A NFR-based Framework for User-Centered Adaptation

Fabiano Dalpiaz, Estefanía Serral, Pedro Valderas, Paolo Giorgini, and Vicente Pelechano

June 2012

Technical Report # DISI-12-022

A NFR-based Framework for User-Centered Adaptation

Fabiano Dalpiaz[1], Estefanía Serral[2], Pedro Valderas[2], Paolo Giorgini[1], and Vicente Pelechano[2]

[1] DISI, University of Trento
{dalpiaz,pgiorgio}@disi.unitn.it
[2] PROS, Universitat Politècnica de València
{eserral,pvalderas,pele}@dsic.upv.es

Abstract. Pervasive environments support users' daily routines in an invisible and unobtrusive way. To do so, they include a technical pervasive infrastructure, which is aware of and adaptive to both the operational context and the users at hand. Non-Functional Requirements (NFRs) have been effectively used to inform decision-making in software engineering: functional alternatives are compared in terms of their contribution to NFRs satisfaction. In this work, we consider user preferences over NFRs as a key driver for the adaptation of a pervasive infrastructure. We devise a model-driven framework for building pervasive systems that maximize fitness with the context and the user. Our contributions are: (i) *adaptive task models*, a conceptual model to describe user routines that accounts for user preferences over NFRs; and (ii) an adaptation framework, which uses our models at runtime to guide a pervasive infrastructure in adapting its behaviour to user preferences and context.

Keywords: conceptual models at runtime; pervasive environments; self-adaptation; non-functional requirements

1 Introduction

Pervasive computing [20] is a computational paradigm in which technical systems are developed and deployed in the environment—which becomes a so-called *pervasive environment*—so as to support humans in their daily activities. A key requirement for pervasive environments is to remain, while executing, invisible and unobtrusive to users. Pervasive environments include a technical infrastructure—a *pervasive infrastructure*—that effects changes in the environment and suggests appropriate activities to the users. While being guided by this system, the user should not realize that the system is "thinking" on her behalf.

Task models [15] are a state-of-the-art modelling language to represent user routines [21]. They are an example of executable conceptual models, as they hierarchically specify the tasks a system should execute in order to support a user in the conduction of her daily routines. Task models have been successfully adopted in model-driven pervasive infrastructures [21]; however, although these models have shown to be effective for the functional aspects of routine automation, they provide limited adaptation to user preferences about non-functional properties. In order to be unobtrusive and invisible, the system has to select and execute routines that support courses of action the user finds natural to her (i.e., that match her preferences). If the user is highly concerned with energy saving and carbon emissions reduction, the system should minimize heating usage and suggest eco-friendly transportation means. Instead, if the user has scheduled early meetings, she will give lower priority to eco-friendliness, and the system should recommend efficient transportation means such as driving.

Our aim is to investigate how user preferences over non-functional properties can be taken into account by a pervasive infrastructure. Our approach is based on using Non-Functional Requirements (NFRs) [13]. NFRs have been successfully used in software engineering to inform decision-making—based on the quality of each alternative—in requirements models [26], architectures [5], and business processes [16]. The challenge is to effectively use NFRs with task models.

In this paper, we extend task models and propose a model-driven framework that enables a pervasive infrastructure to adapt its behaviour to the user preferences and the current context. Our contributions are as follows:

- *adaptive task models*, a modelling language that enriches task models with NFRs. The model is created at design-time, and used by the system at runtime to decide upon how to adapt its behaviour. User preferences over NFRs are captured by our proposed *contextual preference model*: each user specifies, in a context-dependent way, the priority she assigns to each NFR;
- *an adaptation framework* for building pervasive infrastructures that exploit adaptive task models and contextual preference models at runtime. We investigate the event types that trigger system adaptation, and devise algorithms for a system to execute adaptive task models at runtime.

The rest of the paper is organized as follows. Sec. 2 presents our baseline: task models. Sec. 3 introduces adaptive task models and the contextual preference model. Sec. 4 proposes an adaptation framework for pervasive infrastructures that exploits our proposed models at runtime. Sec. 5 discusses related work, while Sec. 6 draws our conclusions and outlines future directions.

2 Research Baseline: Task Models

We build on top of the task models by Serral et al. [21], which specify how a pervasive infrastructure can support its users in carrying out everyday activities. Task models are inspired by Hierarchical Task Analysis (HTA) [23], which constructs a task tree that refines a high-level task into a set of executable ones.

Running example. A smart-home pervasive environment supports the daily routines of the home inhabitants. The smart-home is equipped with Ambient Intelligence devices, controlled by pervasive services [21], that monitor and collect context information, as well as enable interaction with the users and the environment. One of the supported routines 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 Bob to take a shower. At 8:00 a.m., the system makes a wake-up call, repeating it until Bob wakes up. Then, the room is illuminated and Bob is notified about the weather. Afterwards, when Bob enters the kitchen, the system makes a coffee, and suggests him the best way to go to work." □

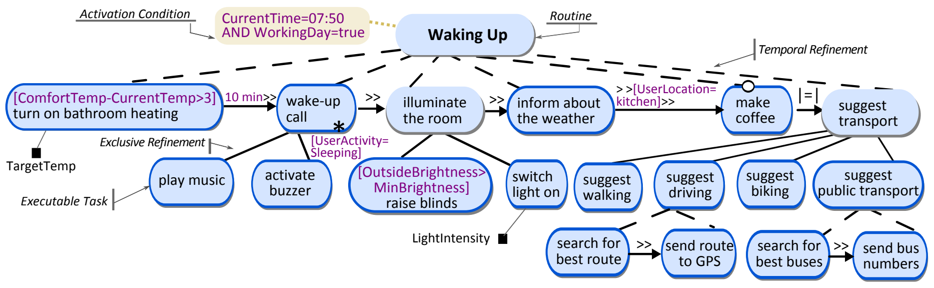

Fig. 1. Task model representing the "Waking Up" user routine

Using HTA, each user routine is described as a task hierarchy. Fig. 1 illustrates the "Waking Up" routine. The root task represents the routine itself, 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 a specific order which is graphically depicted by the arrows between sibling tasks. Temporal constraints make use of Concurrent Task Trees (CTT) operators [15]. For example, in Fig. 1, we use:

- *Enablement* ($T_1 \gg T_2$): task T_2 is triggered when task T_1 finishes. For instance, 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, "make coffee" and "suggest transport" are temporally independent.

The task refinement process ends when every leaf task in the tree is associated with a pervasive service (controlled by the pervasive infrastructure), 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" [11]). Situations are used to indicate the relationship between context and routine execution (see Fig. 1):

- *Activation condition*: it is associated with the root task of each routine. It indicates the situation in which the routine is activated. For instance, the "Waking Up" routine is to be executed every working day at 7:50 a.m.;

– *Task precondition*: it can be associated with a task to indicate that its execution depends on the whether a situation holds (depicted between square brackets before the name of a task). For instance, 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 Fig. 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;
  - $T_1\ t \gg T_2$: 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".

3 Adaptive Task Models

Guided by the notions of context suggested by Sutcliffe et al. [25], we propose an extended version of task models to describe system behaviour that takes into account a comprehensive set of contextual factors. We call them *adaptive task models* to highlight their capability to adapt to the context at hand.

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 and open doors).

Our extension enables not only adaptation to the preferences of different users—while one may be more concerned with energy efficiency, another may give priority to user comfort—, but also to the changing preferences of a specific user. For instance, if a user is in a hurry, she may favour efficiency over comfort; when not at home, instead, she may be more interested in energy saving.

Fig. 2 depicts the meta-model of adaptive task models. The red-coloured classes show our proposed contextual preference model (Sec. 3.1), which indicates user preferences over NFRs. The white-coloured classes represent the extended task model itself: we introduce optional and parametric tasks (Sec. 3.2), as well as task contributions to NFRs (Sec. 3.3). The green-coloured classes represent the context model (from previous work [21]).

3.1 Contextual Preference Model for NFRs

Each user has different preferences, which depend on the situation and vary over time. To represent the preferences over NFRs of a user, we propose the contextual

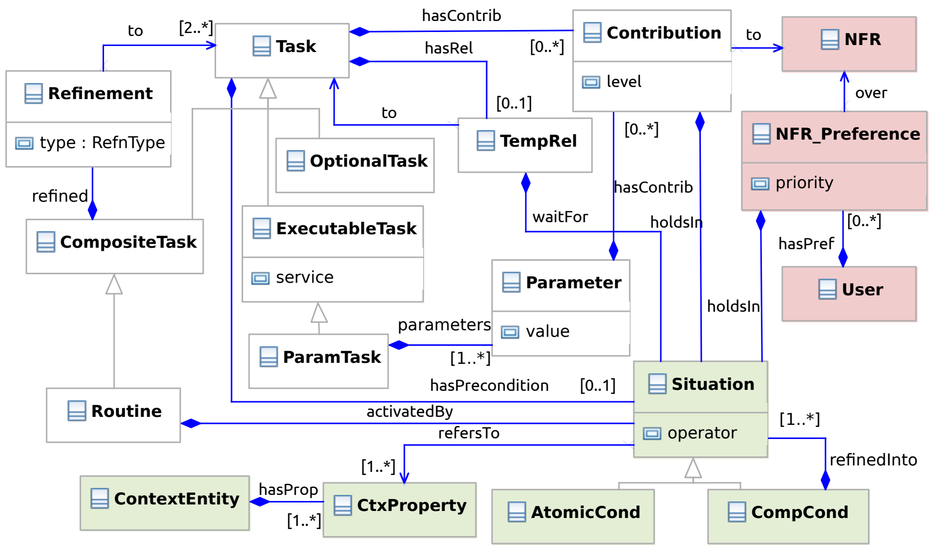

Fig. 2. Overview of adaptive task models

preference model inspired by Henricksen et al.'s preference model [11]. Using the contextual preference model, analysts define the relevant NFRs and the priority each user assigns to them in different contexts.

Table 1. Partial contextual preference model for user Bob

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⟩

For each NFR, a set of couples consisting of a situation s and a weight w (a real number in the range [0,1]) is specified. Each couple indicates that, when situation s holds, the NFR has priority w. Value 0 indicates minimum importance, while value 1 indicates maximum importance. The situations for a given NFR have to be mutually exclusive. Table 1 shows part of the contextual preference model for Bob. The weight of NFR user comfort is 0 when he is not at home; if Bob is at home, it is 0.7 if he has no urgent tasks, 0.5 otherwise.

Analysts can elicit this kind of models, for instance, by using or extending state-of-the-art techniques in requirements prioritization. We do not address this challenge here; for details, see, e.g., the work by Perini et al. [17].

3.2 Task Models with Optional and Parametric Tasks

Our approach to adaptation is model-driven in general, and NFR-driven in particular. The system adapts by choosing an adequate course of action from its task model, on the basis of contextual factors and user preferences over NFRs. To such extent, we enrich task models with optional and parametric tasks.

Optional tasks are not essential to accomplish a specific routine. These tasks are based on the optional tasks proposed in CTT [15] 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. 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. Note that optional is different from contextual: a contextual task is executed only if the context precondition holds, while an optional task relies on preferences. Graphically (see Fig. 3), optional tasks are represented by drawing a hollow circle to the incoming refinement link (like optional features in feature models [10]).

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. Graphically (see Fig. 3), parameters are labels—representing the parameter name—connected to leaf tasks by a dotted line ending with a square-shaped arrow.

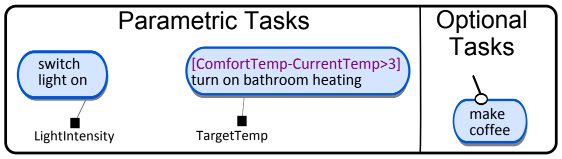

Fig. 3. Graphical representation of parametric and optional tasks

3.3 Linking Tasks to NFRs via Contributions

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

In the tradition of goal analysis [8], we require analysts to indicate the contribution of each task to individual NFRs 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). If analysts specify no value, we consider a neutral contribution relation between the considered task and NFR.

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 accepting the suggestion and walking to work, and not to the action of recommending the user.

We require contributions to be specified in correspondence of 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: can be either executed or skipped; and (iii) parametric tasks: parameters can be tuned, leading to different runtime behaviours. We detail each variation point type:

Exclusive refinement: the system has to choose the best alternative subtask by comparing their 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, consider NFR user comfort and task “suggest driving” in the “Waking Up” routine. The contribution of this task could be +0.5 if the user lives and works in the city outskirts, -0.5 if the user either works or lives in the city centre, where traffic jams are very likely to occur.

Optional task: the decision is whether to execute or skip the task, depending on the contribution to NFRs and user preferences. 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 Bob’s preferences from Table 1. Depending on the current context, the task is executed or skipped:

– "UrgentTasks=true": user comfort has weight 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 weight 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 so to optimize NFRs. If the task depends on multiple parameters, in order to simplify the specification of contributions, we suppose the analysts will merge these parameters into a single numeric parameter in a discrete interval (e.g. an integer between 1 and 10). The analyst assigns contribution values for a set of known values, which she obtains either by her expertise, through interviews, from data sheets, or via measurements. To determine the contribution for the missing values, the system will use interpolation functions [22] (e.g., polynomial, spline, cubic).

Take, for instance, task "switch light on". Depending on the light intensity, NFRs energy efficiency, user comfort, and user efficiency receive different contributions. In Fig. 4, the analysts have specified contributions to the three NFRs for different light intensity values (50, 100, 250, 400, 800 lux), based on her own experience and the light bulb data sheet. A spline interpolation has been applied to compute the contribution values for the missing light intensity values.

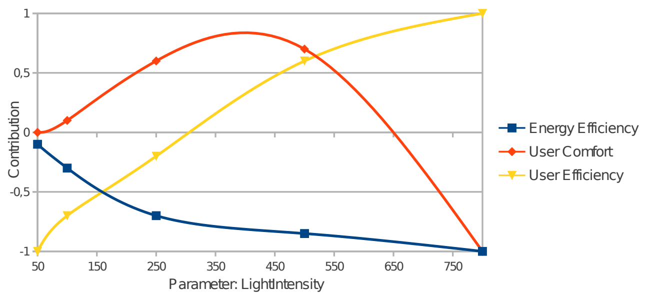

Fig. 4. Interpolation functions to calculate the contributions to NFRs of the parametric task "switch light on"

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

4 Executing Adaptive Task Models

Following Model-Driven Engineering (MDE) guidelines [14], adaptive task models are machine-processable and usable as executable models. At runtime, they become the artefact that drives the adaptive behaviour of a pervasive infrastructure. All the efforts invested at design time are reused at runtime providing new opportunities for adaptation capabilities without increasing development costs.

The adaptation process is activated by triggering events, which define *when* the system should adapt. The following triggers refer to changes affecting adaptive task models that the pervasive infrastructure can monitor at runtime:

1. *Changes in preferences*: a user changes her preferences over NFRs, by altering her contextual preference model. For example, Bob may decide that, due to expensive electricity bills, energy saving has priority over user efficiency;
2. *Task execution faults*: an executable task may return an error, due to either a wrong implementation of a pervasive service 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 each task in the plan has executed correctly. The routine fails in engaging the users in carrying out specific actions. For example, if Bob does not enter the kitchen, task "make coffee" will be inhibited;
4. *Context evolution*: changes in the context modify the applicability and necessity of specific tasks, as well as they affect the priority of NFRs. For example, task "turn on bathroom heating" is unnecessary in summer, when the indoor temperature is influenced by the outdoor heat. Also, if users are not at home, energy efficiency has higher priority than other NFRs.

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

We propose two algorithms that enable the execution of an adaptive task model, i.e. the selection and enactment of a set of tasks to achieve the purpose of the routine. To achieve this, our algorithms rely upon the information contained in our models (Fig. 2). Algorithm 1 explores an adaptive task model in a top-down fashion and selects the best plan to support the routine. A plan consists of a set of executable tasks in the routine and ordering constraints between those tasks that, together, carry out the routine. The best plan is the one that maximizes satisfaction of NFRs for the considered user.

While determining the plan, decisions are taken about (i) whether to execute or skip optional tasks (e.g., when Bob has urgent tasks, the optional task "make coffee" is skipped so that Bob can arrive earlier at work); (ii) the most appropriate task in exclusive refinements (when Bob increases the priority of the NFR energy efficiency, task "raise blinds" replaces task "switch light on"); and (iii) the best parameters' value for 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).

Specifically, Algorithm 1 describes the recursive function RunAdaptTM, which computes the path to follow in an 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

Algorithm 1 Executing an adaptive task model

```
RunAdaptTM(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          RunAdaptTM(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             RunAdaptTM(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 RunAdaptTM(task, pref, ctx)
```

task is not to be executed and the function returns (line 1). Otherwise, the algorithm 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 RunAdaptTM 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 determined by GetContrib (i.e., $\sum_{j \in NFR} Contrib_{task,j} * Weight_j$). 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. RunAdaptTM 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 RunAdaptTM 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 is still holding in the context (line 25), then RunAdaptTM is invoked with the same parameters (line 26).

Example 1 (Executing an adaptive task model). It's a hot Monday of September, and Bob 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 infrastructure executes Algorithm 1 on the adaptive task model of Fig. 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: RunAdaptTM 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. Bob 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);
– Bob is informed about the weather by executing such task (line 4);
– as shown in Sec. 3.3, when Bob has urgent tasks, the optional task "make coffee" is not executed (lines 19-20);
– a transportation mean is suggested. Since Bob 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 ← GetParamMinValue()
 2  maxvalue ← GetParamMaxValue()
 3  for each nfr ∈ pref.GetNFRs()
 4  do Interpolate(minvalue, maxvalue, nfr)
 5  bestvalue ← $-\infty$
 6  bestindex ← null
 7  for each idx : minvalue ≤ idx ≤ maxvalue
 8  do val ← $\sum_{nfr \in pref.\mathrm{GetNFRs}()}$ nfr.GetContribAt(idx) * nfr.GetWeight(ctx)
 9     if val > bestvalue
10        then bestvalue ← val
11             bestindex ← 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

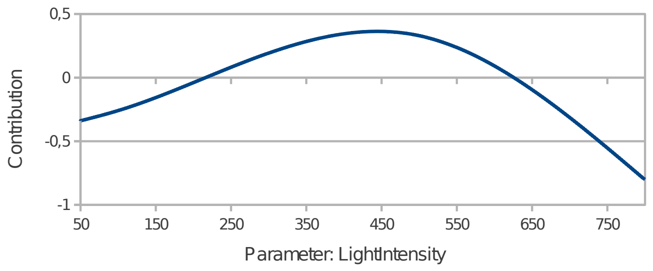

Fig. 5. Weighted sum of the NFRs interpolations for parameter "LightIntensity"

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). Bob does not have urgent tasks, so his 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 inputs, Algorithm 2 takes the interpolations as in Fig. 4 (lines 3-4), sums the interpolated values (line 8) for each point, and defines the best parameter value (lines 7-11). Fig. 5 graphically illustrates the resulting plot. The chosen value is 444, because its weighted contribution to NFRs is the highest (+0.36). □

5 Related Work

The use of NFRs to drive decision-making has been widely explored in Goal-Oriented Requirements Engineering (GORE). Most GORE approaches rely upon (variants of) the NFR framework [13] or the $i*$ framework [26], and exploit the concept of soft-goal to represent NFRs and reason about NFR satisfaction.

Chung et al. [5] use a NFR graph to inform the selection among alternative architectural designs. Approaches in adaptive software systems [18, 9, 12] 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.

Brown et al. [3] 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, Souza et al. [24] define awareness requirements as meta-requirements to drive adaptations. Our framework embodies an adaptation requirement which is globally applied to user routines, i.e. the optimization of user preferences over NFRs.

Some approaches [19, 7, 2] combine Business Process Modelling (BPM) with variability or context models in order to represents variations in processes. However, no approach based on BPM has been proposed to describe user routines.

An interesting research direction is to assess the suitability of BPM languages as an alternative notation to represent user routines.

Other approaches use feature models [4, 1] to describe architectural configurations, each consisting of components that are activated depending on the current context conditions. We consider not only contextual factors, but also user preferences over NFRs in order to adapt the system. Also, our work is not focused on the activation of individual system components, but rather on the adaptation of complex system behaviours (user routines).

6 Conclusions and Future Work

We have proposed adaptive task models, an executable modelling language that a pervasive infrastructure can use to support users in their daily routines. These models do not only support alternative ways to carry out a user routine, but also include the decision-making rationale. Moreover, 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.

We have also devised an adaptation framework that exploits adaptive task models at runtime to drive and adapt system behaviour. The novelty of our proposal lies in the enactment of a user-centric adaptation process by modelling and reasoning about Non-Functional Requirements (NFRs): the system, while automating user routines, adapts its behaviour to choose a course of action that maximizes user preferences over NFRs in the current context.

In future work, we plan to develop a pervasive software infrastructure that implements the proposed adaptation framework. To do so, we will build on top of our previous work on pervasive infrastructures [21] and goal-oriented adaptation [6]. The infrastructure will be applied to pervasive case studies to assess its effectiveness in guiding users. Moreover, we will extend our framework to include historical data about (un)successful executions in order to adapt the system to a more successful behaviour. Finally, we will devise methodological guidance to help the analysts in building the models the system needs at runtime.

References

1. M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J. P. Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *Proc. of Models@run.time 2009*. LNCS, 2009.
2. C. Ayora. Modelling and Managing Variability in Business Process Models, Ms.C. Thesis, ProS, Universidad de Valencia. 2011.
3. G. Brown, B. H. C. Cheng, H. Goldsby, and J. Zhang. Goal-oriented Specification of Adaptation Requirements Engineering in Adaptive Systems. In *Proc. of SEAMS '06*, pages 23–29. ACM, 2006.
4. C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *IEEE Computer*, 42:37–43, 2009.

5. L. Chung, B. Nixon, and E. Yu. Using Non-Functional Requirements to Systematically Select among Alternatives in Architectural Design. In *Proc. of IWASS'95*, page 3143. ACM, 1995.
6. F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Adaptive Socio-Technical Systems: a Requirements-driven Approach. *Requirements Engineering*, 2012. To appear.
7. J. L. de la Vara, R. Ali, F. Dalpiaz, J. Sanchez, and P. Giorgini. COMPRO: A Methodological Approach for Business Process Contextualisation. In *Proc. of CoopIS 2010*, pages 132–149, 2010.
8. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with Goal Models. In *Proc. of ER 2002*, pages 167–181, 2002.
9. H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hugues. Goal-based Modeling of Dynamically Adaptive System Requirements. In *Proc. of ECBS 2008*, pages 36–45. IEEE, 2008.
10. M. L. Griss, J. M. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB. In *Proc. of ICSR'98*, pages 76 –85, jun. 1998.
11. K. Henricksen and J. Indulska. Developing Context-aware Pervasive Computing Applications: Models and Approach. In *Pervasive and Mobile Computing*, volume 2, pages 37–64, 2004.
12. A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos. Requirements-driven Design of Autonomic Application Software. In *Proc. of CASCON 2006*, 2006.
13. J. Mylopoulos, L. Chung, and B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.
14. O. Pastor and J. C. Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling.* Springer-Verlag, 2007.
15. F. Paternò. ConcurTaskTrees: An Engineered Approach to Model-based Design of Interactive Systems. In *The Handbook of Analysis for Human-Computer Interaction*, pages 483–500. Lawrence Erlbaum Associates, 2002.
16. C. J. Pavlovski and J. Zou. Non-Functional Requirements in Business Process Modeling. In *Proc. of APCCM 08*, page 103112, 2008.
17. A. Perini, F. Ricca, and A. Susi. Tool-supported Requirements Prioritization: Comparing the AHP and CBRank methods. *Information and Software Technology*, 51(6):1021–1032, 2009.
18. M. Salehie and L. Tahvildari. Towards a Goal-driven Approach to Action Selection in Self-adaptive Software. *Software: Practice and Experience*, 42:211–233, 2012.
19. E. Santos, J. Pimentel, D. Dermeval, J. Castro, and O. Pastor. Using NFR and Context to Deal with Adaptability in Business Process Models. In *Proc. of RE@RunTime*, 2011.
20. M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.
21. E. Serral, P. Valderas, and V. Pelechano. Supporting Runtime System Evolution to Adapt to User Behaviour. In *Proc. of CAiSE'10*, pages 378–392, 2010.
22. D. Shepard. A Two-dimensional Interpolation Function for Irregularly-spaced Data. In *Proc. of the ACM national conference*, pages 517–524, 1968.
23. A. Shepherd. *Hierarchical Task Analysis.* Taylor & Francis, London, 2001.
24. V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness Requirements for Adaptive Systems. In *Proc. of SEAMS '11*, pages 60–69, 2011.
25. A. Sutcliffe, S. Fickas, and M.M. Sohlberg. Personal and Contextual Requirements Engineering. In *Proc. of RE 2005*, pages 19–28, 2005.
26. E. Yu. *Modelling Strategies Relationships for Process Reengineering.* PhD thesis, Department of computer science, University of Toronto, 1995.