ProDSPL: Proactive Self-Adaptation based on Dynamic Software Product Lines

Inmaculada Ayala[a,b,*], Alessandro V. Papadopoulos[c], Mercedes Amor[a,b], Lidia Fuentes[a,b]

[a] *Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain*
[b] *ITIS Software, Universidad de Málaga, Spain*
[c] *Mälardalens högskola, Sweden*

Abstract

Dynamic Software Product Lines (DSPLs) are a well-accepted approach to self-adaptation at runtime. In the context of DSPLs, there are plenty of reactive approaches that apply countermeasures as soon as a context change happens. In this paper we propose a proactive approach, ProDSPL, that exploits an automatically learnt model of the system, anticipates future variations of the system and generates the best DSPL configuration that can lessen the negative impact of future events on the quality requirements of the system. Predicting the future fosters adaptations that are good for a longer time and therefore reduces the number of reconfigurations required, making the system more stable.

ProDSPL formulates the problem of the generation of dynamic reconfigurations as a proactive controller over a prediction horizon, which includes a mapping of the valid configurations of the DSPL into linear constraints. Our approach is evaluated and compared with a reactive approach, DAGAME, also based on a DSPL, which uses a genetic algorithm to generate quasi-optimal feature model configurations at runtime. ProDSPL has been evaluated using a strategy mobile game and a set of randomly generated feature models. The evaluation shows that ProDSPL gives good results with regard to the quality of the configurations generated when it tries anticipate future events. Moreover, in doing so, ProDSPL enforces the system to make as few reconfigurations as possible.

Keywords: Dynamic Software Product Lines, Proactive Control, Self-Adaptation, Optimization, Linear constraint

1. Introduction

The demand for self-adapting software systems has risen sharply, specially motivated by the growing importance of Cyber-Physical Systems (CPS) operating in complex and changing environments, and the increasing ubiquity possibilities of Information and Communication Technologies (ICT) [1]. Designing self-adaptive systems is a complex task, and has been of great interest in the last years, specially in areas such as mobile computing, robotics or ubiquitous computing [2]. Runtime adaptation mechanisms sometimes are problematic since they tend to be unstable, inefficient and unreliable [3]. Therefore, in order to overcome some of these problems several approaches have been defined [4, 5, 6], being the Dynamic Software Product Lines (DSPL) one of the most widely used. Dynamic Software Product Lines (DSPL) are a systematic engineering approach that uses SPLs (Software Product Lines) artifacts to model the dynamic variability of a system. DSPLs model the different adaptations of the system to context changes, either internal or external, as variation points. Examples of context variations that can trigger a system adaptation can be, the continuous variation of resource availability (e.g., battery or memory), the occurrence of system or environment failures, and also the fluctuations of user's needs [7].

Adopting a DSPL approach requires specifying the dynamic variation points as part of a variabil-

*Corresponding author
Email addresses: ayala@lcc.uma.es (Inmaculada Ayala), alessandro.papadopoulos@mdh.se (Alessandro V. Papadopoulos), pinilla@lcc.uma.es (Mercedes Amor), lff@lcc.uma.es (Lidia Fuentes)

ity model and a reconfiguration service that performs the system adaptation when necessary. One of the most popular variability models is the feature model (FM), which, in the context of DSPLs, is used to specify a variability space model of a single system at runtime, without needing to enumerate all the possible system configurations. The reconfiguration service determines if the system needs to be adapted to satisfy the quality requirements under different execution contexts. Then, the reconfiguration service, containing the DSPL artifacts, will be continuously monitoring the context and finding out the best possible configuration after each context change at runtime. This new configuration should be the optimal one with respect to an objective criterion (e.g., minimize battery consumption while maintaining a prescribed quality of service). There are many reasoning techniques to select the suitable variant of a system, considering the current configuration, the context change and the DSPL artifacts.

Independently of the reasoning approach used to find the new configuration, DSPLs approaches implement decision strategies that are purely reactive: during the system execution the context is continuously monitored and when a context change occurs, the reconfiguration service react to this change by analysing if there is another configuration that fits better the current conditions trying to find an optimal or quasi-optimal solution for the current context [8]. In cyber-physical systems the strategy used to generate the new optimal or quasi-optimal configuration should be implemented at runtime, since the high number of unexpected context changes makes impossible to pre-load all possible reconfiguration plans. Dynamic strategies then require the generation of a valid configuration at runtime, and adapting the system accordingly, which can be both time- and resource-consuming tasks (in terms of energy, memory, computation, etc.) that can negatively affect the performance of battery-powered systems.

Therefore, when analysing different DSPL alternatives, the adaptation cost should also be considered in terms of resources involved in reconfiguration, such as energy consumption and also the time needed to analyse and perform an adaptation. Indeed, in some scenarios, the adaptation process can consume a significant amount of execution time. For example, in Wireless Sensor Networks, where most of the devices have usually limited computational resources, the cost of the reconfiguration is usually too high as it contributes to a faster degradation of the device batteries. In medical applications, it is simply not acceptable to stop–or delay– the normal functioning of the system for a reconfiguration, specially if there is a serious problem that should be fixed urgently. For this kind of scenarios, the preferred techniques are those that lessen the number of reconfigurations, but trying to maintain a good quality of service at the same time. Also, performing continuous adaptations makes the system more unstable, because, if the reconfiguration service makes the system jump from one configuration to another, this might reduce the user quality of experience. Then, in these situations, it would be preferable to adopt a *proactive strategy* in order to reduce the number of adaptations, making the system more stable. Proactive strategies consider not only the current context, but also the system expected evolution over time, so their adaptation solutions tend to stay valid for longer. But the price to pay when we apply a proactive strategy is that sometimes the calculated application configuration has a lower utility or quality compared to the optimal solution or to the solution provided by some reactive approaches. Therefore, the great challenge is to find a proactive strategy whose adaptation solutions are good enough to increase the lifespan of the system while reducing the number of required reconfigurations. By decreasing the number of runtime reconfigurations, the overall adaptation cost is reduced and also the system is more stable.

In this work, we propose ProDSPL, that is a self-optimizing approach that combines DSPL with a control-based proactive decision strategy for dynamically generating optimal configurations in a proactive way. Self-optimizing systems based on Proactive Controllers have been explored in the context of software systems adaptation with successful results [9, 10, 11], and in this work we explore how well it behaves in conjunction with DSPL artifacts.

ProDSPL formulates the problem of dynamic reconfiguration as a proactive controller (PC) that generates valid configurations of a DSPL at runtime. In the control theory field, this type of adaptation, which is known as Model Predictive Control (MPC), relies on dynamic models learnt from the system observation, and comes with a well-developed theory and myriads of successful applications [12].

The main contribution of this paper is the formulation of the DSPL reconfiguration service as a

single-objective optimization problem over a prediction horizon subject to the linear constraints of the DSPL feature model following a proactive approach. This formulation is based on a mapping between extended feature models and linear constraints, which is part of our solution to combine proactive control with DSPL. Although previous works have proposed this kind of mapping [13, 14], they only consider basic feature models. Feature models of this kind do not support numerical features and consider only simple cross-tree constraints (i.e., include or exclude), so they cannot be applied in several real case studies.

Both the time required to generate the configurations and the quality of the solutions found are crucial aspects for the success of a DSPL-driven reconfiguration at runtime. So, the performance and quality of the results obtained by the proactive controller ProDSPL are compared with DAGAME [5], a reactive approach that uses a DSPL approach with a genetic algorithm for the generation of feature model configurations at runtime. Taking into account the concept of optimality [15] (i.e., the ratio between the utility of the solution obtained by an algorithm and the utility of the optimal solution obtained using the exact method), DAGAME is able to obtain solutions with an optimality higher than 87.4% and execution times between 20 and 100 milliseconds. In this paper, ProDSPL performance is evaluated using a simulator of a mobile app developed in Java, in order to make experiments and results reproducible by third parties. The comparison with the purely reactive approach shows good results with regard to the performance and the quality of the configurations generated. The results obtained support in practice our initial hypothesis about the use of a proactive approach. That is, **our solution is more sustainable as it contributes to reduce the overall cost–in time and energy–of reconfiguration and leads to more stable systems, while maintaining a satisfactory quality of adaptation**.

The remainder of the paper is organized as follows: Section 2 provides background on DSPLs, the decision-making process, and the model predictive control. Section 2 overviews the main activities of our approach, ProDSPL. Section 4 illustrates how to apply our proposal using a Mobile Game as a case study. The experimental results are presented, analyzed and compared with DAGAME in Section 5, while the threats to the validity of our study are discussed in Section 6. Section 7 discusses related work. Finally, Section 8 presents some conclusions to the paper.

2. Background

2.1. Dynamic Software Product Lines

DSPL is an approach to self-adaptation that takes concepts from the domain of SPL, including the management of variability through a model. DSPLs redefine existing SPL engineering processes by moving them to runtime, with the goal of adapting the system to the current environment by performing reconfiguration autonomously. While SPLs are able to generate several systems of the same family at design time, a DSPL is able to adapt a single system behaviour at runtime.

Variability of SPLs can be specified using different modelling languages, being feature models [16] the most popular. Since its conception, a lot of notations and extensions have been defined for feature models [17]. A feature model contains an explicit representation of the configuration space by means of features. An FM organizes features into a tree, and includes the corresponding tree and cross-tree constraints representing dependencies among features. In DSPL, the system elements that can be reconfigured are modelled as *dynamic variation points*, while the set of selected features that fits the current context is known as *dynamic configuration*. In DSPLs, optional features will be included or not in the system at runtime, depending on how the context conditions affect the application during its execution.

With this aim, and as part of a DSPL definition, the engineer must [18][19, 7] (i) identify the range of potential adaptations supported by the system in terms of architectural components that can be deleted, replaced or added at runtime and the software components parameters that could vary so that the system meet some quality requirements; (ii) define an explicit representation of the valid configuration space of the system that can be included as part of the running system; (iii) identify the context changes that may trigger an adaptation; (iv) identify the set of possible reactions to context changes that should be supported by the system; and (v) define a decision-making process (DMP) that fulfils some optimization goals (one or many) by choosing the best DSPL configuration that fits current context.

However, the way these issues are implemented may differ greatly between different proposals. The main difference lies in when the dynamic valid configurations are computed. Some approaches compute all the possible reconfiguration at design-time [20] and upload all or a subset of them at runtime [21]. Other approaches are able to generate these configurations at runtime [18]. The main advantage of the latter is all the possible configurations are considered and not only a subset, but it is difficult to find a DMP capable of generating reconfiguration plans efficiently in an acceptable computation time at runtime.

2.2. Decision-making process in DSPLs

Then, one of the key issues of DSPLs approaches is the decision-making strategy and the objective optimization techniques applied to generating optimal dynamic configurations.

The authors in [8] offer an overview of different DSPL approaches for self-adaptation and their decision-making processes. According to [8], most of the DMP for dynamic reconfiguration approaches can be classified into three major optimization categories: (i) randomized stochastic approaches; (ii) exact solution approaches; and (iii) learning-based approaches. A common feature of DSPLs approaches is that the adaptation follows a reactive strategy: during the system execution, when the context changes occurs, the reconfiguration service analyses whether this change requires or not replacing the current configuration by a new one that fits better the current context conditions.

Then, in reactive approaches, adapting the system to context changes should be done after a change is detected. Hence, the reconfiguration service needs a mechanism able to generate new optimal configurations as fast as possible at runtime, which is a challenging task. Usually, the input of a DMP takes as input the current configuration, the event that triggered an adaptation, the DSPL artifacts and an optimization goal according to system and user requirements. All this information from the past must be available at runtime.

2.3. Model Predictive Control

The idea of proactivity in the adaptation of software systems has been explored in the past few years by different approaches, ranging from hidden Markov chains [22, 10] to dynamic systems [11], with the aim to forecast the future behaviour of the system and of the environment. Model Predictive Control (MPC) [12, 23] is an important advanced control technique for multivariable control problems that makes an explicit use of a reasonably accurate dynamic model to predict the system output at future time instants (prediction horizon). The model is the main element of the controller as it has to capture the system dynamics to be able to calculate the predictions. Model predictive control offers several important advantages for supporting the decision-making strategy in DSPLs: (1) the process model captures the dynamic and static interactions between input, output and disturbance variables; (2) constraints on inputs and outputs are considered in a systematic manner; (3) the control calculations can be coordinated with the calculation of optimum set points; and (4) accurate model predictions can provide early warnings of potential problems.

2.4. Case study: Strategy Mobile Game

Our illustrative case study is a strategy video game for a mobile phone taken from [18]. In strategy games, the player controls multiple elements in a large game world (e.g., a game board). These games are more delay-tolerant than action and racing games [24, 25], having latency requirements similar to those of Web browsing (in the order of seconds) [26, 27].

In this strategy game for mobile devices some characteristics can be configured at runtime (by adding or deleting the corresponding software components, or tuning up the values of their parameters) to adapt the game to the context, i.e., the user preferences or the device resource availability (e.g., battery, memory, etc.).

The player can configure graphics quality and sound according to their preferences. The user can select between using 2D or 3D graphics. Internally, graphics quality is also determined by the level of detail (low, medium or high) and the frame rate (in frame per second–fps) of the video. In this case, a high graphics quality (3D graphics, high detail level or high frame rate) requires more resources (CPU, battery, memory). As part of the audio settings, the user can mute the game sounds or use vibration instead. When the sound is on, its quality can be configured by means of the bit rate in kbps. The higher the bit rate is, the more resources the sound reproduction will consume on the mobile device.

The game offers an online and a local multiplayer mode to challenge other players, offline or

in the same local or personal area network. Players activity may or may not be organized in turns. Regarding connectivity, the application can make use of different network interfaces and technologies supported by the mobile device, such as WiFi (WLAN), LTE (4G) and HSPA (3G), but also Bluetooth, which is used to support the interaction with other players in the local multiplayer mode. Real-time multiplayer games require low response times, but strategy games can support round-trip delays of 1–3 seconds without ruining the gameplay. Anyway, latency should be under 5 seconds [24, 25]. A WiFi connection has the shortest response time and the fastest transfer speed, providing the best gamer experience, but its energy consumption is the highest and its availability and speed are limited by the location of the user in the coverage area of the WLAN. The use of LTE is suited for multiplayer real-time games when WiFi is not available or offers a poor coverage, but LTE battery consumption is higher than WiFi and its use may involve additional costs from the ISP company.

3. The ProDSPL approach

We propose ProDSPL to drive the decision-making strategy of a DSPL. ProDSPL is based on the idea of a proactive control approach, and formulates the adaptation problem as an optimization problem over a prediction horizon subject to the linear constraints of the DSPL.

The proposed approach exploits an automatically learnt model of the system, which captures how the system reacts to different feature model alternatives over time, in order to predict what is the impact of the chosen dynamic configurations on the system requirements, based on the available resources (e.g., battery). A distinctive characteristic of our approach is that, at any time, ProDSPL accounts for the current effect and also predicts the future impact of the reconfiguration on the behaviour of the system for a specific period of time (i.e., a look-ahead horizon). This might lead to decide whether to adapt or reconfigure the DSPL by means of solving an optimization problem to select the valid product configuration that maximizes an objective function over a finite look-ahead horizon.

A schematic overview of ProDSPL is shown in Figure 1. ProDSPL is based on the iterative solution, i.e., it periodically recomputes a new solution, by searching through a finite-horizon optimization problem that is solved at every iteration. At design time, we need to define the DSPL feature model including the tree and cross-tree constraints. The optimization problem must include such constraints among features–to compute a feasible feature configuration–and a model that maps the selection of features to the relevant Key Performance Indicators (KPIs). The model is automatically learnt from logged data, and it can possibly be changed at run-time in the case of major changes in how the features map into the KPIs occur. Also, we need to decide the objectives of the DSPL in terms of performance that ProDSPL will try to maximize during system execution. At runtime, the *Decision-Making Strategy* component determines if there is a configuration that fits better the current context, by periodically solving the optimization problem, not just when a sudden change occurs, but whenever the reconfiguration service detects a variation in the *performance indicators*. Periodically, a better solution that *maximizes the accumulated performance* and that fulfils the system constraints is sought, and if a new solution is found, ProDSPL replaces the current configuration by the new one.

The rest of this section describes how to design the extended feature model, how the learnt model is calculated through a mathematical model, and how the feature model is transformed into linear constraints. This is done once, at design time before running the system. Section 4 describes how the approach is applied to an illustrative case study.

3.1. Variability model design

The first phase is to design the **variability model** of the DSPL in the form of an extended feature model.

In this work, to specify dynamic variability we use extended feature models, which are basic feature models with variables, cardinality groups, and arithmetic and logical constraints. In order to generate a new dynamic configuration, the features of the feature model are resolved taking into account its specific type (e.g., choices are selected or not, values are given to variables...).

The extended feature model, including cross-tree constraints, defines what the feasible reconfiguration solutions are. The use of feature models to specify dynamic variation points will ensure that system adaptations lead the system to a valid state, i.e., it will restrict the solution space to runtime valid configurations. The extended feature model is one of the two main artifacts of ProDSPL (see

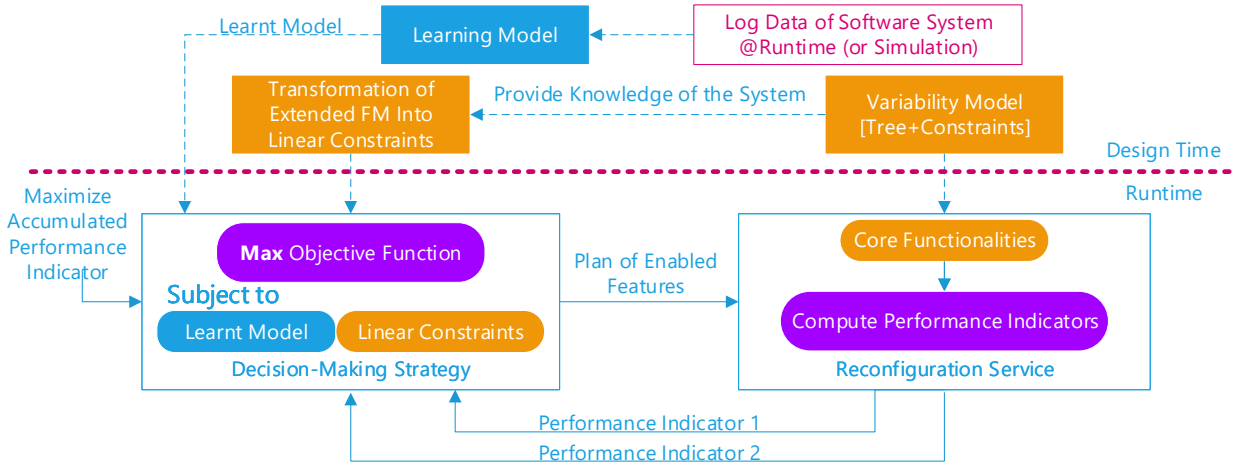

Figure 1: Overview of the ProDSPL approach

right part of the Design Time of Figure 1). Figure 2 in Section 4.1 shows the extended feature model of our illustrative case study, the strategy mobile game. This model comprises all those features of the game whose activation/deactivation or tuning affect battery consumption or user experience. These include connectivity (WiFi, LTE, Bluetooth), playing modes, graphics and sound quality, etc. In addition, cross-tree constraints will be used to express (1) that sound quality ranges between 128 and 256 kps when the gamer is playing in local multiplayer mode; (2) the connection used can be WiFi or Bluetooth; and (3) that showing and updating the global score board require network connectivity.

3.2. Learning the system model

ProDSPL is a model-based approach that exploits a mathematical model of the system performance to capture how the system reacts to different choices of the features over time. The definition of an accurate model of the system is far from being trivial. However, the main aim of the model in the optimization problem is not to provide an accurate mapping between the features and the KPIs, since given a set of selected features it is sufficient to capture the main trend of their effect to support the decision-making strategy. For example, it is sufficient to capture that enabling LTE will consume more battery than selecting WiFi for the connectivity of an application. As a result, ProDSPL includes a learning phase, where the model of the optimization problem is computed from experimental data. To learn such model, we use a non-iterative subspace identification approach [28], widely adopted in the learning community, and successfully used in control of software systems [29], implemented in Matlab with the `ssest` function.

To consider the current and anticipated adaptation needs of the system, ProDSPL learns a model $\mathcal{M}$ (i.e., the MPC model) that captures the dynamics of the system (see Figure 1). Considering that we are modelling the dynamic variation points with extended feature models, the model of the system has to capture how the features $\mathbf{u} = \{u_1, u_2, \ldots, u_m\}$ affect the relevant performance indicators $\mathbf{y} = \{y_1, y_2, \ldots, y_p\}$. This model can be either manually designed, based on some prior knowledge of the system, or automatically generated from logged data obtained from the executions of the software system, or from a simulator of the system. The model can be computed by means of learning techniques [28]. In particular, we follow the technique presented in [29], in which the identified model $\mathcal{M}_t$ at a specific time instant t will be in the form:

$$\mathbf{x}(t+1) = A\mathbf{x}(t) + B\mathbf{u}(t), \quad (1)$$

$$\mathbf{y}(t) = C\mathbf{x}(t), \quad (2)$$

where $\mathbf{u}$ is the vector of control parameters (i.e., the features), $\mathbf{y}$ is the set of performance indicators that highlights how the system is performing, and A, B and C are matrices learnt from the logged data. The model is used to predict the future behaviour of the system, by "unrolling" the equation (1) over H time steps (called *prediction horizon*). For example,

for time steps $t = 1, 2$:

$$\mathbf{x}(1) = A\mathbf{x}(0) + B\mathbf{u}(0),$$
$$\mathbf{x}(2) = A\mathbf{x}(1) + B\mathbf{u}(1) = A^2\mathbf{x}(0) + AB\mathbf{u}(0) + B\mathbf{u}(1).$$

If the dynamic model is unrolled for a generic time $t = H$ [12], the unrolled dynamics can be expressed as:

$$\mathbf{x}(H) = A^H\mathbf{x}(0) + \sum_{i=1}^{H} A^{H-i}B\mathbf{u}(i-1).$$

This makes possible to compute the value of all the future outputs y as a function of the current state of the system ($\mathbf{x}(0)$) and of the possible changes determined by the feature model $\mathbf{u}(0)$, $\mathbf{u}(1)$, ..., $\mathbf{u}(H-1)$, which must be also computed. This computation is performed considering, that $\mathbf{u} = \{u_1, u_2, \ldots, u_m\}$ is the set of the m features included in the DSPL that are either binary or have a numerical value. The identified learnt model can be adapted at runtime, based on the current behaviour of the system, so the accuracy of this initial model does not have to be necessarily very high [28]. In fact, this model serves for understanding the complex relationships between the features u and the performance indicators y, in order to take sensible decisions of what features must be included in the runtime configuration.

3.3. Transformation of the variability model into linear constraints

In order to compute a plan of what are the features u that must be enabled during the prediction horizon H, the description of the extended feature model must be transformed into constraints that can be used by a numerical solver (see left part of Design Time of Figure 1). The extended feature model and the cross-tree constraints are reformulated in terms of a set of linear constraints $\mathcal{C}$. This procedure has to be done carefully, in order to introduce only linear constraints, which are usually simpler to handle.

Even though the use of just linear constraints seems to limit the applicability of the proposed approach, this is not true. Below, we present general rules that can be used to transform complex generic structures of extended feature models into linear constraints. The transformation of a feature model into a set of linear constraints has been approached in previous contributions [13, 14]. However, these works do not consider the transformation of extended feature models. They just consider simple feature models, i.e., without numerical attributes, propositional constraints or groups with customised cardinalities. The transformation is performed from the following mapping between feature models and linear constraints:

- **Paternity:** Let p be the parent and c the child in parent-child relation, then the equivalent constraint is $u_c \leq u_p$.
- **Mandatory:** Let p be the parent and c the child in a mandatory relation, then the equivalent constraint is $u_p \leq u_c$.
- **Groups:** Let $\{u_1, u_2, \ldots, u_k\}$ be a group of features in a group relation which allows to choose between n and m elements of the group, being $n \leq k \leq m$, and p the parent feature of this group, then the equivalent constraints are $\sum_{i=1}^{k} u_i \geq nu_p$ and $\sum_{i=1}^{k} u_i \leq mu_p$.

The mappings above cannot be applied when a feature (e.g., printer) has a numerical value associated. We consider these features as *numerical features* that can be bound to a variable value (e.g., a high resolution printer has a value that can range from 100 to 300 ppi). If one of the features has a positive value, then the inequality expressions will not hold. In order to incorporate numerical features in our decision-making strategy, we follow a two step solution. Firstly, we normalize the range of values that can be taken for the numerical feature. This normalized range starts with 0, associated with the decision of not selecting the feature in the resolution model (e.g., not selecting the printer feature). Therefore, if the original range of values of a variable is $[LB, UB]$, the normalized range is $[0, UB - LB + 1]$.

Secondly, a binary value $y_i \in \{0, 1\}$ associated with u_i is introduced to decide if u_i should be enabled, and to introduce "big M" constraints for the groups of features. The mapping for variables with normalized ranges will be as follows:

- **Paternity:** Let p be the parent and c the child in parent-child relation. If the size of the range of c is higher than the size of the range of p, then the equivalent constraint is $u_c \leq (UB_c - LB_c + 1)u_p$ where LB_c and UB_c are the lower and the upper bounds of c.
- **Mandatory:** Let p be the parent and c the child in a mandatory relation. If the size of

the range of p is higher than the size of the range of c, then the equivalent constraint is $u_p \leq (UB_p - LB_p + 1)u_c$, where LB_p and UB_p are the lower and the upper bounds of p.

- **Groups:** Let $\{u_1, u_2, \ldots, u_k\}$ be a group of features in a group relation which allows to choose between n and m elements of the group, being $n \leq k \leq m$, then the equivalent constraints are $u_1 \leq Mz_1$, $u_2 \leq Mz_2$, ..., $u_k \leq Mz_k$, with $\sum_{i=1}^{k} z_i \geq n$ and $\sum_{i=1}^{k} z_i \leq m$ where $z_1, z_2, \ldots, z_k \in \{0, 1\}$ are auxiliary binary variables introduced for the formulation of the linear constraints, and M is a constant large number (ideally, $M \to \infty$).

Additionally, we can express propositional logic constraints and arithmetic constraints using integer linear programming constraints. There are several works that propose mappings between propositional logic and integer linear constraints [30, 31, 32, 33]. Table 1 summarizes some of these mappings. Some kinds of extended feature models supports the use of quantifiers (i.e., $\forall$ and $\exists$) for the definition of cross-tree constraints. However, they are applied to cardinality-based feature models whose use for DSPL has not been explored. We intend to investigate this as future work.

3.4. Formulation of the decision-making strategy

The decision-making strategy (see left part of Runtime of Figure 1) is formulated as an optimization problem that takes the form:

$$\begin{aligned} \underset{\mathbf{u}(1),\ldots,\mathbf{u}(H)}{\text{maximize}} \quad & \sum_{i=1}^{H} F(\mathbf{u}(i)) \\ \text{subject to} \quad & \mathcal{M}_t, && t = 1, \ldots, H \\ & \mathcal{C}_t, && t = 1, \ldots, H, \\ & \mathbf{y}(0) = \mathbf{y}_{\text{measured}}, \end{aligned} \tag{3}$$

where the last constraint initializes the optimization problem based on the last measured value of the performance $\mathbf{y}_{\text{measured}}$, and $F(\mathbf{u}(i))$ is an objective function that must be maximized over the given prediction horizon. One can think of the function F as the (instantaneous) utility function associated with the selected features $\mathbf{u}(t)$ at time t. If no numerical features are present, classical 0-1 programming solvers can be used for solving (3) [34].

The output of the optimization problem is a long-term plan over the prediction horizon H. ProDSPL applies the so-called "receding horizon principle" [12], i.e., only the first action $\mathbf{u}(1)$ of the long-term plan is applied, and, at the next control period, a new plan is generated by solving the new optimization problem, initialized with the new measured output $\mathbf{y}_{\text{measured}}$. In our case, the first reconfiguration action of the plan is the runtime application configuration (including numerical features) that fits not only the current context, but also the expected behaviour of the system.

ProDSPL has both a proactive and a reactive nature. It is proactive, thanks to the exploitation of the model in the creation of the plan for the future. ProDSPL is also reactive, since the solution of the optimization problem is re-assessed periodically; in practice, different factors, e.g., environmental conditions, distance from routers, etc., may affect the KPIs without being accurately captured by the model, but the periodic re-assessment allows for a re-alignment of the model with the current state of the real system. As such, the reactive nature of the proposed approach compensates also for potential model inaccuracies.

The quality of the resulting decision-making strategy is directly linked to the selection of the period, which is a design parameter of ProDSPL. Such a design choice is rather common for control systems [12], where a lower period results in a tighter and higher quality control. However, more frequent control actions have the drawback of a higher overhead in terms of required computation. Such a design choice, therefore, is often problem dependent, since it is a function of the available computational capacity, and of the level of accuracy required by the application.

Since $\mathcal{M}_t$ is selected to be a linear model, and $\mathcal{C}_t$ is a set of linear constraints introduced by the variability model, all the constraints of the optimization problem are linear. Therefore, if $F(\mathbf{u}(i))$ is chosen to be linear with respect to the decision variables $\mathbf{u}(i)$, then the formulated control problem is an integer linear programming problem [35]. Thanks to the linearity of the approach, ProDSPL will converge to the same solution until the quantity of resources are critical. For example, if there is enough battery (e.g., at least 20%), the proposed approach will continuously select the configuration that best optimizes the user experience in the game. When the battery gets close to the threshold level, the selection of the features become more critical, and the focus of the optimization problem will be on the minimization of the battery drainage while

Table 1: Variable transformation

Statement	Constraint
$\neg P_1$	$x_1 = 0$
$P_1 \vee P_2$	$x_1 + x_2 \geq 1$
$P_1 \rightarrow P_2$	$x_1 \leq x_2$
$P_1 \neq P_2$	$x_1 + x_2 = 1$
at least k out of n are TRUE	$x_1 + x_2 + \ldots + x_n \geq k$
exactly k out of n are TRUE	$x_1 + x_2 + \ldots + x_n = k$

keeping a satisfactory level of user experience. This behavior of ProDSPL is obtained thanks to the periodic re-assessment of the current KPIs and the subsequent recomputation of a (possibly) new plan.

4. ProDSPL in action

In this section, we illustrate how our proposal is applied for adaptation at runtime using as a case study the strategy mobile game presented in Section 2.4.

4.1. Derivation of the feature model

The first step of a developer that wants to apply our approach is to derive the extended feature model of the system to adapt. Figure 2 shows the extended feature model of the strategy mobile game. The root feature, $MobileGame$, is decomposed in features that model the dynamic variation points that can affect energy consumption and user experience. Derived from the game description (in section 2.4), the model includes the following features in a first level of decomposition: *Sound*, *Connectivity*, *GraphicsQuality*, *GlobalScoreboard* and $Multiplayer$. A feature can be bound to its parent by a solid or a dashed line. In the first case, it means that, if the parent has been selected (i.e., the parent feature is true), a value has to be assigned for that feature too. Secondly, a dashed line means that, if the parent has been evaluated as true, it is not mandatory to decide a value for this feature. While the *GraphicsQuality* and *GameMode* features are mandatory and thus have to be included in all the generated configurations, the other features are optional. For instance, if *Sound* is selected, it is not necessary to decide a value for $Vibration$. The Connectivity and Bluetooth features are part of an OR group, meaning that one or both of them can be selected simultaneously. *Low*, $Medium$ and $High$ features (children features of *Graphics* feature), are in an XOR group, and thus exactly one of them can be part of a particular configuration. This model includes two different kinds of features: *choices* and *variables*. *Choices*, which are depicted in the figure as rectangles (e.g., *Sound*), are evaluated as true or false. They correspond to features $u_i \in \{0, 1\}$. On the other hand, *variables* (e.g., $frameRate$), which are depicted as ovals, can be evaluated as values of different types (e.g., numerical values). For the running case study, all the variable features are evaluated as positive integer numbers, i.e., $u_i \in \mathbb{N}$.

Feature models allow the specification of cross-tree constraints in order to delimit the degree of variability. These constraints are defined as relationships between different features of the model. The grey box of Figure 2 shows the constraints C of our case study. For instance, since having a global score board requires to have a network connection, we include $C1$, that states that, if *GlobalScoreBoard* is selected in the resolution model, it is mandatory to include (i.e., select) the feature $Network$. It is possible to specify constraints involving the values of a variable feature. For instance, constraint $C5$ states that the value of feature $frameRate$ ranges between 40 and 60. If the $frameRate$ feature is selected, its value must comply $C5$. $C4$ states that, in the case that the OnlineMultiplayer feature is selected, it is necessary to select the HSPA, LTE or WiFi features. If LocalMultiplayer is selected, WiFi or Bluetooth feaures are required to be selected ($C3$). Taking into account tree and cross-tree constraints, the presented extended feature model has 1,804,194 possible configurations. Note, that this is the number of the possible valid configurations that the system can take during runtime, so this case study can be considered sufficiently representative [5].

4.2. Linear constraint derivation

Once the extended feature model of the case is derived, it is reformulated as a set of linear con-

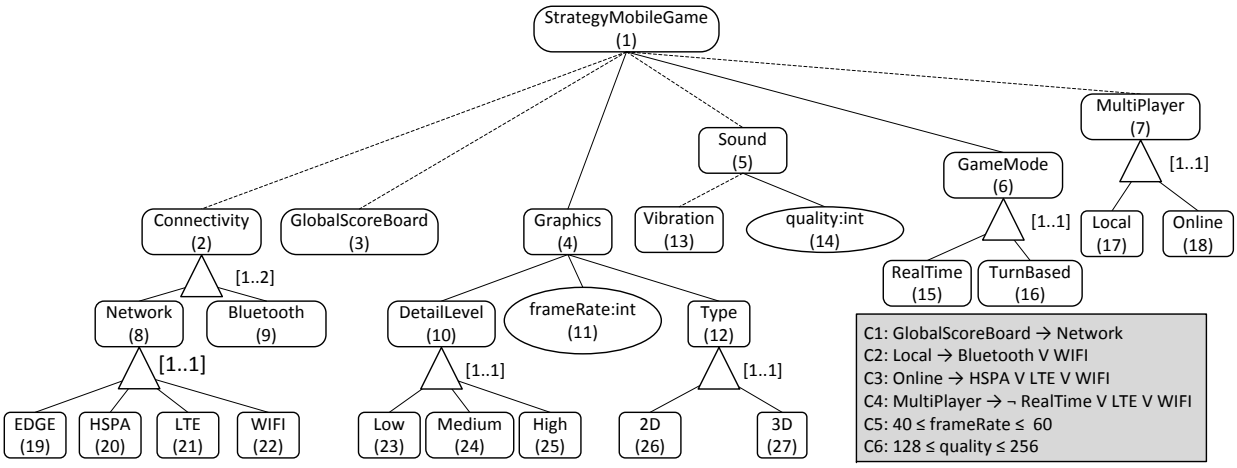

Figure 2: Extended feature model of the Strategy Mobile Game

straints $\mathcal{C}$ needed for the optimization problem. This transformation procedure follows the guidelines provided in Section 3.3. We introduce the constraints on the variables u_i where i is the number of the feature in Figure 2. Additionally, we have to consider the special mappings for variable features (i.e., the case of u_{11} and u_{14}). Therefore, the resulting paternity constraints are:

$$\begin{aligned}
&u_i \leq u_1, && i = 2, \ldots, 7\\
&u_i \leq u_2, && i = 8, 9\\
&u_i \leq u_4, && i = 10, 12\\
&u_{11} \leq 21 u_4,\\
&u_{13} \leq u_5,\\
&u_{14} \leq 129 u_5,\\
&u_i \leq u_6, && i = 15, 16\\
&u_i \leq u_7, && i = 17, 18\\
&u_i \leq u_8, && i = 19, \ldots, 22\\
&u_i \leq u_{10}, && i = 23, 24, 25\\
&u_i \leq u_{12}, && i = 25, 27
\end{aligned} \tag{4}$$

The mandatory constraints are the following:

$$\begin{aligned}
&u_1 \leq u_i, && i = 4, 6, 7\\
&u_4 \leq u_i, && i = 10, 11, 12\\
&u_5 \leq u_{14},
\end{aligned} \tag{5}$$

The group constraints are the following:

$$\begin{aligned}
&u_2 \leq u_8 + u_9 \leq 2u_2,\\
&u_6 = u_{15} + u_{16},\\
&u_7 = u_{17} + u_{18},\\
&u_8 = u_{19} + u_{20} + u_{21} + u_{22},\\
&u_{10} = u_{23} + u_{24} + u_{25},\\
&u_{12} = u_{26} + u_{27}.
\end{aligned} \tag{6}$$

Finally, the cross-tree constraints are the following:

$$\begin{aligned}
&u_3 \leq u_8, && (\mathbf{C1})\\
&u_{17} \leq u_9 + u_{22}, && (\mathbf{C2})\\
&u_{18} \leq u_{20} + u_{21} + u_{22}, && (\mathbf{C3})\\
&u_{15} \leq u_{21} + u_{22}, && (\mathbf{C4})\\
&0 \leq u_{11} \leq 21, && (\mathbf{C5})\\
&0 \leq u_{14} \leq 129. && (\mathbf{C6})
\end{aligned} \tag{7}$$

Constraints **C5** and **C6**, which refer to the lower and upper bounds of variable features graphics $frameRate$ and sound *quality*, are defined based on the domain of u_{11} and u_{14} and the transformations of bounds described in Section 3.3. Finally, when the game application is running, then $u_1 = 1$. The set of constraints is therefore $\mathcal{C} = \{(4) \cup (5) \cup (6) \cup (7)\}$, that must hold true for every time instant in the prediction horizon.

4.3. Formulation of the decision-making strategy

The model $\mathcal{C}$ is used to derive the learnt model $\mathcal{M}_t$. At design time, an initial model $\mathcal{M}_t$ is learnt from data logged from the running system, where

the features $\mathbf{u} = \{u_1, \ldots, u_{27}\}$ are the ones described in the previous section, and the performance indicators are $\mathbf{y} = \{y_1, y_2, y_3\}$, where y_1 is the current value of the utility, y_2 is the estimated battery level; and y_3 is the current battery consumption.

The objective function of (3) is the utility $F(\mathbf{u}) = y_1$, and it is computed as:

$$F(\mathbf{u}) := y_1 = f_{u_{11}}(u_{11}) + f_{u_{14}}(u_{14}) + \sum_{i \notin \{11,14\}} \alpha_i u_i(k),$$

with $f_{u_{11}}$ and $f_{u_{14}}$ being functions that increase the utility as the quality of experience increases with the enabled quantitative feature, and α_i are weights on the relevance of having the i-th feature enabled.

Finally, we can introduce additional constraints related to the battery level, that should never become negative, i.e., $y_2(t) \geq 0$ for all the points in time.

The solution needs to be re-assessed periodically, since different factors, e.g., environmental conditions, distance from routers, etc., might affect the performance indicators, and perhaps they are not being accurately captured by the model. In our case, the first reconfiguration action of the plan is the runtime application configuration (including numerical features) that fits not only the current context, but also the expected behaviour of the system. At runtime, a new plan is computed periodically, based on the current measured operating conditions. The selection of the period depends on: (i) the application, (ii) the number of features in the model and (iii) the available computation capacity. Taking into consideration the aforementioned issues for our case study, the selected period is 250 milliseconds.

One of the challenges to apply DSPL approaches in real environments is to obtain a model that captures the relationships between the features and the KPIs, i.e., the configuration model or the $\mathcal{M}_t$ model in ProDSPL. In general, this issue is considered out of the scope of the DSPL approach. Then, the models can be obtained using different techniques like machine learning models, analytical models, simulation-based models [36] or the information provided by an expert. The chosen technique strongly depends on the specific application. In some cases, capturing such relationships are more challenging or even impossible for different reasons. Sometimes, the relationship between the features and the KPIs is not clearly defined; other times, it is not possible to log a trace of the system operation to learn the model; in other cases, the operational environment is very dynamic, and this is just to mention a few. In the case of ProDSPL, it needs traces from previous executions or logs of the behaviour of the system to learn the $\mathcal{M}_t$ model. However, this model does not require to be very accurate as it can be adapted at runtime (see Section 3.2).

For the Strategy Mobile Game, the initial set of traces of the system is generated putting the system in random configurations and measuring the utility and the battery consumption. As we work in a simulated environment, these values are given by the features selected in a simulation step. An initial $\mathcal{M}_t$ model is learnt from this data and can be used at runtime to guide the decision strategy driven by the adapter. The adapter is part of the mobile application simulator–described in Section 5.2. During the simulation runtime, the adaptation is performed taking as input the values of utility and battery consumption generated by the simulator.

Therefore, in order to apply our approach, it is fundamental to have a testing environment to get these traces or experimental data. In the worst scenario, the $\mathcal{M}_t$ model should be adapted at runtime, something also done by other approaches like FUSION [3]. However, this limits the application of our approach in mission critical applications.

5. Experimental results

The aim of experimentation is characterizing the time and estimate the energy required to generate the configurations, the number of reconfigurations needed, and the quality of the proactive ProDSPL solutions in comparison with a reactive approach used in DSPL, DAGAME [5], which uses a genetic algorithm for the generation of feature model configurations at runtime. We have chosen DAGAME because it has been proven to give quasi-optimal solutions in minimum time, it is a DSPL approach like ProDSPL, and we have the original code, so that the comparison is fair enough.

5.1. Objectives and Research Questions

The methodology of this study is defined according to the goal-question-metric approach [37] as follows: "Analyze the feasibility of using proactive

control for DSPL and assess its weaknesses and strengths". To achieve this goal, we set the following research questions (RQ):

RQ1. How is the utility of the configurations generated? This question studies the utility of the configurations generated by the proactive controller, taking into consideration different prediction horizons. We compare our results with static configurations of the system and with the configurations generated by a reactive approach, the genetic algorithm presented in [5].

RQ2. How good are the execution times of the proactive control for DSPL? This question explores the feasibility of applying our approach at runtime, in terms of execution time, for contexts with different response time requirements. Execution times around 1 second could be acceptable for interactive apps (which require of user interaction) [26, 24, 27], but unfeasible for apps and contexts with real-time requirements. Additionally, we explore the influence of the prediction horizon on the execution time.

RQ3. When is a proactive strategy better than a reactive strategy for a given system? In order to answer this question, we analyze the evolution of a given system when it is controlled by our proactive strategy, and when it is controlled by the reactive approach presented in [5], paying special attention to the number of reconfigurations required by both strategies. Also, the size of the prediction horizon will be considered for the proactive strategy. We consider that a strategy is better when it requires fewer reconfigurations than other maintaining a similar quality of the running system. We put the focus on reducing the number of reconfigurations since they involve the consumption of time and resources and can have a negative impact on the system's stability.

5.2. Data collection

To answer the research questions proposed in the previous section, we will study different aspects of ProDSPL related to reconfiguration (i.e., execution time, utility and decision-making strategy) for the presented case study (i.e., the Strategy Mobile Game), and also for DSPLs with variability models randomly generated for reactive and proactive scenarios.

The ProDSPL reconfiguration actions are enabling or disabling a specific functionality and the tuning of parameters. Regarding the strategy game, one possible solution to collect the information to answer the RQs could be to implement it with a specific interface for the ProDSPL to reconfigure the mobile application (i.e., to enable and disable services and tuning sound and graphics parameters) and record the utility and power consumption during the experiment. However, the use of a real execution context hampers the possibility of reproducing our experiments by third parties. In real execution contexts, even using the same mobile phone, how the battery of the device degrades over time is different from one device to another [38]. Additionally, there are silent services that interfere and drain the battery, which are not under control of ProDSPL. This implies that, even if we perform the experiments in the same device, we will have different battery consumption for different executions of the same experiment. As the focus of this contribution is analysing the quality of the solutions provided by the proactive strategy compared to a reactive one, we decided to perform our experiments in a more controlled environment. So, we have developed our own simulator of a mobile application. In any case, moving the experiments to a real mobile device would not be difficult, because our solution is implemented in Java, which can be easily translated to Java for Android, and the management of computational resources in current mobile devices allows the adoption of a self-adaptation mechanism that is able to reconfigure app parameters.

The mobile application simulator consists of a set of services that represent the services of the mobile app that can be reconfigured due to self-adaptation. The adapter, which is also part of the simulator and performs the adaptation, takes as input the values of utility and battery consumption. Utility measures how easy and satisfying to use is the application (i.e., user experience) by taking real values between 0 and 10 (being 0 the worst and 10 the best mark for utility). Battery consumption models the increase in battery consumption (measured in milli-ampere) introduced by the features by taking values between 10 and 20. These values of utility and battery consumption are randomly generated previously to the experiments according to continuous uniform distributions $U(0, 10)$ and $U(10, 20)$ respectively.

The services that can be configured as part of the adaptation can be enabled or disabled at runtime, and in each step of the simulation, the utility and the battery consumption of the simulated mobile application are registered. Our simulator can use

different strategies to maximize the utility of the application or to minimize battery consumption: (i) a static strategy, (ii) a reactive strategy and (iii) a proactive strategy. First, it can adopt a static strategy in which the application is in its optimal configuration for utility (i.e., user experience), regardless of the battery consumption, or it can choose the configuration with the minimum battery consumption, regardless of the utility. Second, it can use a reactive strategy using the DAGAME genetic algorithm. And third, it can use ProDSPL, the proactive strategy presented in this paper. With regard to this last option, it can consider different prediction horizons.

We have performed the same test using the same sets of utility and battery consumption values for the different strategies considered, using the feature model of our case study and also feature models randomly generated using SPLOT[1] that differ in the number of features (between 20 and 100 features). These models have been created with the following proportions: 25% of mandatory features, 25% of optional features, 25% of alternative OR features and 25% of exclusive XOR features. With regard to the branching factor, the minimum is 1 and the maximum is 6. The maximum size of the groups is 6 and all the models are consistent. The cross-tree constraints are 3-CNF formulas that consider the 20% of features with a clause density of 1.

The simulator starts with the maximum battery level, that will be depleted by the services that are enabled in each step of the simulation. When the battery runs out, the simulator stops and registers the history of the execution. After conducting these experiments, we realized that the results were very similar for small models, so in this work we report the results for larger models, of 20 (2268 possible configurations and battery capacity of 1200 mAh), 40 (867,840 possible configurations and battery capacity of 2400 mAh), 50 (85,953,600 possible configurations and battery capacity of 4800 mAh) and 100 (impossible to compute the number of configurations and battery capacity of 12,000 mAh). As we have models of different sizes, they will work with devices with different battery capacity. Note, that we consider the number of possible configurations a system can take during runtime, taking into account only the dynamic variation points. The global variability space modelled at design time of SPLs is usually much greater than in DSPLs that only consider runtime variability, so the variability space analyzed in the experiments can be considered sufficiently large [5].

As a proof of concept, our proactive strategy has been implemented using Matlab and connected to our simulator by means of the Matlab API for Java[2]. The implementation of the proactive strategy is based on the Matlab Optimization toolbox[3], exploiting the `optimproblem` and `solve` functions. The generation of the configurations is implemented as an optimization problem whose constraints are derived from the feature model.

Although reactive strategies can be implemented using exact or genetic algorithms, we have chosen the second option because they are more appropriate for problems in which the solution space is very wide, which is the case of DSPLs of more than 20 features, when it is not feasible to evaluate all of the possible solutions in a short time frame [15]. Genetic algorithms use heuristic search to find solutions for optimization problems. This kind of algorithms are able to provide nearly optimal solutions for optimization problems without having to explore the solution space. The execution of a genetic algorithm distinguishes three phases: (i) generation of a population of random solutions to the problem; (ii) combination and mutation of the population of initial solutions; and (iii) selection of the solution that returns the highest value for the objective function. Normally, these three phases are executed periodically, and if the current solution is worse than the new generated solution, the new configuration is applied to the system.

The reactive strategy implemented in our simulator uses the genetic algorithm DAGAME [5]. The selection of DAGAME facilitates the comparison between the reactive and proactive strategies in similar conditions: like ProDSPL, DAGAME generates at runtime the configurations of a DSPL implemented using a variability model taking into account the utility and the contextual information (i.e., the battery level). DAGAME also works with extended feature models, and makes use of a fit function (i.e., objective function) to drive the configuration process, which is an unusual characteris-

[1] `http://www.splot-research.org/`

[2] `https://www.mathworks.com/help/matlab/matlab-engine-api-for-java.html`

[3] `https://www.mathworks.com/products/optimization.html`

tic. Finally, the solutions generated by DAGAME have an optimality higher than 87.5% compared with the optimal solution. The policy implemented by the reactive strategy considers two situations for reconfiguring the system. If the battery of the mobile phone is between 80% and 100% of its capacity, DAGAME will look for configurations that maximize utility. But, once the battery level is lower than the 20% of the battery capacity, it tries to minimize the battery consumption to extend the lifetime of the system.

Both the proactive and the reactive strategy of our experiments use models of the system to drive their behaviour. In the case of ProDSPL, it uses the model learnt from the system traces at design time (i.e., the aforementioned model $\mathcal{M}_t$, explained in Section 3), whereas DAGAME needs a model of the system to compute its fit function, because it needs to know the relationship between features and utility and battery consumption. For these experiments, it works with the random values generated for the simulations about the system behaviour. This implies that DAGAME uses perfect information to generate the successive dynamic configurations at runtime, while the quality of the configurations generated by ProDSPL depends on the accuracy of the learnt model $\mathcal{M}_t$.

5.3. Answers to research questions

RQ1. How is the utility of the configurations generated? That is, how good is the quality of ProDSPL solutions. In order to answer this question, we compare the accumulated utilities of ProDSPL and DAGAME, a pure reactive approach that gives quasi-optimal solutions (90% of the optimal solution). As we explained in Section 3, the proactive strategy generates a plan over a prediction horizon subject to the constraints derived from the DSPL. Our hypothesis is that *the wider the prediction horizon is, the better the accumulated utility of the plan and the accumulated utility of the overall system.* Therefore, we include the prediction horizon of the proactive strategy in our analysis. The maximum value of prediction horizon considered is the one that allows the strategy to provide a plan in a reasonable time. In the average iteration, this should be below ~ 2 seconds. Hence, this corresponds to a value of 10 for the feature model of our case study and the feature model with 20 features. For the rest of the models used, the maximum value of the prediction horizon is 8. Additionally, as DAGAME is a randomized stochastic approach that gives different results each time it is executed, we consider the average of 20 executions of the experiment, the best case with regard to the accumulated utility and the worst case. Figure 3 and Tables 2, 3 and 4 summarize the results of our experiments.

Figure 3 shows the evolution of the utility and the accumulated utility of the Mobile Game case study for the different adaptation strategies supported by our simulator: static for maximum utility (*max-utility*), static for minimum battery consumption (*min-bc*), the three cases considered for the DAGAME (*genetic-avg*, *genetic-best* and *genetic-worst*) and the proactive strategy for different prediction horizons ($H = i, i = 1..10$). According to these plots, the proactive strategy $H = 10$ reaches the highest accumulated utility after *min-bc*. Indeed, it provides a solution with a utility higher than the *min-bc*, which elongates the lifetime of the system more than the other strategies used. On the other hand, the reactive strategy maximizes the utility of the configuration in the first phase of the simulation reaching a sub-optimal (see *genetic-best* and *genetic-avg*). When the battery level is lower than 20%, the reactive strategy adapts the system to minimize battery consumption. However, this strategy does not reach an accumulated utility higher than the proactive strategy and even *genetic-average* does not improve the accumulated utility of *max-utility*. This behaviour is similar to the proactive strategy with $H = 1$ (nearly reactive), which reaches a utility higher than the *genetic-best* and an accumulated utility higher than the *genetic-avg*.

The results obtained for the random feature models are shown in Tables 2 and 3. For ProDSPL we highlight in bold the highest results of the accumulated utility for each horizon (Table 2). Additionally, Table 4 shows the percentage difference of these results between the ProDSPL experiment with the highest accumulated utility for each feature model (in bold in Table 2) and the average case of DAGAME. Looking at the results shown in Tables 2 and 3, we can see that the *Accumulated Utility* of the system during all its lifetime is remarkably similar for ProDSPL and DAGAME (nearly the same for feature models with 50 and 100 features, with a difference of -0.23% and +0.21% as shown in Table 4). For the smallest model of 20 features the results are better for DAGAME (20% as Table 4 shows), but for the model of 40 features they are slightly better for ProDSPL (almost 6%).

Table 2: Summary of the experiments for ProDSPL

FM[a]	H[b]	Accumulated Utility	Lifetime[c]	Number of Reconfigurations	Min. Utility	Max. Utility	Average Utility	Std. Dev. of Utility
20	**1**	**21,505**	**571**	**4**	**15**	**44**	**37.59**	**11**
	2	20,455	1327	3	15	32	15.41	1.28
	4	20,469	1327	3	15	32	15.41	1.28
	6	20482	1327	3	15	45	15.42	1.45
	8	20,499	1327	3	15	45	15.42	1.45
	10	21,480	571	3	15	44	37.61	11.01
40	1	48,591	469	4	26	121	103.38	26.81
	2	**48,641**	**469**	**4**	**26**	**147**	**103.49**	**26.88**
	4	42171	1391	3	26	114	30.29	13
	6	42,076	1391	3	19	69	30.22	12.81
	8	48,554	469	4	26	121	103.3	26.88
50	1	126,380	763	4	78	188	168.41	35.85
	2	138,660	1700	2	78	108	81.56	9.71
	4	138,376	1700	3	76	108	81.56	9.7
	6	138,743	1700	3	78	108	81.56	9.7
	8	**138,817**	**1700**	**3**	**78**	**157**	**81.6**	**9.87**
100	1	270,424	1400	4	14	233	193.02	57.37
	2	**277,752**	**4106**	**3**	**61**	**385**	**67.41**	**19.85**
	4	274,040	4064	3	61	408	67.41	20.31
	6	269,269	1360	4	61	233	197.84	53.1
	8	273,508	1402	4	61	336	194.7	55.29

[a] Size of the FM in number of features
[b] Prediction horizon in time steps
[c] System lifetime in time steps

Table 3: Summary of the experiments for DAGAME

FM[a]	Experiment	Accumulated Utility	Lifetime[b]	Number of Reconfigurations	Min. Utility	Max. Utility	Average Utility	Std. Dev. of Utility
20	Best	29,004	528	16	26	65	54.87	13.9
	Worst	24,833	483	17	26	65	51.25	13.25
	Average	26,307	535		1.3	65	48.99	18.11
40	Best	46,440	395	37	29	161	117.41	55.43
	Worst	45,207	388	36	16	161	116.36	56.61
	Average	45,919	397		1.45	161	115	57.28
50	Best	144,046	1377	13	93	187	104.53	27.04
	Worst	133,443	1306	11	87	214	102.09	34.63
	Average	139,144	1382		4.75	202.75	100.61	34.5
100	Best	291,737	1359	25	80	360	214.51	56.99
	Worst	264,024	1277	26	99	404	206.59	70
	Average	277,142	1318		0.075	398.7	203.78	79.44

[a] Size of the FM in number of features
[b] System lifetime in time steps

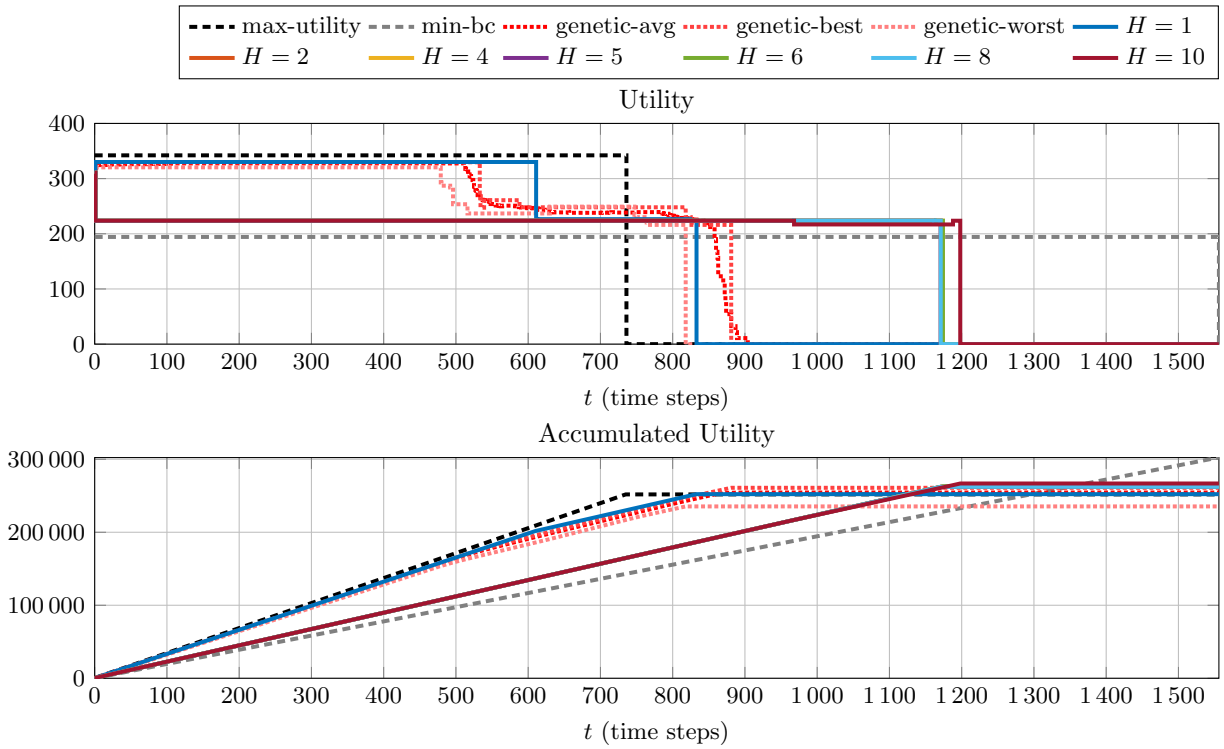

Figure 3: Comparison of utility (top) and accumulated utility (bottom) for ProDSPL with different prediction horizons (H, solid lines), DAGAME (dotted lines) and static strategies (dashed lines)

However, it is worth noting that the proactive strategy makes the system live longer in all the scenarios (see *Steps* column, that refers to number of simulation steps). For the model of 100 features, the system lives with ProDSPL during 4106 steps, and with DAGAME only during 1318 steps (the percentage difference is around 100%). Another very important difference is how stable the system is during its lifetime in each case, i.e., the approach which needs the least number of reconfigurations to achieve a given accumulated utility, is the best one. Taking a look at the results we can see that to obtain similar accumulated utility as commented above, ProDSPL always requires a lower number of reconfigurations (see column *Number of Reconfigurations*). The proactive approach makes 3 or 4 reconfigurations, and the reactive one makes many more, from 11 to 37 reconfigurations, being the percentage difference for this value between ProDSPL and DAGAME higher than 120% for all the experiments.

With regard to the descriptive statistics, we will analyze the utility of each configuration generated by both strategies. The configurations generated by ProDSPL have minimum values of utility higher than the ones of DAGAME (see column *Min. Utility*). However, these configurations do not reach values of utility as high as the configurations generated by DAGAME (see column *Max. Utility*). With regard to the average utility (see column *Average Utility*), DAGAME obtains better results for all the experiments, but the higher number of steps of ProDSPL experiments makes possible for ProDSPL to accumulate similar values of utility. Finally, the standard deviation of the utility (see column *Std. Dev. of Utility*) is lower for ProDSPL. This result illustrates that the proactive strategy is able to keep the system more stable, because the utility offered by the system during its whole lifetime is similar and does not show high variations. This stability has repercussions in the user experience, which will be less stressful. We consider these results are positive because, as DAGAME is able to generate configurations with about 90% optimality, and the utility of the solutions obtained using ProDSPL is very similar, they mean that ProDSPL is able to generate configurations with also approximately 90% of the optimal configura-

tion that would be obtained using an exact algorithm.

Finally, regarding the influence of the prediction horizon, we observe great similarities in the accumulated utility obtained when predictions horizons are 1 (almost reactive) or 8/10. The interpretation we made of this result is that, for this kind of models, the best strategy (even in the long term, with prediction horizon 8/10) is to maximize the utility as much as possible in each step of the simulation. This will also explain the similarities found between ProDSPL and DAGAME for these experiments.

Taking into account the results of our experiments, our answer for RQ1 is that ***the accumulated utility of the proactive strategy is very good, since it is comparable with the accumulated utility of the reactive strategy that tries to maximize utility in every step. Also, the proactive approach makes the system live longer and needs much less reconfigurations to keep a certain quality, so the system execution will be more stable from the point of view of the user utility***. Which strategy scores better depends on the size and attributes of the specific feature model. As future work, it would be interesting to explore which kind of feature models are better for each strategy.

RQ2. How good are the execution times of the proactive control for DSPL? In order to answer this question, we have measured the execution time of ProDSPL for different prediction horizons (see Table 5). We stopped our experiments when the response time of the algorithm becomes unacceptable for the reconfiguration of an application that requires user interaction. We have reached this limit with a prediction horizon of 8 for some experiments and 10 for others. To address the self-adaptation of real-time applications, ProDSPL can be adjusted to reduce the worst execution times. Related to this, the real-time community has explored different options [11] that includes to exploit simple properties of interior point algorithms, to execute the next proposed step by the proactive strategy instead of computing a new plan and to reduce the complexity of the problem. We will explore this issue as future work.

The answer to this question is that the execution times of the proactive strategy are in the order of milliseconds on average. However, in the worst case it could be around some minutes, so we plan to explore some strategies to reduce these times. In any case, these results make our approach suitable for user interaction when the prediction horizon is not so high.

RQ3. When is a proactive strategy better than a reactive strategy for a given system? In accordance with the results of our experiments, the ProDSPL is able to generate configurations with an accumulated utility similar to DAGAME and make the system live for longer. Additionally, ***we have shown with our experimental results that* ProDSPL *is able to provide more stable configurations of the system, requiring less reconfigurations and maintaining a good and regular quality of service***. In these experiments, we do not consider the cost of reconfiguring the internal architecture of the system, although of course, it introduces an important overhead in resource consumption (e.g., energy, CPU time), and it could interrupt user work during reconfiguration [39].

So, we consider that any reconfiguration strategy that minimize the quantity of system adaptations required is very advantageous. As we have already commented above, the number of reconfigurations performed in our experiments (see column *Number of Reconfigurations* in Tables 2 and 3), shows that the system adapted by DAGAME required between 11 and 37 reconfigurations of the system, while ProDSPL required reconfiguring the system only between 3 and 4 times, depending on the prediction horizon considered. The number of reconfigurations performed by the genetic algorithm is the expected because these algorithms always look for a solution that improves the quality of the current configuration and use pseudo-random approaches (see Section 5.2). Therefore, it is very likely that during the execution of the experiment DAGAME reconfigures the system many times (i.e., any time it founds a solution better than the current one). ProDSPL generates its plans with the best possible configuration, because it takes into account the learnt model of the system, which models the probable behaviour of the system considering its future performance. Then, ProDSPL usually maintains the system in the same configuration, unless a sudden change in the context happens or the system reaches some boundary requiring immediate reconfiguration. Notice that the experiments take as input the same set of values for utility and battery consumption as contextual information.

Minimizing the energy consumption of adaptation is specially important in battery powered de-

Table 4: Percentage differences between ProDSPL and DAGAME

FM	Accumulated Utility	Steps	Number of Reconfigurations	Min. Utility	Max. Utility	Average Utility	Std. Dev. Utility
20	−20.087%	7.82%	−121.95%	168.09%	−38.53%	−26.33%	−48.84%
40	5.75%	16.62%	−160.49%	178.87%	−9.09%	−10.53%	−72.24%
50	−0.23%	20.63%	−120%	177.03%	−25.43%	−20.86%	−111.02%
100	0.21%	100.53%	−157.89%	199.5%	−3.49%	−100.57%	−120.03%

Table 5: Times statistics for ProDSPL (in milliseconds)

FM	H	Min	Max	Avg.	Std. Dev.
20	1	115	1987	236	99
	2	142	2600	625	226
	4	138	2824	703	256
	6	153	4225	785	339
	8	175	4041	839	408
	10	363	4357	653	208
40	1	232	3060	469	183
	2	232	3553	784	249
	4	263	4496	882	292
	6	295	4776	1005	335
	8	567	5358	805	257
50	1	303	4789	510	180
	2	325	3645	951	253
	4	355	4580	1103	304
	6	355	5194	1221	365
	8	375	434,148	1578	10,497
100	1	615	3884	1073	314
	2	508	398,719	1732	7863
	4	540	399,373	1949	7915
	6	910	308,411	1675	8327
	8	951	309,379	2087	8234

vices like mobile phones. In [39, 40] it is shown that, although the benefit of applying reconfiguration services is always worthy to maintain the good quality and health of the system, it introduces an energy consumption overhead that should be considered when choosing a self-adaptation strategy. Indeed, the authors of these papers compare three different mechanisms to execute a reconfiguration plan in Android phones, and the energy cost of monitoring and executing the plan goes from 0.68% to 2.30% ([40], Figure 6) of the total cost (the decision-making considered here is the simplest one, rules based on if-clauses). So, this study suggests that executing a reconfiguration plan 37 times (the worst case of the reactive strategy) could introduce a substantial energy consumption overhead, compared with the 3 or 4 times of ProDSPL. On the other hand, the energy consumption in mobile phones is important not only for the battery lifetime, but also for sustainability. Any decrease in energy consumption of an app would significantly mitigate the greenhouse effect of ICT technologies, when it is executed in thousand or millions of mobile phones. However, considering the energy consumption of software in mobile phones is very difficult to measure accurately [39], more experiments should be performed to achieve a forceful conclusion, and this is beyond of the scope of this paper.

The other important issue is the response time. DAGAME is able to provide a solution in milliseconds [5], while the average time of ProDSPL is between 236 milliseconds and 2 seconds, which are acceptable times that do not affect user interaction [26, 27].

An interesting feature of the proactive strategy is that **we can guarantee how the system will behave, due to the stability of the configurations and because the number of solutions provided is deterministic**. This is not the case of the genetic algorithm, where each execution gives different solutions. For instance, for the model with 20 features, the average of the accumulated utility is 26,307, but in the best case it is 29,004 and in the worst case is 24,833. Additionally, for the same feature model, there is a difference of 52 simulation steps, for the same system conditions. This is not the case of the proactive strategy, which has the same behaviour for each execution.

In conclusion, ProDSPL is advantageous for scenarios (i) without hard real-time constraints (our system provides solutions with an accuracy near to 90% of the optimal solution), (ii) in which the reconfiguration of the system has a great cost in terms of computational resources (energy, memory and computation) and (iii) when a system reconfiguration is noticeably by the user, leading to her/his dissatisfaction.

6. Threats to validity

This section discusses briefly the internal validity, construct validity and external validity of our study [41]. The internal validity intends to explore whether the results obtained are influenced or not by other factors. Threats to construct validity concern to the completeness of our study, as well as any potential bias. Finally, external validity analyses whether the results obtained in the experiments can be generalized or not.

One important threat for the internal validity of our study is the accuracy of the results provided. This is specially important in experiments with real devices because measuring tools usually focus on the entire device (not on a specific application) and introduce an additional error. In order to avoid like this, we have focused on simulated scenarios. Additionally, our experimental setting facilitates the reproduction of our results.

The first construct validity identified is related to the algorithms used to compare the work of ProDSPL. Once the use of exact algorithms for DSPL was discarded, since according to the literature [8], they only work with small feature models (i.e., around 20 features), we decided to compare our solution with a stochastic approach. In this area, there are a lot of relevant works [15, 5, 18, 36, 42], but, to avoid the introduction of additional bias, we focused on those algorithms that work with extended feature models and whose source code has been made available by the authors. These conditions are fulfilled by FEMOSAA[4] and DAGAME[5]. Finally, we settled on DAGAME because it shares two main points with our work: it is single-objective and takes into account the battery consumption of features to generate the dynamic configuration of the feature model. This means that it is not possible to generate a configuration with a battery consumption higher than the remaining battery level of the device.

Threats to construct validity may also be related to the stochastic nature of genetic algorithms which can influence the measurements. We have addressed this threat by doing five repetitions of each experiment. Additionally, we have summarized the results for the best case, the worst case and the average case.

[4] `https://github.com/JerryI00/Software-Adaptive-System`

[5] `http://www.lcc.uma.es/~gustavo/mo-dagame.html`

An important threat to the external validity is the selection of the feature models used in the experiments. We have addressed this threat by selecting the same feature model used for the evaluation of DAGAME in [5] and some randomly generated feature models. However, we are aware that there could be issues with some particular combination of the FM size, the variability degree and the number of cross-tree constraints and, therefore, the results obtained in our experiments could be different.

We consider the number of features of feature models used in the experiments as a threat to the external validity. When the number of features increases, the number of possible configurations of the feature model increases too, and it is more difficult for the algorithms to generate a solution. In this regard, we take into consideration scenarios in which the number of features is high enough to pose a challenge for the algorithms. According to the literature, this limit is around 20 features [8]. On the other hand, we have to consider an upper bound for the number of features. In the case of mobile apps, as in our case study, this upper bound is set around 20 features [18], while, for general applications, some works suggest an upper bound of 100 [43], and other works consider 1000 features [36]. However, in our opinion, an application that could change 1000 features at runtime is unrealistic. So, our bound ranges between 20 and 100 features, considering in this range 4 different sizes of feature model to have enough variety.

7. Related work

7.1. Self-adaptation of mobile applications

Self-adaptive systems have been of great interest in the last years, and specially in many areas such as mobile computing, robotics or ubiquitous computing [11, 44, 10]. However, the approaches that explore self-adaptation in mobile applications are still scarce, and works in critical scenarios (gaming, emergency, medical) are just starting to emerge. A recent work [45] provides a systematic literature study that identifies, classifies and evaluates the characteristics of existing approaches for self-adaptation in mobile apps from the researcher's and practitioner's point of view. The results of this study reveal that our approach aligns with the existing approaches in important characteristics. According to [45], the most frequent goals for adaptation in mobile systems are related to technical qual-

ity properties, such as performance and energy consumption. Non-technical goals, such as promoting user behavioural change and lifestyle improvement, are less frequent. Self-adaptation is performed autonomously to optimize the system in terms of a trade-off among different dimensions (e.g., to reduce battery consumption) and resources (such as CPU usage, user experience, etc.). Similarly to most of these approaches considered in [45], ProD-SPL adapts the system autonomously to optimize both user experience and energy consumption.

Two important differences regarding the adaptation can be found between the majority of the approaches surveyed in [45] and ours. The first one is that proactiveness is not considered, as all the approaches are reactive (i.e., adapt when the change occurs). Although most of the self-adaptive systems foresee what the changes are that could occur during the normal operation of the system, adaptation does not anticipate the possible occurrence of these changes as our approach does. The second difference regards the time required for executing the adaptation mechanisms applied to self-adaptation. Most of the approaches considered in the study do not guarantee an upper bound on the execution time of the self-adaptation process. In some application domains, such as gaming apps, to know and respect upper bounds on the execution time of the self-adaptation process is mandatory. As the results of our experiments show, our approach can guarantee an upper limit on the duration of the self-adaptation process (in the order of seconds). This makes it suitable for scenarios with real-time constraints and in which the reconfiguration of the system has a great cost in terms of computational resources (energy, memory and computation).

7.2. Decision-making process in DSPLs

There are several works that rely on randomized stochastic approaches (mainly genetic algorithms, that successively generate a configuration of a feature model [15, 5, 18, 36, 42]). The main advantage of these approaches is that they can provide quasi-optimal solutions that can be generated during the execution of the system in a reasonable time. The quality of the solutions obtained is evaluated using fitness functions. However, the use of genetic algorithms and feature models requires tackling three important challenges: (i) to avoid the exploration of not valid feature model configurations; (ii) to make a homogeneous exploration of the search space; and (iii) to manage numerical features.

One of the first approaches that considers the combination of DSPL and genetic algorithms is GAFES [15], which generates configurations taking into account resource constraints of the system to be configured. This approach considers feature models without numerical features, and the selection of configurations is based on the operator *FesTransform*, which is able to fix an invalid configuration.

DAGAME [5] and MODAGAME [18] are approaches for the self-adaptation of mobile devices applications using genetic algorithms. The decision strategy used is a combination of fitness functions and ECA rules. When the fitness of the system is below a given threshold the DSPL requests a new configuration to the algorithm. These approaches use a *fix operator* to repair non valid configurations generated by the genetic algorithm and make them valid. In [18] authors explore the solution space by applying different Multi-Objective Evolutionary Algorithms (MOEA) using the Hyper Volume metric. The conclusion is that, depending on the size of the model, MOEA algorithms exhibit a significantly different behaviour. The case of the algorithm MCHC, which shows the best results for big feature models (i.e., more than 500 features) and the worst results for small feature models (i.e., less than 25 features) is remarkable. In these works, numerical features are modelled using XOR groups whose members are all the possible values that the numerical feature can take. This solution complicates the definition of cross-tree constraint involving numerical features.

FEMOSAA [36] is a framework that explores the synergy between feature models and a given MOEA to optimize systems at runtime. In order to avoid the exploration of incorrect FM configurations, it uses a special encoding of the feature model into chromosomes. This encoding considers the core features of the feature model, distinct *elitist features* and dependency operators to drive the search space exploration and avoid incorrect FM configurations. Regarding the numerical features, FEMOSAA considers a solution similar to the one used by DAGAME, but defining also numeric dependencies.

The work presented in [42] follows the same ideas of the previous works, but it is based on a dependency operator that drives the generation of chromosomes. This algorithm demonstrates a better response time compared to [18]. However, this proposal does not consider numerical features, and the

completeness of the solution space explored is not evaluated.

The use of exact solutions in DSPL approaches is not very common due to their computational complexity. However, for some specific domains, and depending on the number of FM configurations, this kind of decision strategy can be adopted. The work in [4] presents an adaptation strategy for the self-healing of service compositions. This work models the self-healing as a pseudo-boolean optimization problem that is resolved using a SAT solver. For feature models with less than 2000 possible configurations, the adaptation process requires around 300 milliseconds. A Branch&Bound algorithm is used in the work [6] for the dynamic adaptation of Multi-Cloud Applications. This algorithm is able to generate the optimal configuration of the application in 2 milliseconds for feature models with less than 60 configurations. In the context of mobile applications, [46] proposes an extension of the MUSIC framework to adapt mobile applications taking into account utility and power consumption. The goal of MUSIC [47] is to dynamically reconfigure component-based applications at runtime to react to context changes and to maintain the application utility in a dynamic environment.

Approaches supported by learning, such as [3, 48], use artificial intelligence techniques to learn the influence of a configuration of the feature model on one or more goals that the DSPL should maintain.

FUSION [3] is an approach that uses learning to determine the influence of the activation of features in the non-functional goals of a system. This information is captured in an analytical model of the system. The definition of goals in FUSION comes with a threshold that establishes whether the accomplishment of a goal is acceptable or not for the user. If the accomplishment of a goal is unacceptable, an adaptation process is launched, which only affects the features that influence the goal and the features that are linked with those features. One of the contributions of this proposal is an on-line process which is able to update the analytical model of the system at runtime. If there is a discordance between the expected utility of the system and the current utility, an induction process is launched that updates the analytical model of the system.

Other approach based on learning is the work presented in [48]. This approach is based on context feature models and it is able to adapt the DSPL to pursue different goals depending on the context. In the manner of FUSION and our approach, it is able to learn at design time an analytical model of the influence of the features in goals using system traces (i.e., a performance-influence model). These performance-influence models encode the relationship between context, system features and optimization goals. The DMP strategy of this proposal incorporates consistency and non-functional influences to this model to generate optimal and consistent configurations at runtime using SAT and MILP solvers.

ProDSPL could be classified in the class of approaches supported by learning. The approaches mentioned above share with ProDSPL the formalization of the feature model as a mixed integer linear program. However, they do not support extended feature models as we do, which limits the applicability of these proposals. They do not properly consider groups with variable number of members (i.e., not just OR, AND or XOR), complex cross-tree constraints and numerical features. On top of that, the main difference between ProDSPL and these approaches is that the latter are reactive, and as such they react to specific situations (e.g., the violation of a goal) and try to provide the optimal or nearly optimal configuration in a given context for some specific goals. We do not claim that proactive adaptation is the best option for all scenarios as we have already discussed, but it is worth noting this is the main difference between our approach and these approaches.

7.3. Reconfiguration based on proactive control

All the solutions presented above in this section are characterized for being reactive, i.e., the scenario that triggers the decision-making process is a change in the execution context.

Although the use of proactive control in DSPL as a decision-making strategy is relatively new, recent proactive self-adaptation mechanisms can be found that apply applying ideas from control theory, such as MPC [12]. There are approaches that implement predictive control in different domains, such as cloud computing, to guarantee non-functional properties [49], on a Denial of Service (DoS) attack scenario [50], or Meeting-Scheduling System [11]. The work in [10] compares two approaches, which are inspired by MPC, using the same benchmark system. The main conclusion is that the improvement that can be obtained with each type of proactive approach is scenario-dependent.

8. Conclusions

In this work we have presented ProDSPL, a combination of Proactive control with DSPL for the self-adaptation of software system. Usually, decision-making strategies for DSPL are based on reactive approaches that try to optimize the configuration of the system for the present situation. However, in some scenarios, a reactive strategy could lead to a faster depletion of system resources and even to an unstable behaviour of the system, as each time the configuration is adapted implies a cost in terms of computational resources or time. Our approach supports the adoption of a proactive strategy instead of a reactive one so as to reduce the number of adaptations. This is because the adaptation involves proposing a new configuration considering not only the current context of the system, but also the system's expected evolution over time. This maintains the system in a valid and stable configuration for a longer time.

Experimentation has shown the feasibility of ProDSPL, which has been surveyed by the analysis of the utility of the configurations obtained, the number of reconfigurations needed, the useful life of the system and the response time. Also, ProDSPL has been compared with DAGAME, a genetic algorithm that generates nearly optimal configurations of a DSPL. In the comparison, in terms of response time, our approach shows times that are slightly higher, but still reasonable, as compared with DAGAME. In contrast, ProDSPL shows better results for the accumulated utility of the system, being able to generate system configurations near to the optimal ones. Additionally, ProDSPL generates more stable configurations of the system, and this minimizes the number of reconfigurations required.

In our ongoing work, the goal is to optimize ProDSPL to reduce the response time and the resources used (e.g., CPU, memory), so as to consume less energy, while improving the quality of solutions. Another line of future work is to explore the use of this proactive strategy for multi-objective DSPL.

Acknowledgements

This work is supported by the projects TASOVA MCIU-AEI TIN2017-90644-REDT, by the projects co-financed by FEDER funds LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 and Rhea P18-FR-1081, by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)", with grant number GMT14-0032, by the Swedish Research Council (VR) for the project "PSI: Pervasive Self-Optimizing Computing Infrastructures", by the Knowledge Foundation with the SACSys project, and by the post-doctoral plan of the Universidad de Málaga.

References

[1] J. Tavčar, I. Horváth, A review of the principles of designing smart cyber-physical systems for run-time adaptation: Learned lessons and open issues, IEEE Transactions on Systems, Man, and Cybernetics: Systems 49 (1) (2019) 145–158. doi:10.1109/TSMC.2018.2814539.

[2] M. Maggio, T. Abdelzaher, L. Esterle, H. Giese, J. O. Kephart, O. J. Mengshoel, A. V. Papadopoulos, A. Robertsson, K. Wolter, Self-adaptation for Individual Self-aware Computing Systems, Springer International Publishing, Cham, 2017, pp. 375–399. doi:10.1007/978-3-319-47474-8_12.

[3] A. Elkhodary, N. Esfahani, S. Malek, Fusion: A framework for engineering self-tuning self-adaptive software systems, in: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE '10, ACM, New York, NY, USA, 2010, pp. 7–16. doi:10.1145/1882291.1882296.

[4] M. Bashari, E. Bagheri, W. Du, Self-adaptation of service compositions through product line reconfiguration, Journal of Systems and Software 144 (2018) 84 – 105.

[5] G. G. Pascual, M. Pinto, L. Fuentes, Self-adaptation of mobile systems driven by the common variability language, Future Generation Computer Systems 47 (2015) 127 – 144.

[6] A. Almeida, F. Dantas, E. Cavalcante, T. Batista, A branch-and-bound algorithm for autonomic adaptation of multi-cloud applications, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, Piscataway, USA, 2014, pp. 315–323. doi:10.1109/CCGrid.2014.25.

[7] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid, Dynamic software product lines, Computer 41 (4) (2008) 93–95. doi:10.1109/MC.2008.123.

[8] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, M. Hinchey, An overview of dynamic software product line architectures and techniques: Observations from research and industry, Journal of Systems and Software 91 (2014) 3 – 23.

[9] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, A. Leva, Comparison of decision-making strategies for self-optimization in autonomic computing systems, ACM Trans. Auton. Adapt. Syst. 7 (4). doi:10.1145/2382570.2382572.

[10] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, B. Schmerl, Comparing model-based predictive approaches to self-adaptation: Cobra and pla, in: 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS),

SEAMS '17, IEEE, Piscataway, USA, 2017, pp. 42–53. doi:10.1109/SEAMS.2017.2.

[11] K. Angelopoulos, A. V. Papadopoulos, V. E. S. Souza, J. Mylopoulos, Engineering self-adaptive software systems: From requirements to model predictive control, ACM Transactions on Autonomous and Adaptive Systems 13 (1) (2018) 1:1–1:27. doi:10.1145/3105748.

[12] E. Camacho, C. Bordons, Model Predictive Control, Advanced Textbooks in Control and Signal Processing, Springer London, 2007.

[13] M. Noorian, E. Bagheri, W. Du, Toward automated quality-centric product line configuration using intentional variability, Journal of Software: Evolution and Process 29 (9) (2017) e1870. doi:10.1002/smr.1870.

[14] J. Li, X. Liu, Y. Wang, J. Guo, Formalizing feature selection problem in software product lines using 0-1 programming, in: Y. Wang, T. Li (Eds.), Practical Applications of Intelligent Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 459–465.

[15] J. Guo, J. White, G. Wang, J. Li, Y. Wang, A genetic algorithm for optimized feature selection with resource constraints in software product lines, Journal of Systems and Software 84 (12) (2011) 2208 – 2221.

[16] K. C. Kang, J. Lee, P. Donohoe, Feature-oriented product line engineering, IEEE Software 19 (4) (2002) 58–65. doi:10.1109/MS.2002.1020288.

[17] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (6) (2010) 615 – 636. doi:https://doi.org/10.1016/j.is.2010.01.001.

[18] G. G. Pascual, R. E. Lopez-Herrejon, M. Pinto, L. Fuentes, A. Egyed, Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications, Journal of Systems and Software 103 (2015) 392 – 411.

[19] M. Hinchey, S. Park, K. Schmid, Building dynamic software product lines, Computer 45 (10) (2012) 22–26. doi:10.1109/MC.2012.332.

[20] T. Dinkelaker, R. Mitschke, K. Fetzer, M. Mezini, A dynamic software product line approach using aspect models at runtime, in: First Workshop on Composition and Variability, Technische Universität Darmstadt, Darmstadt, Germany, 2010, pp. 1–8.

[21] M. Rosenmüller, N. Siegmund, M. Pukall, S. Apel, Tailoring dynamic software product lines, in: Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering, GPCE '11, ACM, New York, NY, USA, 2011, pp. 3–12. doi:10.1145/2047862.2047866.

[22] G. A. Moreno, J. Cámara, D. Garlan, B. Schmerl, Proactive self-adaptation under uncertainty: A probabilistic model checking approach, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, ACM, New York, NY, USA, 2015, pp. 1–12. doi:10.1145/2786805.2786853.

[23] D. Seborg, T. Edgar, D. Mellichamp, F. Doyle, Process Dynamics and Control, 4th Edition, Wiley, 2016.

[24] M. Claypool, K. Claypool, Latency and player actions in online games, Commun. ACM 49 (11) (2006) 40–45.

[25] P. Quax, A. Beznosyk, W. Vanmontfort, R. Marx, W. Lamotte, An evaluation of the impact of game genre on user experience in cloud gaming, in: 2013 IEEE International Games Innovation Conference (IGIC), 2013, pp. 216–221.

[26] B. S. Gardner, Responsive web design: Enriching the user experience, Sigma Journal: Inside the Digital Ecosystem 11 (1) (2011) 13–19.

[27] J. Nielsen, Usability engineering, Interactive Technologies, AP Professional, Cambridge, Mass.

[28] L. Ljung, System Identification: Theory for the User, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[29] M. Maggio, A. V. Papadopoulos, A. Filieri, H. Hoffmann, Automated control of multiple software goals using multiple actuators, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, ACM, New York, NY, USA, 2017, pp. 373–384. doi:10.1145/3106237.3106247.

[30] E. Hadjiconstantinou, G. Mitra, Transformation of propositional calculus statements into integer and mixed integer programs: An approach toward automatic reformulation, Tech. rep., U.S. Army's European Research Office (1991).

[31] R. G. Jeroslow, Lecture 5: Propositional logic and mixed integer programming, in: Logic-Based Decision Support, Vol. 40 of Annals of Discrete Mathematics, Elsevier, Amsterdam, Netherlands, 1989, pp. 79 – 102. doi:https://doi.org/10.1016/S0167-5060(08)70527-2.

[32] G. G. Brown, R. F. Dell, Formulating integer linear programs: A rogues' gallery, INFORMS Transactions on Education 7 (2) (2007) 153–159. doi:10.1287/ited.7.2.153.

[33] H. Williams, Logic applied to integer programming and integer programming applied to logic, European Journal of Operational Research 81 (3) (1995) 605 – 616.

[34] P. Hansen, Methods of nonlinear 0-1 programming, in: P. Hammer, E. Johnson, B. Korte (Eds.), Discrete Optimization II, Vol. 5 of Annals of Discrete Mathematics, Elsevier, 1979, pp. 53 – 70. doi:https://doi.org/10.1016/S0167-5060(08)70343-1.

[35] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, Cambridge, United Kingdom, 2004.

[36] T. Chen, K. Li, R. Bahsoon, X. Yao, Femosaa: Feature-guided and knee-driven multi-objective optimization for self-adaptive software, ACM Trans. Softw. Eng. Methodol. 27 (2) (2018) 5:1–5:50. doi:10.1145/3204459.

[37] V. R. Basili, Software modeling and measurement: The goal/question/metric paradigm, Tech. rep., University of Maryland at College Park, College Park, MD, USA (1992).

[38] L. He, K. G. Shin, How long will my phone battery last?, arXiv preprint arXiv:1711.03651.

[39] A. Cañete, J. Horcas, I. Ayala, L. Fuentes, Energy efficient adaptation engines for android applications, Inf. Softw. Technol. 118.

[40] A. Cañete, J. Horcas, L. Fuentes, Mecanismos de reconfiguración eco-eficiente de código en aplicaciones móviles android, SISTEDES, JISBD.

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[42] A. Alidra, M. T. Kimour, Adapting large pervasive and context-aware systems. a new evolutionary-based approach, International Journal of Knowledge-based and Intelligent Engineering Systems 21 (2) (2017) 103–121.

[43] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, G. Saake, Spl conqueror: Toward

optimization of non-functional properties in software product lines, Software Quality Journal 20 (3) (2012) 487–517. doi:10.1007/s11219-011-9152-9.

[44] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, J. Mylopoulos, Model predictive control for software systems with cobra, in: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '16, ACM, New York, NY, USA, 2016, pp. 35–46. doi:10.1145/2897053.2897054.

[45] E. M. Grua, I. Malavolta, P. Lago, Self-adaptation in mobile apps: a systematic literature study, in: Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019, ACM, 2019, pp. 51–62. doi:10.1109/SEAMS.2019.00016.

[46] G. Brataas, S. Jiang, R. Reichle, K. Geihs, Performance property prediction supporting variability for adaptive mobile systems, in: Proceedings of the 15th International Software Product Line Conference, Volume 2, SPLC '11, ACM, New York, NY, USA, 2011, pp. 37:1–37:8.

[47] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz, Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments, in: B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 164–182. doi:10.1007/978-3-642-02161-9_9.

[48] M. Weckesser, R. Kluge, M. Pfannemüller, M. Matthé, A. Schürr, C. Becker, Optimal reconfiguration of dynamic software product lines based on performance-influence models, in: Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 98–109. doi:10.1145/3233027.3233030.

[49] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Computing 12 (1) (2009) 1–15. doi:10.1007/s10586-008-0070-y.

[50] J. Cámara, W. Peng, D. Garlan, B. R. Schmerl, Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation, Sci. Comput. Program. 167 (2018) 51–69. doi:10.1016/j.scico.2018.07.002.