Noname manuscript No.
(will be inserted by the editor)

Personalized Adaptation in Pervasive Systems via Non-Functional Requirements

Estefanía Serral · Paolo Sernani · Fabiano Dalpiaz

Received: date / Accepted: date

Abstract Pervasive environments are socio-technical systems that support the daily routines of their users in an invisible and unobtrusive manner. These systems are aware of and adapt to both the operational context and the characteristics and preferences of their users. Designing adaptation mechanisms that guarantee maximal user satisfaction is challenging, due to the inherent differences between users and the changing context where the system operates.

In order to tackle this problem, we propose an approach that compares alternative system behaviors in terms of how well they satisfy the preferences of the current user concerning Non-Functional Requirements (NFRs) such as efficiency, comfort, energy saving, etc. Specifically, we propose a model-driven framework in which the models represent the user routines that the pervasive system helps to achieve. These routines include variability points, thereby enabling their behavior to be adapted at runtime in order to fit the context and the user preferences over NFRs.

Our contributions include: (i) user-adaptive task models, a modeling language to describe user routines that accounts for user preferences over NFRs; (ii) algorithms that use our models at runtime to guide a pervasive system in adapting its behavior to user preferences and context; and (iii) an implementation and evaluation of our techniques.

Estefanía Serral
Department of Decision Sciences and Information Management
Faculty of Economics and Business, KU Leuven
Naamsestraat 69, B-3000 Leuven, Belgium. E-mail: estefania.serralasensio@kuleuven.be

Paolo Sernani
Department of Information Engineering
Università Politecnica delle Marche
Via Brecce Bianche 12, 60131 Ancona, Italy E-mail: p.sernani@univpm.it

Fabiano Dalpiaz
Department of Information and Computing Sciences
Faculty of Science, Utrecht University
Princetonplein 5, De Uithof, 3584CC Utrecht, the Netherlands. E-mail: f.dalpiaz@uu.nl

1 Introduction

Pervasive computing (Satyanarayanan, 2001) is a computing paradigm in which technical systems are developed and deployed in the environment to support humans in their daily routines. The interplay of the technical system and the surrounding environment is called *pervasive (computing) system* (Henricksen et al, 2002). To maximize user satisfaction and system acceptance, the system has to remain as invisible and unobtrusive as possible and adapt to its users.

In the Ambient Intelligence (AmI) domain, pervasive computing aims at building unobtrusive, interconnected, adaptable, dynamic, embedded and intelligent environments around people; environments such as smart homes (Sadri, 2011). The pervasive computing paradigm is applied to enrich environments with smart capabilities, for example by providing multimedia and comfort functions and optimizing energy usage at home (Henniger et al, 2017). Despite the manifest goal is to support humans in their daily routine, the scientific literature highlights that current systems are tailored more on the available technologies, rather than supporting the end-users' needs (Calvaresi et al, 2017).

Task models (Paternò, 2002) are a state-of-the-art modeling language for representing and supporting user routines (Serral et al, 2010b). Task models are executable models: they are a hierarchical specification of the tasks that a pervasive system should execute in order to support a user in conducting her daily routines. Task models have been successfully adopted in model-driven pervasive infrastructures (Serral et al, 2010b, 2013b).

However, these models provide limited adaptation to user preferences, a key requirement to ensure the system stays unobtrusive and invisible. For example, a user who gives high importance to energy saving should be supported by minimizing heating usage and by suggesting eco-friendly transportation means. On the other hand, if that user has an urgent meeting and she is late, the system should recommend the most efficient transportation means.

This paper deals with the problem of how to account for user preferences by using non-functional requirements (NFRs) (Mylopoulos et al, 1992). NFRs have been successfully used in software engineering to inform trade-off analysis between alternatives in requirements models (Yu, 1995), architectures (Chung et al, 1995), and business processes (Pavlovski and Zou, 2008). Our challenge is to combine NFRs with task models and support the execution of user routines that maximize fitness with user preferences.

We extend task models and propose a model-driven framework for developing pervasive systems that can adapt their behavior to the user preferences and to the current context. Our contributions are as follows:

– *User-adaptive task models*, a modeling language that enriches task models with NFRs. The model is created at design-time by analysts and end users, and is used by the system at runtime to decide upon how to adapt its behavior. Individual user preferences over NFRs are captured by our proposed *contextual preference model*;

– *Adaptation algorithms* that use our models at runtime to let a pervasive system exhibit user-centered adaptation;
– *An implementation and evaluation of our approach.* We extend an existing middleware for automating user routines in pervasive systems, report on its scalability analysis, and discuss the benefits of our approach.

This paper is an extended version of our previous work (Dalpiaz et al, 2012) in which we presented an initial modeling approach. Compared to that paper, the present one adds fully-fledged adaptation algorithms that use our models at runtime, and an implementation and evaluation of our work.

The rest of the paper is organized as follows. Section 2 presents our baseline: task models. Section 3 introduces user-adaptive task models. Section 4 explains our model-driven algorithms for the pervasive system to adapt at runtime. Section 5 presents our implementation and evaluation. Section 6 discusses related work, while Section 7 outlines conclusions and future directions.

2 Research Baseline: Task Models

We build on (context-adaptive) task models (Serral et al, 2010b, 2013b), which specify how a pervasive system can support the daily activities of its users. Task models are inspired by Hierarchical Task Analysis (HTA) (Shepherd, 2001), tree structures that refine high-level tasks into a set of executable ones.

Illustrative domain: smart homes. A smart home pervasive system supports the daily routines of home inhabitants. The smart home is equipped with Ambient Intelligence devices, controlled by pervasive services (Serral et al, 2010b), that monitor and collect context information, as well as enable interaction with the users and the environment. One of the supported routines from previous work (Serral, 2011) is described as follows:

> "Every working day, the system turns on the bathroom heating at 7:50 a.m. to make it warm enough for Sarah to take a shower. At 8:00 a.m., the system makes a wake-up call, repeating it until Sarah wakes up. Then, the room is illuminated and Sarah is notified about the weather. Afterwards, when Sarah enters the kitchen, the system makes a coffee, and suggests her the best way to go to work."

Following HTA, every user routine is described as a hierarchy of tasks. Figure 1 illustrates the "Waking Up" routine. The root task represents the routine as a whole, and is iteratively broken down into simpler tasks by means of two task refinement constructs: *exclusive refinement* and *temporal refinement*. Exclusive refinement (represented by a solid line) decomposes a task into a set of subtasks in such a way that exactly one subtask will be executed. Temporal refinement (represented by a dashed line) also decomposes a task into subtasks; however, all the subtasks shall be performed following the order that is graphically depicted by the arrows between sibling tasks. Temporal constraints make use of Concurrent Task Trees (CTT) operators (Paternò, 2002). For example, in Figure 1, we use:

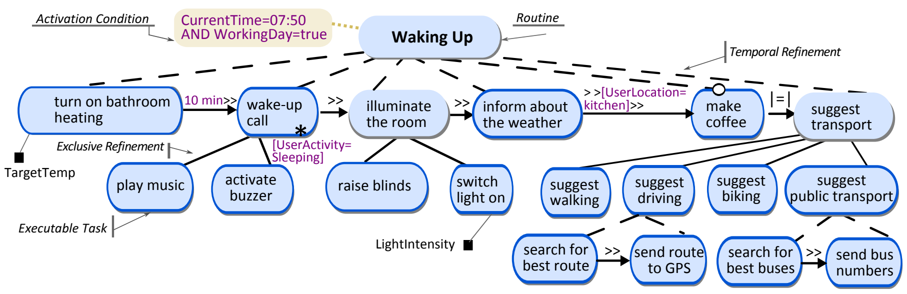

Fig. 1 Task model representing the "Waking Up" user routine

– *Enablement* ($T_1 \gg T_2$): task T_2 is triggered when task T_1 finishes. For example, the system has to illuminate the room after waking up the user;
– *Task Independence* (T_1 |=| T_2): T_1 and T_2 can be performed in any order. For instance, tasks "make coffee" and "suggest transport" are temporally independent.

The task refinement process terminates when every leaf task in the tree is associated with a pervasive service (controlled by the pervasive system), which is responsible for executing the task. For example, task "raise blinds" is executed through a pervasive service that controls the blinds engine.

A routine can be carried out through alternative sets of tasks depending on the current state of the context (the "situation" (Henricksen and Indulska, 2004) at hand). Situations are used to indicate the relationship between context and routine execution (see Figure 1):

– *Activation condition*: it is associated with the root task of each routine, and indicates when the routine gets activated. For instance, the "Waking Up" routine is triggered every working day at 7:50 a.m.
– *Task precondition*: it can be associated with a task to indicate that its execution depends on whether a situation holds. This condition is not expressed graphically in order to keep the diagrams easy to read. For instance, one may specify that the system has to turn on the bathroom heating only if the difference between comfort and current temperature is greater than three degrees.
– *Iterative task*: it is executed repeatedly while the situation associated with the task holds. The iteration stops as soon as the situation ceases to hold. Iterative tasks are graphically marked with an asterisk. For instance, task "wake-up call" is iterated while the user sleeps, and the iteration stops as soon as the user wakes up.
– *Temporal constraints:* the following relationships indicate that the execution of two tasks (linked by an arrow) is subject to a temporal constraint:
  – $T_1 \gg [s] \gg T_2$: after the completion of T_1, T_2 is started as soon as situation s holds. In Figure 1, the system makes coffee after informing about the weather, as soon as the user is in the kitchen (situation

"UserLocation=kitchen" holds). Note that s could already hold when T_1 ends; in such case, T_2 starts immediately.
- $T_1\ t \gg T_2$: after the completion of T_1, T_2 is started as soon as the time period t has elapsed. For instance, 10 minutes after turning the bathroom heating on, the system shall execute task "wake-up call".

Once the routines have been specified by a system analyst, they can be directly executed by MAtE, a Model-based user task Automation Engine (see Section 5 for more details).

3 User-Adaptive Task Models

We employ the facets of context suggested by Sutcliffe et al (2005)—personal, physical, social—to extend task models by describing a system behavior that take into account contextual factors. We call them *user-adaptive task models* to highlight their capability to adapt to the user preferences at hand.

User-adaptive task models represent both the *personal context*, i.e., the individual requirements and preferences of specific users, and the *physical and social context*, i.e., observable characteristics of the environment that the system can monitor (e.g., who is in the room, temperature, closed doors).

Our extension enables not only adaptation to the preferences of different users, but also to the changing preferences of a specific user depending on the current situation. For instance, if a user is in a hurry, she may favor efficiency over comfort; when not at home, instead, she may be more interested in energy saving. Another user, instead, may always give comfort the highest priority.

Figure 2 depicts the meta-model of our proposed user-adaptive task models. The orange-coloured classes on the right show our proposed contextual preference model (Section 3.1), which indicates user preferences over NFRs. The white-coloured classes represent the extended task model itself: we introduce optional and parametric tasks (Section 3.2), as well as task contributions to NFRs (Section 3.3). The green-coloured classes (at the bottom) represent the context model from our previous work (Serral et al, 2010b).

3.1 Contextual Preference Model for NFRs

Each user has different preferences, which depend on the situation and may vary over time. To represent user preferences over non-functional aspects of the pervasive system, we propose the contextual preference model inspired by the one proposed by Henricksen and Indulska (2004). Using the contextual preference model, analysts define the relevant NFRs and the priority each user assigns to them in different contexts.

For each NFR, a set of couples consisting of a situation s and a priority p (a real number in the range [0,1]) is specified. Each couple indicates that, when situation s holds, the NFR has priority p. Value 0 indicates minimum importance, while value 1 indicates maximum importance. The situations for a

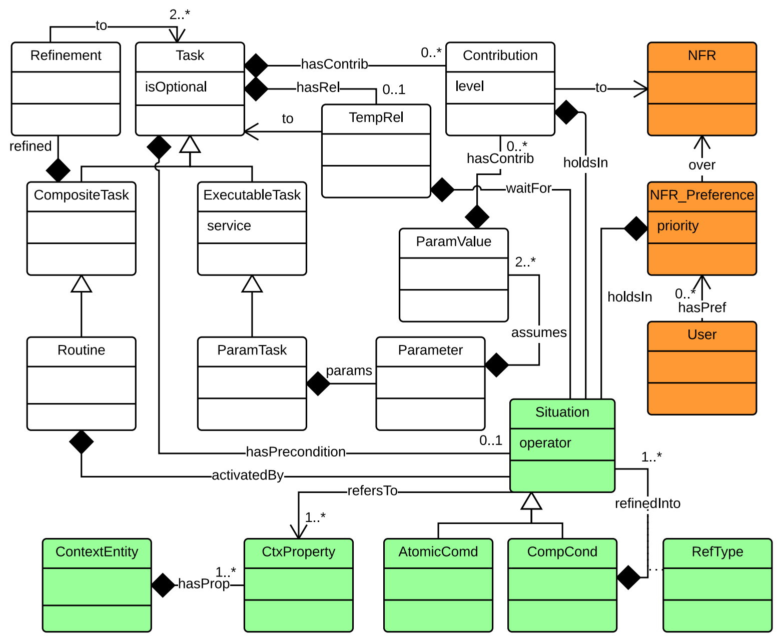

Fig. 2 Overview of the Metamodel for user-adaptive task models

Table 1 Partial contextual preference model for user Sarah

User Comfort (UC) :	⟨UserLocation≠Home, 0⟩
	⟨UserLocation=Home ∧ UrgentTasks=false, 0.7⟩
	⟨UserLocation=Home ∧ UrgentTasks=true, 0.5⟩
User Efficiency (UE) :	⟨UserLocation≠Home, 0⟩
	⟨UserLocation=Home ∧ UrgentTasks=true, 1⟩
	⟨UserLocation=Home ∧ UrgentTasks=false, 0.3⟩
Energy Efficiency (EE) :	⟨UserLocation≠Home, 0.9⟩
	⟨UserLocation=Home, 0.4⟩

given NFR have to be mutually exclusive. Table 1 shows part of the contextual preference model for Sarah. The priority of NFR user comfort is 0 when she is not at home; if Sarah is at home, the priority increases to 0.7 if she has no urgent tasks to do, 0.5 otherwise.

In order to facilitate the elicitation of this model, we have developed a novel contextual requirements prioritization method, which is explained in detail in (Serral et al, 2017). The method provides a simple web questionnaire[1] a user can directly fill in to express her contextual preferences over NFRs, or that an analyst can show to a given user through a tablet device. The number of questions that the user must answer depends on the number of NFRs and the contextual parameters that are relevant for the user for each NFR. For 3

[1] Online questionnaire for gathering contextual preferences over NFRs of a smart home: https://goo.gl/ir65zM

NFRs and a maximum of 2 contextual parameters per NFR, as in the running example, the questionnaire takes less than 15 minutes. Taking into account that our target audience are people who are going to spend an extended period of time (normally years) in a smart home, spending such little time filling a questionnaire is not critical, and the return on investment is highly beneficial: adjusting the home's behavior to reflect their preferences rather than those of a generic person.

The questionnaire minimizes inconsistency in the expressed preferences in two ways. First, the questions are asked in a top-down manner, i.e., the NFRs are first prioritized in a context-agnostic manner using the well-established Analytical Hierarchy Process (AHP) (Saaty, 1990), and the subsequent questions refine the weights to accommodate the extent to which specific contexts affect the relative priority of the NFRs. Second, the AHP process outputs also a consistency ratio: when this is too low, the user is shown the judgments that led to the inconsistency, and is then required to reconsider them.

3.2 Task Models with Optional and Parametric Tasks

Our adaptation approach relies on models and non-functional requirements. The pervasive system chooses an adequate course of action from its task model, on the basis of contextual factors and user preferences over NFRs. To such extent, we extend task models with optional and parametric tasks.

Optional tasks are not essential to accomplish a routine. These tasks are based on the optional tasks proposed in CTT for user interface design. In our approach, the execution of optional tasks depends on user preferences over NFRs. For instance, task "make coffee" is optional. In a situation where the user has a scheduled meeting early in the morning, the NFR user efficiency will have a high priority in the contextual preference model for the user at hand. Consequently, task "make coffee" will not be in the plan to carry out the waking up routine, as its execution would contribute negatively to user efficiency. Graphically (see Figure 3), optional tasks are represented by drawing a hollow circle to the incoming refinement link (like optional features in feature models, see Griss et al (1998)).

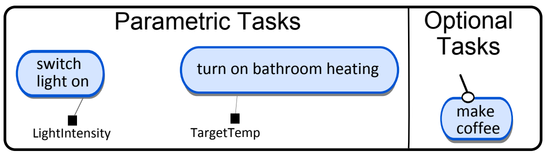

Fig. 3 Graphical representation of parametric and optional tasks

Parametric tasks are leaf tasks in the task model whose execution can be tuned by adjusting the values of a specified set of parameters. This tuning is intended to satisfy user preferences as much as possible. For instance, task "turn on bathroom heating" can be tuned by adjusting the target temperature, while task "switch bedroom light on" can be tuned by setting light intensity. Depending on the value of their parameters, the aforementioned tasks will have different impacts on NFRs such as energy efficiency and user comfort. Every parametric task should have at least two parameter values, specifying the minimum and maximum values that the task can assume. Graphically (see Figure 3), parameters are labels—representing the parameter name—connected to leaf tasks by a dotted line ending with a square-shaped arrow.

3.3 Linking Tasks to NFRs via Contributions

In a model-driven system, variation points are the enablers of adaptability: they are the loci in the model where decisions about alternatives are taken by the system (depending on the current context). In this section, we explain how task contributions to NFRs can be exploited in user-adaptive task models to drive system adaptation so as to choose the alternative that fits best with user preferences.

Inspired by goal analysis (Giorgini et al, 2002), we require analysts to indicate the contribution of each task to individual NFRs. Two possibilities are offered. Analysts can indicate this contribution in the range [-1,+1]. If c is such contribution, a task can be neutral (c=0), provide a negative contribution (c<0), or a positive contribution (c>0). Indicating a numeric value in the range [-1,+1] provides a fine granularity for a better adaptation. The other possibility is that analysts just indicate whether the task provides a negative, positive, or neutral contribution. The system will automatically translate negative into -1, positive into +1, and neutral into 0. Although the specification granularity of this possibility is not as fine as in the first one, it is easier to specify by the analysts. In the rest of the paper, we use the more granular specification. Whatever possibility is used, if analysts specify no value, we consider a neutral contribution relation between the considered task and NFR. This choice about neutrality is made to keep the model easier to specify and read.

While expressing contributions, analysts have to distinguish between tasks where the system automates an activity (automation) and tasks in which the system suggests the user a specific course of actions (recommendation):

- *automation*: the contribution quantifies the direct impact of task execution by the system. For example, contributions for task "raise blinds" refer to the system action of turning on the blinds' engine;
- *recommendation*: the contribution evaluates the indirect impact of having the user following the suggestion. For example, contributions for task "suggest walking" refer to the impact of the user accepting the suggestion and walking to work, and not to the mere action of the recommendation itself.

We require contributions to be specified in correspondence with all variation points in a routine. We suppose that the running system explores the task model for a routine in a top-down fashion, and takes decisions about which alternative to choose whenever it encounters a variation point.

Adaptive task models include three variation point types: (i) exclusive refinement: one subtask is to be selected and executed; (ii) optional tasks: tasks that can be either executed or skipped; and (iii) parametric tasks: tasks dependent on parameters that can be tuned, leading to different runtime behaviors.

Exclusive refinement: the system has to choose the best alternative subtask by comparing the subtasks' contribution to NFRs. For a non-executable task, the contribution approximates the level of contribution of the abstract task, irrespective of the specific executable tasks that refine it. Contributions can be context-dependent. For instance, the contribution of the task "suggest driving" could be -1 to user efficiency and -0.5 to user comfort if there is a traffic jam on the way to work, while it could be 1 to both NFRs otherwise. Take, for instance, the task "suggest transport", and imagine for simplification that is only refined into "suggest driving" and "suggest public transport". The contribution of "suggest driving" is as previously indicated. The contribution of "suggest public transport" is 0.7 to user efficiency and 0.5 to user comfort. Depending on the current context, the subtask that provides the highest contribution will be executed. For instance, consider the case that Sarah has urgent tasks to do (user comfort has priority 0.5, user efficiency 1.0):

- "TrafficJam=true". The average contribution value of "suggest driving" is $\frac{(-1*1)+(-0.5*0.5)}{0.5+1} = -0.83$. The average contribution value of "suggest public transport" is $\frac{(0.7*1)+(0.5*0.5)}{0.5+1} = 0.63$. The contribution of "public transport" is the highest and therefore it will be the subtask to execute.
- "TrafficJam=false". The average contribution value of "suggest driving" is $\frac{(1*1)+(1*0.5)}{0.5+1} = 1$. While the contribution value of "suggest public transport" remains 0.63 (as in the previous case). The contribution of "driving" is the highest and therefore it will be the subtask to execute.

Optional task: the decision is whether to execute or skip the task, depending on its contribution to the NFRs. Contribution to NFRs is expressed as explained for the exclusive refinement variation point. The rule of thumb is that the task is carried out if the weighted contribution to NFRs is positive (>0), and is skipped otherwise. Take, for instance, task "make coffee", and suppose its contribution to NFR user comfort is +0.6 if the user has no early meetings (situation "UrgentTasks" does not hold), -0.8 otherwise; and its contribution to NFR User Efficiency is -0.4. Take Sarah's preferences from Table 1. Depending on the current context, the task is executed or skipped:

- "UrgentTasks=true": user comfort has priority 0.5, user efficiency 1.0. The average contribution value is $\frac{(-0.8*0.5)+(-0.4*1)}{0.5+1} = -0.53$; being negative, the task is skipped;
- "UrgentTasks=false": user comfort has priority 0.7, user efficiency 0.3. The average contribution value is $\frac{(0.6*0.7)+(-0.4*0.3)}{0.7+0.3} = +0.3$; being positive, the task will be executed by the system.

Parametric task: the system has to tune the parameters in order to optimize NFRs. To ease our explanation, we suppose a parametric task has a single numeric parameter in a discrete interval (e.g., the parameter light intensity as an integer number in the range [0,800]). The analyst is expected to assign contribution values for a set of known values, which she obtains either by her expertise, through interviews, from data sheets, or via measurements. The contribution for the missing values are automatically determined by the system via interpolation functions (Shepard, 1968) (e.g., polynomial, spline, cubic). Note that the identification of the most suitable interpolation function goes beyond the scope of this paper, and it generally depends on the number of known data points, their distance, whether they are equidistant, etc.

Take, for instance, the task "switch light on" in Fig. 3. Depending on the light intensity, NFRs energy efficiency, user comfort, and user efficiency receive different contributions. In Table 2, the analysts have specified contributions to the three NFRs for five light intensity values (50, 100, 250, 500, 800 lux), based on her own experience and the light bulb data sheet. In the same table, as an example, a spline interpolation has been applied to compute the contribution values for the missing light intensity values. In this case, we assume that the pervasive system chooses an intensity value with intervals of 25 lux: 50, 75, 100, 125, etc. The actual value will be chosen depending on the context at hand.

Let us assume Sarah is at home; the chosen intensity is determined in function of whether or not she has urgent tasks:

- "UrgentTasks=true": in Table 1, user comfort has priority 0.5, user efficiency 1.0, energy efficiency 0.4. The weighted contribution to these NFRs is computed for each of the intensity values that the system can adopt: 50, 75, 100, etc. The best tuning is either 575, 600, or 625 lux, for any of those values maximize the weighted contribution to the NFRs (see Table 2).
- "UrgentTasks=false": in this context, user comfort has priority 0.7, user efficiency 0.3, energy efficiency 0.4. In this case, the best tuning is 475 lux; this is lower than in the previous case, which is sensible due to the lack of urgency, the higher priority on user comfort, and lower priority on user efficiency. See the last column of Table 2 for the weighted contribution per each possible value of the parameter.

A special case is when a task is both parametric and a subtask in an exclusive refinement. In such case, the analyst has to specify parametric task contributions (and apply interpolations) instead of contextual contributions.

4 Runtime Algorithms

Following Model-Driven Engineering (MDE) guidelines (Pastor and Molina, 2007), user-adaptive task models are machine-processable and are effectively usable as executable models. At runtime, they become the artefact that drives

Table 2 Finding the optimal value for a parametric task based on the average contribution to NFRs. In columns 2–4, the analyst's inputs are marked with a grey background, while the other values are calculated via a spline interpolation (as an example of a well-established interpolation function). Columns 5–6 represent the weighted contribution to NFRs—calculated automatically using the priorities in Table 1—depending on whether or not there are urgent tasks; the optimal values are highlighted with a light green background.

Lux	Contribution to individual NFR			Average contribution	
	Energy Efficiency	User Comfort	User Efficiency	Urgent Tasks	No Urgent Tasks
50	-0.40	0.00	-1.00	-0.547	-0.243
75	-0.20	0.04	-0.84	-0.474	-0.217
100	-0.30	0.10	-0.70	-0.405	-0.186
125	-0.39	0.17	-0.59	-0.348	-0.153
150	-0.47	0.26	-0.49	-0.288	-0.109
175	-0.54	0.35	-0.41	-0.237	-0.067
200	-0.60	0.44	-0.34	-0.189	-0.024
225	-0.65	0.52	-0.27	-0.141	-0.016
250	-0.70	0.60	-0.20	-0.095	0.057
275	-0.74	0.65	-0.13	-0.053	0.086
300	-0.77	0.72	-0.05	0.001	0.129
325	-0.80	0.76	0.03	0.047	0.158
350	-0.81	0.79	0.10	0.090	0.185
375	-0.82	0.81	0.19	0.141	0.211
400	-0.84	0.82	0.28	0.186	0.23
425	-0.85	0.81	0.36	0.224	0.239
450	-0.85	0.78	0.44	0.258	0.241
475	-0.85	0.75	0.52	0.292	0.244
500	-0.85	0.70	0.60	0.321	0.236
525	-0.85	0.64	0.67	0.342	0.221
550	-0.85	0.56	0.74	0.358	0.196
575	-0.85	0.47	0.81	0.371	0.166
600	-0.85	0.37	0.86	0.371	0.126
625	-0.85	0.25	0.92	0.371	0.079
650	-0.86	0.11	0.99	0.369	0.021
675	-0.87	0.04	1.00	0.354	-0.014
700	-0.88	-0.20	1.00	0.288	-0.137
725	-0.90	-0.38	1.00	0.237	-0.233
750	-0.93	-0.57	1.00	0.181	-0.336
775	-0.96	-0.78	1.00	0.119	-0.450
800	-1.00	-1.00	1.00	0.053	-0.571

the adaptive behavior of a pervasive system. All the efforts invested for modeling at design time are reused at runtime providing new opportunities for system adaptation capabilities without further development costs.

The adaptation process is initiated by the occurrence of triggering events, which define *when* the system should adapt. The following triggers refer to changes affecting user-adaptive task models that the pervasive system should keep monitoring at runtime, in order to exhibit the most adequate behavior for the situation at hand:

1. *Changes in preferences*: a user changes her preferences over NFRs, by retaking the NFR contextual prioritization questionnaire (briefly introduced

in Section 3.1 and fully described in (Serral et al, 2017)). This obtains a revised version of the contextual preference model that reflects the new preferences. For example, Sarah may decide to reconsider her user efficiency maximization policy, due to expensive electricity bills, and give higher priority to energy saving.
2. *Task execution faults*: an executable task may return an error, due to either a wrong implementation or faulty effectors. For example, task "raise blinds" may be unsuccessful due to a malfunctioning engine.
3. *Plan failures*: a plan may fail in delivering the expected outcome, even when every task in the plan has executed correctly. The routine fails in engaging the users in carrying out specific actions. For example, if Sarah does not enter the kitchen, task "make coffee" will be inhibited.
4. *Context evolution*: changes in the context may affect the applicability and necessity of specific tasks, as well as they may alter the priority of NFRs. For example, task "turn on bathroom heating" is unnecessary in the hot season, when the indoor temperature is influenced by the outdoor heat. Also, if users are not at home, energy efficiency may be assigned higher priority than other NFRs.

The pervasive system will employ the information about the occurrence of a trigger in the next execution (instance) of a routine. In particular, the system exhibits adaptive behavior if the selected and enacted plan differs from the one that was enacted in the previous routine instance.

We propose two algorithms that enable the execution of a user-adaptive task model, i.e., by selecting and enacting a set of tasks to achieve the purpose of the routine. Being a model-driven approach, our algorithms rely upon the information contained in our models (Figure 2).

Algorithm 1 explores a user-adaptive task model in a top-down fashion and selects an optimal plan to support the routine. A *plan* consists of a set of executable tasks in the routine and ordering constraints between those tasks that define how the routine is carried out step by step. An *optimal plan* is the one that maximizes satisfaction of NFRs for the considered user.

While determining the optimal plan, decisions are taken about (i) whether to execute or skip optional tasks (e.g., when Sarah has urgent tasks, the optional task "make coffee" is skipped so that Sarah can arrive at work earlier); (ii) the most appropriate task in exclusive refinements (when Sarah increases the priority of the NFR energy efficiency, task "raise blinds" replaces task "switch light on"); and (iii) the best value for the parameters in parametric tasks (to improve energy efficiency, the parameter "LightIntensity" of task "switch light on" is set to a lower value than before), i.e., tune parameters (see Algorithm 2).

Algorithm 1 describes the recursive function RunUAdaptTM, which computes the path to follow in a user-adaptive task model, starting from a specific *task*, and given a contextual preference model *pref* and a reference to the current context *ctx*. If the task precondition does not hold in *ctx*, then the task is not to be executed and the function returns (line 1). Otherwise, the algorithm

Algorithm 1 Executing a user-adaptive task model

```
RunUAdaptTM(task, pref, ctx)
 1  if !ctx.Holds(task.GetPrec()) then return
 2  switch task.GetType()
 3    case executable-non-iterative :
 4          task.Execute()
 5    case excl-ref :
 6          toexec ← null
 7          for each subtask ∈ task.GetChildren()
 8          do if subtask.IsTunable()
 9                then subtask.SetBestTuning(pref, ctx)
10             contribval ← subtask.GetContrib(pref, ctx)
11             if toexec = null then toexec ← subtask
12             else if contribval > toexec.GetContrib(pref, ctx)
13                   then toexec ← subtask
14          RunUAdaptTM(toexec, pref, ctx)
15    case temp-ref :
16          for each subtask ∈ task.GetChildren()
17          do if subtask.IsTunable()
18                then subtask.SetBestTuning(pref, ctx)
19             if subtask.IsOptional() ∧ subtask.GetContrib(pref, ctx) < 0
20                then continue
21             RunUAdaptTM(subtask, pref, ctx)
22             rel ← subtask.GetTemporalRelationshipTo()
23             if rel.GetType() = seq-cond then WaitFor(edge.GetEvent())
24             if rel.GetType() = seq-time then Sleep(edge.GetTime())
25  if task.IsIterative() ∧ ctx.Holds(task.GetIterateCond())
26    then RunUAdaptTM(task, pref, ctx)
```

follows depending on the task type (line 2). If the task is executable, it is executed (lines 3-4) by its associated pervasive service. Note that, in this algorithm, tasks are tuned before RunUAdaptTM is invoked for an executable task. If an exclusive refinement is found (line 5), the algorithm selects one subtask (lines 6-13). In particular, if a subtask is tunable, the best tuning is set by function SetBestTuning, based on the user preferences and context (lines 8-9). The contribution value for a task is computed by GetContrib (i.e., $\sum_{j \in NFR} Contrib_{task,j} * Priority_j$). Note that, if the task is tunable, the contribution is the one for the best parameter value that has just been set in lines 8-9. RunUAdaptTM is recursively invoked on the best subtask (line 14). If the task is temporally refined (line 15), the subtasks are sequentially executed. If a subtask is tunable, function SetBestTuning is called (lines 17-18). Optional subtasks are executed only if their contribution is positive (lines 19-20). After invoking RunUAdaptTM on the subtask (line 21), if the outgoing temporal relationship (line 22) is subject to a condition, then either an event is waited for (line 23), or the system sleeps for a specific time period (line 24). If the task is iterative, and the iteration condition still holds (line 25), then RunUAdaptTM is invoked with the same parameters (line 26).

Example 1 (Executing a routine) It's a hot Monday of September, and Sarah has urgent tasks at work. In this context, the weights of NFRs are: user comfort = 0.5, user efficiency = 1, energy efficiency = 0.4. The pervasive system executes Algorithm 1 on the user-adaptive task model of Figure 1. The root task is temporally refined, so its subtasks are examined (lines 16-24):

– the precondition of task “turn on the heating” does not hold since it is a hot day: RunUAdaptTM is recursively called and returns immediately (line 1);
– after ten minutes (line 24), a wake up call is made. The task is exclusively refined (lines 5-14). The buzzer option is chosen by comparing the weighted contributions to NFRs (lines 7-13): since user efficiency has priority over comfort, task “activate buzzer” is executed. Sarah awakes immediately;
– the room is illuminated by choosing to raise the blinds: indeed, the outside brightness is more user efficient than any tuning (lines 8-9) of the parametric task “switch light on” (we will detail the tuning of this task in Example 2);
– Sarah is informed about the weather by executing such task (line 4);
– as shown in Section 3.3, when Sarah has urgent tasks, the optional task “make coffee” is not executed (lines 19-20);
– a transportation mean is suggested. Since Sarah is in a hurry, driving is suggested, as this option has the best contribution to user efficiency. □

Algorithm 2 Tuning a parameter to maximize NFRs

```
SetBestTuning($pref$, $ctx$)
 1  $minvalue \leftarrow$ GetParamMinValue()
 2  $maxvalue \leftarrow$ GetParamMaxValue()
 3  for each $nfr \in pref$.GetNFRs()
 4  do Interpolate($minvalue$, $maxvalue$, $nfr$)
 5  $bestvalue \leftarrow -\infty$
 6  $bestindex \leftarrow$ null
 7  for each $idx : minvalue \leq idx \leq maxvalue$
 8  do $val \leftarrow \sum_{nfr \in pref.\text{GetNFRs()}} nfr.\text{GetContribAt}(idx) * nfr.\text{GetPriority}(ctx)$
 9      if $val > bestvalue$
10         then $bestvalue \leftarrow val$
11              $bestindex \leftarrow idx$
12  SetParam($bestindex$, $bestvalue$)
```

Algorithm 2 describes function SetBestTuning, which configures the parameter of a tunable task to maximize contribution to NFRs. SetBestTuning is a method of class *task* and its inputs are a preference model *pref* and a reference to the current context *ctx*. The minimum and maximum values for the parameter are retrieved (lines 1-2). Then, contributions are interpolated for each NFR (lines 3-4). To determine the best tuning, each possible configuration is considered, and the weighted contribution to NFRs is computed (lines 5-11). Finally, the parameter is tuned (line 12).

Example 2 (Tuning a parameter) Sarah does not have urgent tasks, so her priorities over NFRs are comfort (0.7), energy efficiency (0.4), and user efficiency (0.3). We focus on choosing the best light intensity. Based on these

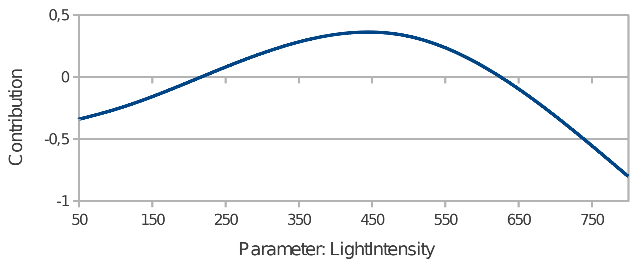

Fig. 4 Weighted sum of the NFRs interpolations for parameter "LightIntensity"

inputs, Algorithm 2 takes the interpolations (lines 3-4), sums the interpolated values (line 8) for each point, and defines the best parameter value (lines 7-11). Figure 4 graphically illustrates the resulting plot. The chosen value is 444, because its weighted contribution to NFRs is the highest (+0.36). □

5 Implementation and Evaluation

5.1 Prototype Implementation

To evaluate the feasibility of our approach, we have implemented the proposed algorithms to execute the modeling language. We have extended the prototype implemented for executing the task models used as our baseline (Section 2) to support user-centered adaptation based on optimizing NFR satisfaction.

The baseline prototype builds on the services provided for the pervasive environment to control the needed devices (see Fig. 5). We consider a service as a mechanism that provides a coherent set of functionality described in terms of atomic operations (or methods). These operations allow the system to control the devices of the environment in order to change context and/or sense it. Our approach uses these pervasive services in order to perform the tasks of the routines specified in the task models and to sense context changes. The pervasive services are developed in Java/OSGi 4 technology. Besides making use of pervasive services, the baseline prototype consists of three main components:

– OCean (Ontology-based Context model management mechanisms): the context on which the routines depend is specified in an ontology-based context model as OWL individuals. OWL (Ontology Web Language) (Smith et al, 2004) is a machine-readable notation and a W3C standard that provides a high level of expressiveness to represent context data (Ye et al, 2007). The definition of the domain is represented by classes, their relationships and constraints, while the current context data for a specific application is specified as OWL individuals. In order to manage these individuals, OCean was developed using Jena TDB[2], a Java framework for

[2] http://jena.apache.org/documentation/tdb/

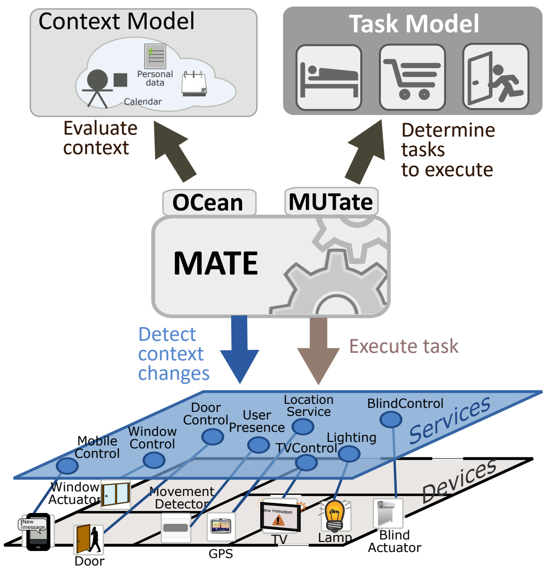

Fig. 5 Overall architecture of our approach

building Semantic Web applications that provides a programmatic environment for OWL and SPARQL (a standard language for querying ontologies), and includes a rule-based inference engine. We have used Jena TDB to be able to interpret the context and determine whether or not the context conditions used in the routines are satisfied. For instance, using OCean, we can determine the temperature and illumination of the room, or whether the user is busy or not.

– MUTate (Model-based User Task management mechanisms): in order to support the management of the task model, its metamodel was specified in Ecore. Ecore is a reference implementation of OMG's EMOF[3] (Essential Meta-Object Facility) and allows expressing other models by defining their constructs. The persistent representation of the Ecore metamodel and its models is XMI (XML Metadata Interchange). From the Ecore metamodel, we used the Eclipse Modelling Framework (EMF)[4] to generate a Java API for managing a task model at runtime. This API was the basis of MUTate and provides *get* and *set* methods to access and change the information of the instances specified in a task model. Thus, MUTate allows, for instance, a routine or a task to be retrieved or modified. In addition, the metamodel class "Situation" represents a context condition. In this class, we implemented a method to check whether the condition is fulfilled or not. This method interprets the logical expression of the condition and builds a

[3] OMG's EMOF: http://www.omg.org/mof/
[4] EMF: http://www.eclipse.org/modeling/emf/

query in SPARQL. Once the query has been built, the method uses OCean to launch the query against the context model.

– MATE (Model-based user task Automation Engine): this component is in charge of monitoring the context and executing the routines specified in the task model when their corresponding activation condition holds. The execution of the routines is performed in a context-adaptive way according to their specification in the task model and the current context stored in the context model. To interpret the task model and the context model, MATE uses MUTate and OCean, respectively. MATE is implemented in Java and built on top of the OSGi middleware[5]. OSGi has bridges to most of the technologies used in home automation systems and provides high-level implementation constructs and facilities to, for instance, manage the services of the system at runtime. Thus, using OSGi, we can easily add new services to the pervasive system, and search for and execute the specific service needed to perform a routine task at runtime.

To be able to support the presented approach for user-centered adaptation, we have built upon this prototype, by reusing OCean and extending MUTate and MATE as follows:

– MUTate: in order to represent user preferences over NFRs and task contributions to those NFRs, we have extended the Ecore metamodel as explained in Section 3. From this metamodel, we use EMF to regenerate the Java API that allows an adaptive task model to be managed at runtime. EMF also generates a tree editor to create syntactically well-formed models. Using this editor, we created the adaptive task model of Figure 1.
– MATE: we have extended the routine execution algorithm to support the user preferences over the NFRs and the task contributions. To do this, we have implemented the method GetContrib to calculate the total contribution of a task to the NFRs for a specific context. Afterwards, we implemented the algorithm SetBestTuning (Algorithm 2, Section 4) to calculate the best possible parameter of a parametric task in a specific context as well as the contribution of the task with that parameter. Finally, we extended the previous algorithm to execute a routine in order to include the possible NFR adaptations as depicted in function RunAdaptTM (Algorithm 1, Section 4).

5.2 Evaluation and Scalability Analysis

Firstly, we remark that the prototype that constituted the research baseline of this work has been already extensively evaluated with positive results. Earlier studies have examined the completeness and correctness of the implementation, as well as its scalability and performance in real-world pervasive environments (Serral et al, 2013b,a). Also, the advantage of the model-driven

[5] OSGi - http://www.osgi.org/

approach in comparison to traditional development methods has been studied in (Serral et al, 2013a).

The novelty of this work consists in supporting adaptation to NFR. This addition introduces some overhead in the time needed to design the system, when compared to the research baseline approach and implementation. This overhead is due to the need of creating two additional models for designing the system: the NFR model, which is created for each system user; and the contribution model, which is created for each task that may have a different contribution to the NFRs according to context. However, this overhead, which may take a few days for a real-world smart home system, is considerably compensated by the improvements that the approach provides:

– *Personalization to different users*: our proposal allows the system to be easily personalized by specifying the user preferences in the contextual preference model. Using the contextual requirements prioritization method that we have developed (Serral et al, 2017), these preferences are smoothly captured and automatically transformed into the formal specification of user contextual preference model. This model enables a routine to be executed according to the personal preferences of each user over NFRs.
– *Modularity and reusability*: in the baseline, adaptations were possible only by specifying context preconditions and conditional temporal constraints. These adaptations were all included in the task model, and were not user-specific. In our proposal, instead, user preferences are located in the contextual preference model. This introduces higher modularity, which facilitates the reusability of the different models. For instance, it is now possible to reuse the same task model for different users, or the same contextual preference model for different pervasive systems that support the same user.
– *Explicit semantics for exclusive refinements*: in the baseline, the system would always execute the first encountered task in an exclusive refinement whose context conditions evaluated to true. This semantics was somewhat implicit and system designers had to take this into account to let the system exhibit their desired behavior. In user-adaptive task models, the semantics becomes clear and explicit: the task that provides a higher contribution to the NFRs is the one to be executed.
– *Higher quality of service*, thanks to the specification and use of user preferences over NFRs. Routines are executed with a higher level of adaptation that makes them fit better the behavior expected by the users in each situation, thereby making the users perceive a higher quality of service and satisfaction. Specifically, *optional tasks* enable skipping the execution of a task when it does not fit the user preferences in the current context. Thus, for instance, the task "make coffee" can be avoided when the user cares more about user efficiency than user comfort. *Parametric tasks* support the fine-grained tuning of a task execution, determining the values, such as room temperature or illumination, that better fit the user preferences and the context at execution time. Also, the higher level of adaptation in the *exclusive refinements* supports a better selection of the task to be

executed; for instance, when the user comfort has priority over the user efficiency, the user will be waken up by listening to music; otherwise, the user will be waken up by the buzzer sound, which is more effective for a prompt wake-up.

We have studied the scalability of our prototype by measuring and comparing it against the baseline approach. Our intent is to determine to what extent our extension incurs in an overhead for interpreting the extended model at runtime. We quantified both the execution time and the memory consumption of computing the plan for executing a specific routine in each approach. We run both prototypes with generated models that are close to reality by adding in an iterative way a duplication of the Waking up routine (see Figure 1). The duplication was linked by an enablement temporal relationship from the last task of the model used in the previous iteration (i.e., suggest transport) to the root task of the duplicated model (i.e., the waking up task). We duplicated the routine up to a size that was 40 times larger than the original. The largest executed routine had 741 tasks, way larger than any real-world human routine.

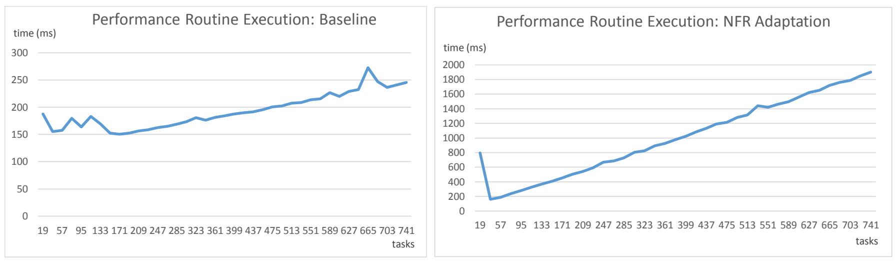

Fig. 6 Temporal cost of executing a routine using the baseline approach and the NFR-adaptive approach

The results concerning time are shown in Figure 6: the baseline is presented on the left, while the NFR-adaptive approach is on the right. Regarding memory consumption, the baseline approach did not require more than 8MB of memory during the scalability test, while the NFR-adaptive approach did not use more than 30MB. While there is some overhead, the results show that this is not significant for modern machines, and especially that the approach still scales up linearly, and not exponentially. Furthermore, the response time offered by the presented approach is acceptable when compared to the performance of the devices and communication networks usually used in pervasive systems, which is usually in terms of tens or hundreds of milliseconds.

6 Related Work

We review relevant related work and show how it relates with our approach. Specifically, we discuss techniques for routine automation in Subsection 6.1, approaches based on goals and NFRs in Subsection 6.2, and the use of models at runtime in Subsection 6.3.

6.1 Routine Automation Approaches

Several approaches have been proposed to support user routines by using pervasive systems. In order to classify these approaches, we use the taxonomy published by Chin et al (2008), which defines three different categories: machine-learning approaches, rule-based context-aware approaches and user-centered approaches.

Machine-learning approaches use algorithms to infer user behavior patterns from historical data and to automate them. Some outstanding examples are: MavHome (Youngblood et al, 2005) and iDorm (Hagras et al, 2004). The MavHome project, extended by the CASAS project (Rashidi and Cook, 2009), uses prediction algorithms to identify common sequential patterns from data captured from the sensors of a smart home. From this learning, Mavhome and CASAS build a Markov model of user behavior in which patterns are specified through a series of states linked by transitions with certain probabilities. The iDorm project predicts user behavior by learning fuzzy rules that map sensor state to actuator readings representing inhabitant action. These approaches introduce a great automation in the learning of the routines; however, they have some limitations:

1. They require a great amount of training data to star infer user actions, and therefore these approaches cannot be used until sufficient user past actions are registered. This is known as the cold-start problem, see (Serral et al, 2011) for more details.
2. Lack of knowledge about tasks performed by the users may lead to automating tasks for which the users may not want automation or to generic automated behaviors that become a burden for the user;
3. They can only reproduce the actions that users have executed in the past in the same way the users executed them.

In our approach, we tackle these problems. We provide the system with a set of tasks that fit the user requirements and that are automated from the moment the system starts running. In this way, the system does not need to wait to collect and register enough user actions, it can start supporting users in their daily tasks from the very beginning. This improves the cold-start problem. In addition, the preferences of the users over NFR are taken into account and tasks can also be automated regardless of whether or not the users have performed them in the past.

Rule-based, context-aware approaches program rules to automate user actions when a certain context condition arises. Some outstanding examples are (Henricksen et al, 2006) and (García-Herranz et al, 2010). In (Henricksen et al, 2006) the system is composed of independent agents, each of which comprises a set of rules that encode reactions to context states. Each rule follows the template: *When <triggers>, if <conditions>, then <action>*. In (García-Herranz et al, 2010) the authors propose situation and preference abstractions that are used by ECA rules to automatically trigger certain actions. However, rule-based techniques generally require large numbers of rules that have to be manually programmed (Cook and Das, 2004) and that efore, the proposed task models do not provide enough expressiveness (such as specific relationships between tasks, context situations, etc.) or runtime support for their execution and evolution.

6.2 Goal- and NFR-Driven Adaptation

The use of NFRs to drive decision-making has been widely explored in Goal-Oriented Requirements Engineering (GORE), i.e., modeling languages that represent requirements as goals and related concepts such as actors, resources, tasks. Most GORE approaches rely upon (variants of) the NFR framework (Mylopoulos et al, 1992) or the $i*$ framework (Yu, 1995), and exploit the concept of soft-goal to represent NFRs and reason about NFR satisfaction.

Chung et al (1995) use a NFR graph to inform the selection among alternative architectural designs. Approaches in adaptive software systems (Salehie and Tahvildari, 2012; Goldsby et al, 2008; Lapouchnian et al, 2006; Dalpiaz et al, 2013) exploit soft-goals to identify the system configuration that maximizes the satisfaction of a set of NFRs. Our approach is also based on the optimization of NFRs; however, unlike existing approaches, we take into account the priorities over NFRs of each user, and we allow for expressing contextual contributions. These characteristics are essential when considering adaptation in pervasive environments.

Liaskos et al (2010) introduce preferences and optionality (in goals and temporal constraints) for GORE, and reduce the problem of configuring a system to that of finding a plan that maximizes the stated preferences. While their framework shares some similarities with our approach—both support optional tasks and preferences—ours is intended for runtime usage, and supports adaptation in response to changes in the context (including multiple users).

Brown et al (2006) take an orthogonal approach: they exploit a goal-oriented specification to define adaptation requirements, i.e. how the system has to behave in order to switch from one configuration to another. In a similar spirit, Silva Souza et al (2011) define awareness requirements as meta-requirements to drive adaptations. Our approach embodies an implicitly-expressed adaptation requirement which is globally applied to user routines, i.e. the optimization of user preferences over NFRs. It would be interesting to study whether adaptation/awareness requirements can be used within user-adaptive task models to specify more fine-grained adaptation policies.

Some Business Process Modelling (BPM) languages (Pavlovski and Zou, 2008; Ayora, 2011) extend BPMN-like notations by including NFRs as performance indicators; however, these approaches are limited to the monitoring of NFRs, and do not consider the adaptation (or variation) of processes according to NFRs. An interesting research direction is to assess the suitability of BPM languages as an alternative notation to represent user routines, and porting our adaptation algorithms to that notation.

6.3 Models at Runtime

In this section, we study some of the most important works that use models at runtime (Blair et al, 2009) for software adaptation.

Several approaches have used models at runtime for adapting system architectures. Oreizy et al (1999) were pioneers in their adoption of an architecture-based model to support runtime software adaptation. Floch et al (2006) promote the use of architecture models to support the development of adaptive mobile applications by the use of policies. Morin et al (2009) propose a combination of model-driven and aspect-oriented techniques to support dynamic runtime reconfiguration. Cetina et al (2009) propose the reuse of design variability models at runtime to support the self-configuration of systems when triggered by changes in the environment. Garlan and Schmerl (2004) use architecture-based models during runtime for system monitoring, problem detection, and repair. The same group also proposed the use of architectures to adapt pervasive systems (Cheng et al, 2002); their focus, however, is mainly on low-level details, such as the number of available servers, or latency.

Differently, MOCAS (Ballagny et al, 2009) relies on behavioral adaptation instead of structural adaptation. It deals with component-based systems whose behavior and adaptation can be described by UML state machines.

Blumendorf et al (2008) focus on the development of adaptive user interfaces and their adaptation at runtime combining multiple models. Also, others works use models at runtime to update software code. Amoui et al (2012) present a Graph-based Runtime Adaptation Framework (GRAF) to provide runtime adaptation to Java applications that are already implemented. Al-Refai et al (2014) have developed the FIGA framework, which permits developers to update running software through changes made to the system's UML models.

We differ from these approaches for our focus on the adaptation of complex system behaviors (user routines) according to user preferences over NFRs.

7 Conclusions and Future Work

We have introduced user-adaptive task models for AmI, an executable modeling language that guides pervasive systems at runtime by providing personalized and context-aware support for users in their daily routines. The distinguishing feature of our approach is that our models include the decision-making rationale for adaptive behavior by representing user preferences over NFRs and the contributions of tasks to NFRs.

By handling preferences over NFRs as a separated model, a specific routine can be presented to the user in different ways just by changing the contextual preference model, without altering the routine description. At the same time, the same preference model can be reused in multiple routines to describe the preferences over NFRs of a specific user.

Based on our modeling language, we have built algorithms that specify how the pervasive system makes runtime use of the models to exhibit user-adaptive behavior. We have illustrated the approach and discussed its benefits in the smart homes domain, and we have implemented the approach and reported on scalability tests that show feasibility for real-world routines.

In future work, we will study the applicability of our approach in other domains where user adaptation is a main requirement. Moreover, we will extend our approach to include historical data about (un)successful executions. By reasoning over this data, we will be able to provide a better tuned system adaptation. Furthermore, we need to study how humans perceive the adaptive behavior that our framework can deliver.

References

Al-Refai M, Cazzola W, France R (2014) Using models to dynamically refactor runtime code. In: Proceedings of the Annual ACM Symposium on Applied Computing, ACM, pp 1108–1113

Amoui M, Derakhshanmanesh M, Ebert J, Tahvildari L (2012) Achieving dynamic adaptation via management and interpretation of runtime models. Journal of Systems and Software 85(12):2720–2737

Ayora C (2011) Modelling and Managing Variability in Business Process Models. Master's thesis, Universitat Politècnica de València

Ballagny C, Hameurlain N, Barbier F (2009) Mocas: A state-based component model for self-adaptation. In: Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, IEEE, pp 206–215

Blair G, Bencomo N, France RB (2009) Models@ run. time. Computer 42(10):22–27

Blumendorf M, Lehmann G, Feuerstack S, Albayrak S (2008) Executable models for human-computer interaction. In: Proceedings of the International Workshop on the Design, Specification, and Verification of Interactive Systems, Springer, pp 238–251

Brown G, Cheng BHC, Goldsby H, Zhang J (2006) Goal-oriented Specification of Adaptation Requirements Engineering in Adaptive Systems. In: Proc. of SEAMS '06, ACM, pp 23–29

Calvaresi D, Cesarini D, Sernani P, Marinoni M, Dragoni AF, Sturm A (2017) Exploring the ambient assisted living domain: a systematic review. Journal of Ambient Intelligence and Humanized Computing 8(2):239–257

Cetina C, Giner P, Fons J, Pelechano V (2009) Using feature models for developing self-configuring smart homes. In: Proceedings of the International Conference on Autonomic and Autonomous Systems, IARIA, pp 179–188

Cheng SW, Garlan D, Schmerl B, Sousa JP, Spitznagel B, Steenkiste P, Hu N (2002) Software architecture-based adaptation for pervasive systems. In:

Proceedings of the International Conference on Architecture of Computing Systems, Springer, pp 67–82

Chin J, Callaghan V, Clarke G (2008) A programming-byexample approach to customising digital homes. IET International Conference Intelligent Environments

Chung L, Nixon B, Yu E (1995) Using Non-Functional Requirements to Systematically Select among Alternatives in Architectural Design. In: Proc. of IWASS'95, ACM, p 3143

Cook D, Das S (2004) Smart environments: technology, protocols and applications, vol 43. John Wiley & Sons

Dalpiaz F, Serral E, Valderas P, Giorgini P, Pelechano V (2012) A NFR-based Framework for User-Centered Adaptation. In: Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012), Springer, LNCS, vol 7532, pp 439–448

Dalpiaz F, Giorgini P, Mylopoulos J (2013) Adaptive socio-technical systems: a requirements-based approach. Requirements engineering 18(1):1–24

Dey AK, Hamid R, Beckmann C, Li I, Hsu D (2004) a CAPpella: Programming by demonstration of context-aware applications. ACM Conference on Human Factors in Computing Systems (CHI 2004) pp 33–40

Floch J, Hallsteinsen S, Stav E, Eliassen F, Lund K, Gjorven E (2006) Using architecture models for runtime adaptability. IEEE Software 23(2):62–70

Gajos K, Fox H, Shrobe H (2002) End user empowerment in human centered pervasive computing. International Conference on Pervasive Computing (Pervasive 2002)

García-Herranz M, Haya PA, Alamán X (2010) Towards an ubiquitous end-user programming system for smart spaces. Journal of Universal Computer Science (JUCS) 16(12):1633–1649

Garlan D, Schmerl B (2004) Using architectural models at runtime: Research challenges. In: Proceedings of the International Conference on Software Architecture, Springer, pp 200–205

Giorgini P, Mylopoulos J, Nicchiarelli E, Sebastiani R (2002) Reasoning with goal models. In: Proc. of ER 2002, pp 167–181

Goldsby HJ, Sawyer P, Bencomo N, Cheng BHC, Hugues D (2008) Goal-based modeling of dynamically adaptive system requirements. In: Proc. of ECBS 2008, IEEE, pp 36–45

Griss ML, Favaro J, d Alessandro M (1998) Integrating feature modeling with the rseb. In: Software Reuse, 1998. Proceedings. Fifth International Conference on, IEEE, pp 76–85

Hagras H, Callaghan V, Colley M, Clarke G, Pounds-Cornish A, Duman H (2004) Creating an ambient-intelligence environment using embedded agents. IEEE Intelligent Systems 19(6):12–20

Henniger O, Damer N, Braun A (2017) Opportunities for biometric technologies in smart environments. In: Braun A, Wichert R, Maña A (eds) Ambient Intelligence: 13th European Conference, AmI 2017, Malaga, Spain, April 26–28, 2017, Proceedings, Springer International Publishing, Cham, pp 175–182

Henricksen K, Indulska J (2004) Developing context-aware pervasive computing applications: Models and approach. In: Pervasive and Mobile Computing, vol 2, pp 37–64

Henricksen K, Indulska J, Rakotonirainy A (2002) Modeling context information in pervasive computing systems. In: Proc. of the International Conference on Pervasive Computing (Pervasive'02), Springer, pp 167–180

Henricksen K, Indulska J, Rakotonirainy A (2006) Using context and preferences to implement self-adapting pervasive computing applications. Sofware-Practice and Experience

Lapouchnian A, Yu Y, Liaskos S, Mylopoulos J (2006) Requirements-driven Design of Autonomic Application Software. In: Proc. of CASCON 2006

Liaskos S, McIlraith SA, Sohrabi S, Mylopoulos J (2010) Integrating preferences into goal models for requirements engineering. In: Proc. of the IEEE International Requirements Engineering Conference (RE), IEEE, pp 135–144

Morin B, Barais O, Jezequel JM, Fleurey F, Solberg A (2009) Models@Run.time to support dynamic adaptation. IEEE Computer 42(10):44–51

Mylopoulos J, Chung L, Nixon B (1992) Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Transactions on Software Engineering 18(6):483–497

Oreizy P, Gorlick MM, Taylor RN, Heimbigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf AL (1999) An architecture-based approach to self-adaptive software. IEEE Intelligent systems 14(3):54–62

Pastor O, Molina JC (2007) Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer-Verlag

Paternò F (2002) Concurtasktrees: An engineered approach to model-based design of interactive systems. In: The Handbook of Analysis for Human-Computer Interaction, Lawrence Erlbaum Associates, pp 483–500

Pavlovski CJ, Zou J (2008) Non-functional requirements in business process modeling. In: Proc. of APCCM 08, p 103112

Rashidi P, Cook DJ (2009) Keeping the resident in the loop: Adapting the smart home to the user. IEEE Transactions on Systems, Man, and Cybernetics 39

Saaty TL (1990) How to make a decision: the analytic hierarchy process 48(1):9–26

Sadri F (2011) Ambient intelligence. ACM Computing Surveys 43(4):1–66

Salehie M, Tahvildari L (2012) Towards a goal-driven approach to action selection in self-adaptive software. Software: Practice and Experience 42:211–233

Satyanarayanan M (2001) Pervasive computing: Vision and challenges. Personal Communications, IEEE 8(4):10–17

Serral E (2011) Automating routine tasks in smart environments. a context-aware model-driven approach. PhD thesis, Polytechnic University of Valencia

Serral E, Valderas P, Pelechano V (2010a) Automating routine tasks in ami systems by using models at runtime. In: AmI-10, pp 1–10

Serral E, Valderas P, Pelechano V (2010b) Supporting runtime system evolution to adapt to user behaviour. In: Proc. of CAiSE'10, pp 378–392

Serral E, Valderas P, Pelechano V (2011) Improving the cold-start problem in user task automation by using models at runtime. In: Information Systems Development, Springer, pp 671–683

Serral E, Valderas P, Pelechano V (2013a) Addressing the evolution of automated user behaviour patterns by runtime model interpretation. Software & Systems Modeling pp 1–34

Serral E, Valderas P, Pelechano V (2013b) Context-adaptive coordination of pervasive services by interpreting models during runtime. The Computer Journal 56(1):87–114

Serral E, Sernani P, Dragoni AF, Dalpiaz F (2017) Contextual requirements prioritization and its application to smart homes. In: Proceedings of the 13th European Conference on Ambient Intelligence, AmI 2017, pp 94–109

Shepard D (1968) A Two-dimensional Interpolation Function for Irregularly-spaced Data. In: Proc. of the ACM national conference, pp 517–524

Shepherd A (2001) Hierarchical Task Analysis. Taylor & Francis, London

Silva Souza VE, Lapouchnian A, Robinson WN, Mylopoulos J (2011) Awareness Requirements for Adaptive Systems. In: Proc. of SEAMS '11, pp 60–69

Smith, Welty, McGuinness (2004) Owl web ontology language guide

Sutcliffe A, Fickas S, Sohlberg M (2005) Personal and contextual requirements engineering. In: Proc. of RE 2005, pp 19–28

Truong KN, Huang EM, Abowd GD (2004) Camp: A magnetic poetry interface for end-user programming of capture applications for the home. 6th International Conference on Ubiquitous Computing (UbiComp) 3205 of Lecture Notes in Computer Science:143–160

Ye J, Coyle L, Dobson S, Nixon P (2007) Ontology-based models in pervasive computing systems. The Knowledge Engineering Review 22(04):315–347

Youngblood GM, Cook DJ, Holder LB (2005) Managing adaptive versatile environments. Pervasive and Mobile Computing

Yu E (1995) Modelling strategies relationships for process reengineering. PhD thesis, Department of computer science, University of Toronto