRep}, f_{paintBW}\}$	0.7	0,662	0.667	0,6767
$c_6 = \{f_{viewAllIm}, f_{viewAllRep}, f_{paintCol}\}$	0.7	0,662	1	**0,7767**

6.2 Experiment

Besides a case study, we validate our approach by simulating context changes and the resulting reconfigurations for various parameter settings. The results demonstrate that a predictive approach considerably improves the reconfiguration process. The simulation process takes the user context automata, costs, and a set of system variants as input. During the simulation we measure two metrics: the achieved user benefit and the incurred reconfiguration costs.

We run the same experiment with different values for the parameters α and h to analyze the effect on the two metrics. For each experiment we construct a set of 200 paths of 100 hops generated according to the probabilities of a fixed user context automata (Sec. 6). We then generate a fixed number of alternatives system variants. For each variant we define randomly the eligible context states and the values of non functional properties. Each experiment consists of iterating through the context automaton according to the 200 predefined paths. At

each state, we select the variant that maximizes Eq. 6. For each chosen variant, we log the current user benefit and reconfiguration cost. Finally, we evaluate the averages of the two metrics over all paths within a single experiment configuration. Then we repeat the experiment with the same paths sequences but varying α and h values. We then compare the results for different combinations of α and h. Setting the horizon h to 1 we simulate a future unaware reconfiguration strategy. There was no difference in the resulting cost and benefit trends for cost vectors $FCost = [2\ 1]$ and $FCost = [10\ 1]$; thus we report only the results for the former.



(a) (b)

Fig. 5. Normalized average user benefit (a) and normalized average reconfiguration cost (b) with $h = 1.0$, $h = 0.5$ and $h = 0.2$ depending on utility objectives weights α.

Figure 5(a) and 5(b) show the normalized user benefit and reconfiguration across 33 experiment configurations. Figure 5(b) compares reconfiguration cost with three different values of h. Here we observe higher reconfiguration cost if we consider only the current user context state ($h = 1$). On the other hand configurations are likely to change less frequently whenever we consider future user preferences. This holds if we consider current and future preferences equally ($h = 0.5$) as well as if we give more relevance to the future state ($h = 0.2$). We can conclude that looking into the future lowers the cost. As shown in the Figure we can reduce the reconfiguration cost by regulating h independent from α. Since α represents the weight for the aggregated user benefit (Eq. 5), it increases the significance of user benefit over the cost when it is close to 1.

Although we can reduce the reconfiguration cost by exploiting future user preferences, we potentially lower user benefit at the same time. A configuration that optimizes both current and future preferences does not necessarily maximize current user benefit. Figure 5(a) presents the difference of user benefit considering the static and predictive approach with value of h. We get the best average user benefit if we consider only the current user context state ($h = 1$) while we get lower values if we consider the future user preferences ($h = 0.2$ and $h = 0.5$).

Figure 5(a) and 5(b) suggest that if we consider the current and future user preferences the relative decrease in user benefit is smaller than the reduction of reconfiguration cost. As shown in the figures there is potential user benefit without raising the cost. In addition, Figure 5(a) and 5(b) also suggest that

within the user benefit component the parameter h regulates the benefit and cost objectives. In fact setting h closer to 1 increases the cost of reconfiguration in order to increment the benefit, whereas by setting h closer to 0 we partially alleviate the cost of reconfiguration by accepting lower user benefit configurations. The difference between α and h is that the horizon has a lower impact on the objectives compared to α.

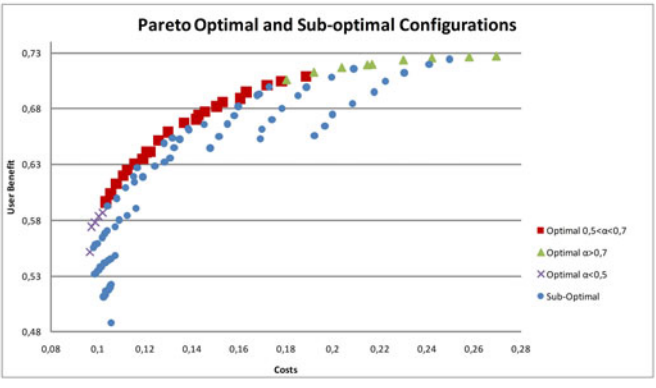


Fig. 6. Pareto-optimal and sub-optimal Configurations

Finally, we analyze the set of Pareto optimal configurations for $h = [0; 0.1; 0.2; \ldots 1]$ and $\alpha = [0; 0.1; 0.2; \ldots 1]$ for a total of 121 compared configurations. Pareto optimal points are roughly evenly distributed across h thus making it possible to select desirable values according to the specific application. We have discovered which range of α could be exploited to get most of the optimal points. In Figure 6, the Pareto optimal configurations are displayed following three different series; red squares stand for points in the range of $\alpha = [0.5; 0.7]$, crosses for optimal points for $\alpha = [0; 0.5]$ and triangles for $\alpha = [0.7; 1]$. Sub-optimal configurations are given in blue circles. As the configuration values are averaged over multiple transitions (as outlined above) also sub-optimal configurations close to the Pareto-optimal ones might me candidates. As shown in the Figure we have noted that around 50% of optimal configurations lie in the range of $\alpha = [0.5; 0.7]$. We can thus conclude that too low α values put too much weight on costs and therefore waste a lot of potential to improve user benefit. Hence, our approach is able to realize considerable user benefit even in very cost-constrained environments. Pareto optimal configurations as shown in the Figure help to decide how to set α while leaving to the designers the choice of h for specific ubiquitous applications.

The results demonstrate that predictive approaches ($h < 1$) allow the reduction of reconfiguration cost while providing an acceptable level of benefit to the user. We can conclude that our predictive approach is as good as non predictive approaches ($h = 1$) whenever we want to maximize the user benefit without focusing too much on cost. For cost-sensitive environments, a non-predictive approach fails to produce Pareto optimal points. Indeed, Pareto optimal points with $h = 1$ have high α values ($\alpha = [0.7; 1]$).

7 Related Work

Self-adaptive systems need automatic reconfiguration at run-time considering the characteristics of execution environments and the user preferences. Since it is important to make adaptation resilient to changes, it is necessary to support the decision-making process with predictive information.

In the literature there are a number of decision making mechanisms exploiting user preference to support the adaptation. Sykes et al. [18] evaluate the utility of each system component primarily by the user preferences upon each non functional property. Then the overall utility degree for each system variant is obtained as the average of each component utility. The authors define the space of adaptation strategies without considering the environment condition explicitly. The PLASTIC approach [2] considers how to exploit user preferences in performing service based adaptation. The approach performs a non-functional selection among the system variants that can be deployed in the current execution environment based on the required resources. The approach proposes a resource model that is similar to our operative context scope since it supports the definition of eligible configurations. However no predictive information is included to drive their adaptation process. In the field of service discovery, Li et al. [13] exploit a user preference model to support the service recommendation to the user. At run-time, services are checked with respect to their precondition and then they are ranked based on the user preferences upon their possible outcomes. The approach considers only a simple context model without considering future changes. Dorn and Dustdar [7] observe the behavior of multiple users to adapt the available software capabilities (i.e. features) to the preferences of the whole group. Their approach, however, does not take into account operative context constraints, neither do they apply predictive knowledge on potential future context changes.

All the mentioned approaches neither consider predictive information about the context resources nor about the user preferences. Adaptations are performed only by exploiting information on the current context.

Cheng et al. [5] extend the Rainbow evolution framework [8] in order to exploit the predictive availability of context resources to enable the adaptations. However they lack the notion of user preferences. Poladian et al. [15,16] face the problem of selecting a sequence of system variants for a predefined sequence of fixed time slots, each of which is characterized by a prediction of resource availabilities. The sequence which better fits the fixed user preferences at each time slot is selected. Also a factor of cost is introduced in order to give an increased utility to components which are already running. In addition to this work, which is heavily focused on resource prediction, we also consider the predictive context states and corresponding user preferences because of our intention to address ubiquitous environments. To the best of our knowledge there are no approaches that support system adaptation by considering run-time user preference changes, operative context changes and cost factors coherently in one formal framework. We claim that considering all these factors together promote better performance of the adaptation process.

8 Conclusions

In this paper we proposed a reconfiguration scheme for ubiquitous applications. In our approach we considered several factors including user preferences, non-functional properties, and reconfiguration cost, which may affect adaptation decisions in response to changing context. By applying feature engineering and context-awareness techniques, we quantified these factors and their aggregated effects in order to provide decision support in the face of multiple adaptation options. We conducted a series of experiments to evaluate the effectiveness of our approach with different configuration parameters. As the analysis of Pareto optimal solutions showed, our mechanism is able to maintain high user benefit while significantly reducing reconfiguration costs. Results further demonstrated that even a trade-off favoring reconfiguration costs over user benefits proves more effective than simply focusing on reducing costs alone. Based on these results, we provided guidelines to users on how to apply different parameters in order to weigh between user benefits and reconfiguration costs. The complete methodology was illustrated by means of a case study in the e-Health domain.

In the future, we will measure actual reconfiguration costs coming from the implemented case study. We also plan to integrate support for directly determining Pareto optimal solutions, thus relieving users of having to select suitable values for α and h. Furthermore, we will conduct theoretical analysis of the context state topology such as number of states, average connectivity between the states, and transition frequency, by which we expect to open up more intelligent and effective reconfiguration strategies.

Acknowledgments. This work has been partially supported by the EU IST CONNECT (http://connect-forever.eu/) No 231167 of the FET - FP7 program, the EU IST CHOReOS (http://www.choreos.eu/) No 257178 of the FP7 program, and the Austrian Science Fund (FWF) J3068-N23.

References

1. Andersson, J., de Lemos, R., Malek, S., Weyns, D.: Modeling dimensions of self-adaptive software systems. In: SEAMS, pp. 27–47 (2009)
2. Autili, M., Di Benedetto, P., Inverardi, P.: Context-aware adaptive services: The PLASTIC approach. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 124–139. Springer, Heidelberg (2009)
3. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. IJAHUC 2(4), 263–277 (2007)
4. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.): Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525. Springer, Heidelberg (2009)
5. Cheng, S.-W., Poladian, V., Garlan, D., Schmerl, B.R.: Improving architecture-based self-adaptation through resource prediction. In: SEAMS, pp. 71–88 (2009)
6. Classen, A., Heymans, P., Schobbens, P.-Y.: What's in a feature: A requirements engineering perspective. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 16–30. Springer, Heidelberg (2008)

7. Dorn, C., Dustdar, S.: Interaction-driven self-adaptation of service ensembles. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 393–408. Springer, Heidelberg (2010)
8. Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B.R., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer 37(10), 46–54 (2004)
9. Inverardi, P., Mori, M.: Feature oriented evolutions for context-aware adaptive systems. In: EVOL/IWPSE, pp. 93–97 (2010)
10. Kapitsaki, G.M., Prezerakos, G.N., Tselikas, N.D., Venieris, I.S.: Context-aware service engineering: A survey. JSS 82(8) (2009)
11. Keck, D., Kuehn, P.: The feature and service interaction problem in telecommunications systems: a survey. In: IEEE TSE (1998)
12. Krause, A., Smailagic, A., Siewiorek, D.P.: Context-aware mobile computing: Learning context-dependent personal preferences from a wearable sensor array. IEEE Trans. Mob. Comput. 5(2), 113–127 (2006)
13. Li, F., Rasch, K., Truong, H., Ayani, R., Dustdar, S.: Proactive service discovery in pervasive environments. In: ICPS, pp. 126–133 (2010)
14. Maia, P.H.M., Kramer, J., Uchitel, S., Mendonça, N.C.: Towards accurate probabilistic models using state refinement. In: ESEC/FSE, pp. 281–284 (2009)
15. Poladian, V., Garlan, D., Shaw, M., Satyanarayanan, M., Schmerl, B.R., Sousa, J.P.: Leveraging resource prediction for anticipatory dynamic configuration. In: SASO, pp. 214–223 (2007)
16. Poladian, V., Sousa, J.P., Garlan, D., Shaw, M.: Dynamic configuration of resource-aware services. In: ICSE, pp. 604–613 (2004)
17. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. TAAS 4(2) (2009)
18. Sykes, D., Heaven, W., Magee, J., Kramer, J.: Exploiting non-functional preferences in architectural adaptation for self-managed systems. In: SAC, pp. 431–438 (2010)