Self-adapting Industrial Augmented Reality applications with proactive Dynamic Software Product Lines

Inmaculada Ayala, Mercedes Amor and Lidia Fuentes
Departamento de Lenguajes y Ciencias de la Computación
ITIS Software
Universidad de Málaga
Málaga, Spain
{ayala,pinilla,lff}@lcc.uma.es

Alessandro V. Papadopoulos
Mälardalen University
Västerås, Sweden
alessandro.papadopoulos@mdh.se

***Abstract*—Industrial Augmented Reality (IAR) is a key enabling technology for Industry 4.0. However, its adoption poses several challenges because it requires the execution of computing-intensive tasks in devices with poor computational resources, which contributes to a faster draining of the device batteries. Proactive self-adaptation techniques could overcome these problems that affect the quality of experience by optimizing computational resources and minimizing user disturbance. In this work, we propose to apply PRODSPL, a proactive Dynamic Software Product Line, for the self-adaptation of IAR applications to satisfy the quality requirements. PRODSPL is compared against MODAGAME, a multi-objective DSPL approach that uses a genetic algorithm to generate quasi-optimal feature model configurations at runtime. The evaluation with randomly generated feature models running on mobile devices shows that PRODSPL gives results closer to the Pareto optimal than MODAGAME.**

***Index Terms*—Industrial Augmented Reality, Dynamic Software Product Lines, Proactive Control, Self-Adaptation, Optimization**

I. INTRODUCTION

Industry 4.0 is pushing the innovation in factories to optimize industrial processes using Cyber-Physical Systems (CPSs) and Internet of Things (IoT) technologies [1]. One of the key enabling technologies is the Mobile Augmented Reality, also known in the industrial context as the Industrial Augmented Reality (IAR) [2]. IAR encompasses interactive real-time applications that combine real and virtual objects and run and/or display contents on mobile or wearable devices [3], e.g., a smartphone or smart glasses. The goal of this technology is to provide useful and attractive interfaces to operators to obtain information on their tasks and to interact with certain elements that surround them [4]. Typical IAR applications are devoted to supporting assembling tasks, training for new tasks, or providing remote assistance [5]. As IAR applications are meant to be used for extended time periods, it is critical to consider how the IAR software impacts the battery life of the mobile device while providing a smooth experience to the operator.

Most IAR applications use location, network connectivity, and the detection of physical objects intensively, which require intensive computation tasks [6]. As most IAR applications provide visual contextualised data in the users' sight-line via devices such as handheld displays of mobile devices or wearables such as "in-sight" smart glasses, such tasks have to be developed considering their runtime performance and energy efficiency. Usually, to reduce energy consumption, such intensive tasks are offloaded in the cloud, or the edge [7] or use approximate computing [8]–[10]. However, both techniques have their own challenges. The offloading of tasks in mobile environments should consider the location and resources available of edge devices, the type of connectivity, and the coverage in a given moment. On the other hand, approximate computing only can be introduced when it is possible to find a trade-off between accuracy and efficiency. More intelligent adaptation technologies are required in this context, which takes the appropriate action while it considers the context of the mobile device (battery level, memory occupation, network connectivity, etc.) and the work of the application [7]. Self-adaptation is a key technology to overcome these problems [11]. Self-adaptive systems can simultaneously optimize application quality attributes, e.g., user experience, battery consumption, or memory occupation, autonomously.

Self-adaptation is a quality of software systems that can evaluate their functioning and adjust their behaviour accordingly to optimize its performance, recover from failures, configure its functionality, or secure some part of an entire system [12]. The development of self-adaptive systems is challenging for several reasons. Runtime adaptation mechanisms can be unstable, inefficient, and unreliable; self-adaptation systems base their adaptation decisions on models that can be inaccurate, and sometimes, they have to operate in environments with considerable uncertainty. Several approaches have been defined for self-adaptation [11], [13], being Dynamic Software Product Lines (DSPL) one of the most widely used. DSPLs are based on the Software Product Line (SPL) paradigm, but they focus on modelling those features that can change during system execution. At runtime, the context is monitored, and when a change occurs, a reconfiguration service analyses whether this change requires or not to replace the current configuration with a new one.

In a previous contribution, we have proposed ProDSPL [14], a self-optimizing approach that combines DSPL with a control-based proactive decision strategy for dynamically proactively generating optimal configurations. Self-adaptive proactive strategies, like the one used by ProDSPL, are especially well suited for applications that interact with the user, such as IAR applications, because they can adapt the system before a malfunction happens and, in this way, it minimizes user disturbance. However, IAR applications require a multi-objective perspective because it is required to consider at the same time several non-functional concerns of the system like performance, memory occupation, or battery consumption. In this contribution, we analyse the feasibility of using ProDSPL, a proactive self-adaptive approach, to adapt IAR applications. Specifically, this work focuses on three different aspects of ProDSPL: (i) how close are the solutions generated by ProDSPL to the Pareto frontier, or in other words, how optimal are the generated configurations; (ii) are the execution times of ProDSPL adequate for an IAR application; and (iii) when is it more advantageous to use a proactive strategy rather than a reactive strategy for interactive applications. We tested our approach in mobile devices and compared its results with MODAGAME [15], a multi-objective evolutionary algorithm (MOEA) suitable for mobile phones. The evaluation with randomly generated feature models running on mobile devices shows that ProDSPL gives results closer to the Pareto optimal than MODAGAME.

This paper is structured as follows: Section II presents the necessary background to understand our proposal; Section III shows how ProDSPL is adapted to be apply in the context of IAR applications; Section IV presents the results of our experiments and answers to research questions; and Section V provides some conclusions and future work.

II. Background

A. DSPL in Industrial IoT

As with many software systems today, industrial IoT systems and CPSs have to respond to external and internal changes (i.e. variations) that might affect their QoS by providing support for adaptation and reconfiguration at runtime. Such adaptive systems are designed to run continuously and may not be shut down for reconfiguration or maintenance tasks. The variability of such systems has to be explicitly managed, together with mechanisms that control their runtime adaptation and reconfiguration [16]. Dynamic software product lines (DSPLs) are a very valuable approach to self-adaptation based on the software product line (SPL) paradigm, focusing on modelling only those features that can change during system execution and additionally supporting system reconfiguration at runtime. The modelling of those variable features can be specified using different modelling languages, being feature models [17] the most popular [18].

A feature model organizes features into a tree and includes the corresponding tree and cross-tree constraints representing dependencies among features. In DSPL, the features 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. Monitoring the current situation and controlling the adaptation are the central tasks of DSPLs. At runtime, the context is monitored, and when a change occurs a reconfiguration service analyses whether this change requires or not to replace the current configuration with a new one.

In the IoT context, the self-adaptation of the system is a relevant issue and different works have applied DSPL to approach reconfiguration at runtime. DSPL has being applied specifically for the optimization of energy consumption in the context of the IoT devices such as eHealth systems [19], and actually, it has become a good choice to enable the selection of different software options in mobile devices, or autonomous and adaptive systems such as healthcare robots. A DSPL approach reasons about different application variants that may be reconfigured dynamically at runtime driven by a QoS model [20]. Also, as mobile and other pervasive devices pose limited hardware capabilities, they can benefit from an automatic variant selection, which is also limited by non-functional properties of the device like memory and energy consumption [15].

B. Proactive approaches

Proactive adaptation considers both the current and the anticipated adaptation needs of the system. Usually, such approaches make use either of a prediction model of the incoming workload [21] or of a model of how the system to be controlled [22]. In both cases, the models are used to proactively adjust the behavior of the system, based on the ability to learn, predict, and act ahead of time.

Although the use of proactive control in DSPL as a decision-making strategy is new, recent proactive self-adaptation mechanisms apply ideas from control theory, such as Model Predictive Control (MPC) [23]. Some approaches apply predictive control in different domains, such as cloud computing to guarantee nonfunctional properties [24], on a Denial of Service (DoS) attack scenario [25], or Meeting-Scheduling System [22]. The work in [26] compares two approaches that are inspired by MPC, applied to the same benchmark system. The main conclusion is that the improvement that can be obtained with each type of proactive approach depends on the application and the required type of adaptation.

III. Multi-Objective Adaptation of Augmented Reality Applications

In this work, we propose a self-adaptation method that is suitable for IAR applications running in mobile devices. ProDSPL is an application model-based proactive approach; it formulates the adaptation problem as an optimization prob-

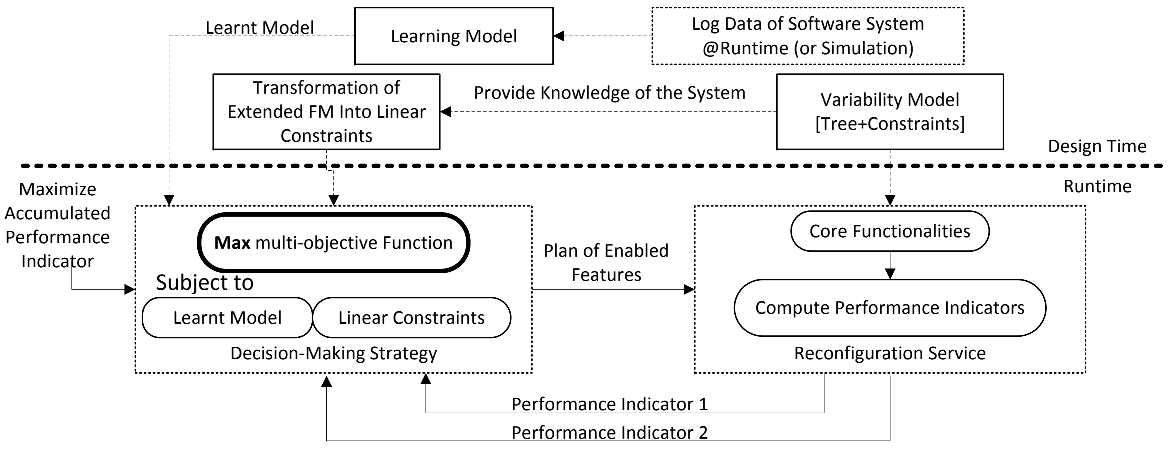
Learnt Model
Learning Model
Log Data of Software System @Runtime (or Simulation)
Transformation of Extended FM Into Linear Constraints
Provide Knowledge of the System
Variability Model [Tree+Constraints]
Design Time
Runtime
Maximize Accumulated Performance Indicator
Max multi-objective Function
Subject to
Learnt Model
Linear Constraints
Decision-Making Strategy
Plan of Enabled Features
Core Functionalities
Compute Performance Indicators
Reconfiguration Service
Performance Indicator 1
Performance Indicator 2

Fig. 1: The PRODSPL approach.

lem over a prediction horizon subject to the linear constraints of the DSPL. PRODSPL uses a dynamic model of the system:

$$\mathcal{M}_t : \begin{cases} \mathrm{x}(t+1) = \mathrm{A}\mathrm{x}(t) + \mathrm{B}\mathrm{u}(t) \\ \mathrm{y}(t) = \mathrm{C}\mathrm{x}(t) \end{cases} \tag{1}$$

that is used to capture how the features $\mathrm{u} = [u_1, u_2, \ldots, u_m]^\top$ selected at time t affect the performance metrics $\mathrm{y} = [y_1, y_2, \ldots, y_m]^\top$; The variable x is called the *state* of the dynamic system [27]. Such a model can be learned offline, but it can also be adapted at runtime, in case a large variability of the model is expected [27]. Such a model can be used to understand how the system will behave in the future based on the current state (represented by the initial condition $\mathrm{x}(0)$), and based on the decisions that are taken over a prediction horizon H (creating a plan of actions $\mathrm{u}(0)$, $\mathrm{u}(1)$, …, $\mathrm{u}(H)$). In fact, Eq. (1) can be "unrolled" for H steps ahead, obtaining:

$$\mathrm{y}(H) = \mathrm{C}\left(\mathrm{A}^H \mathrm{x}(0) + \sum_{i=0}^{H-1} \mathrm{A}^{H-i-1}\mathrm{B}\mathrm{u}(i)\right). \tag{2}$$

The model can be used to optimize the selection of the features u over the prediction horizon, based on the current state $\mathrm{x}(0)$ of the application.

In this work, we consider different optimization criteria $f_1(\mathrm{y}), f_2(\mathrm{y}), \ldots, f_q(\mathrm{y})$ that depend on the measured performance indexes y, and that must be minimized concurrently. To be able to compute a unique solution, we *scalarize* the multi-objective optimization problem through a function $F(f_1(\mathrm{y}), f_2(\mathrm{y}), \ldots, f_q(\mathrm{y})) = \sum_{i=1}^{q} w_i f_i(\mathrm{y})$, where the weights $w_i > 0$ are chosen to prioritize the criterion with highest importance. The resulting optimization problem is:

$$\begin{aligned} \underset{\mathrm{u}(1),\ldots,\mathrm{u}(H)}{\text{minimize}} \quad & \sum_{i=1}^{H} F(\mathrm{y}_i) \\ \text{subject to} \quad & \mathrm{y}_t = \mathrm{C}\left(\mathrm{A}^t \mathrm{x}_0 + \sum_{i=0}^{t-1} \mathrm{A}^{t-i-1}\mathrm{B}\mathrm{u}_i\right), \\ & \mathrm{u}_t \in \mathcal{C}_t, \qquad \forall t = 1, \ldots, H, \\ & \mathrm{C}\mathrm{x}_0 = \mathrm{y}_{\text{measured}}, \end{aligned} \tag{3}$$

where $\mathcal{C}_t$ is a set of constraints that define a feasible set of features u based on the feature model, and the last constraint initializes the optimization problem based on the last measured value of the performance $\mathrm{y}_{\text{measured}}$. If no numerical features are present, classical 0-1 programming solvers can be used for solving (3) [28].

PRODSPL is based on the solution of the optimization problem (3), computed whenever a new measurement $\mathrm{y}_{\text{measured}}$ is available. Therefore, the optimization problem is solved periodically at runtime and generates a plan with the future actions to take within the prediction horizon.

Similar to the approach described in [14], our proposal has activities that take place both at design time and runtime (see Fig. 1). At design time, the variability of the problem is modelled in a variability model and transformed in a set of linear constraints. In addition, we use log data of the system's behaviour to learn the dynamic model used by the proactive control to take its decisions. At runtime, the configuration service runs in a mobile device (or in the device self-adaptive device) and receives periodically from the decision-strategy the plan of enabled features. The reconfiguration service provides the performance indicators. The decision-making strategy uses the dynamic model and the linear constraints from the variability model to provide the plan of enabled features.

A. Use case application

AR has been used in industrial situations to provide useful and attractive interfaces to operators to obtain information on their tasks and to interact with certain elements that surround them [4]. In the specific case of Industry 4.0, the concept of IAR involves the AR hardware, typically a mobile device, e.g., a smartphone or smart glasses, and AR software. As IAR applications are meant to be used for extended periods, e.g., while performing on-site operations, it is critical to take into account how the IAR software impacts the battery life of the mobile device, while providing a smooth experience to the operator.

Several works have focused their attention on how to offload the computation of the IAR application towards the edge of the

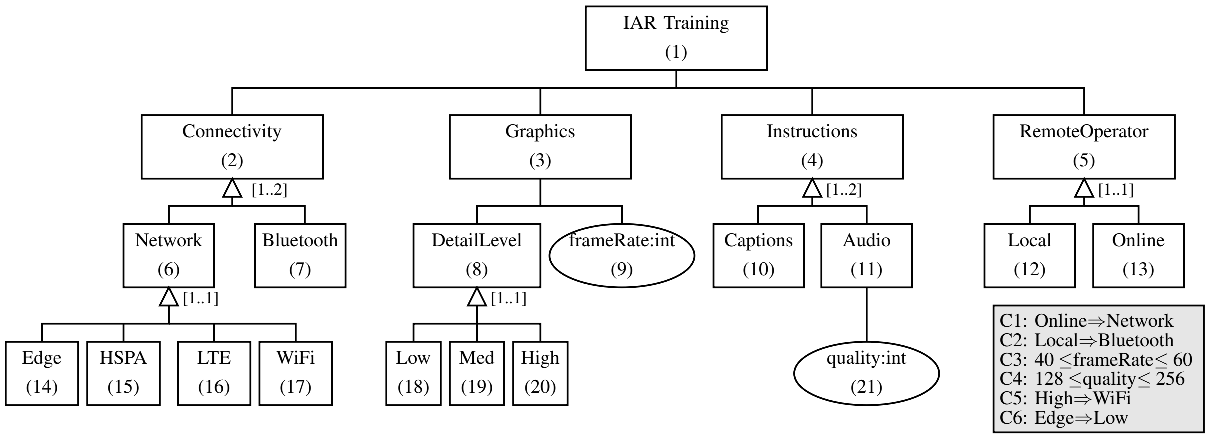

Fig. 2: Feature model of the IAR training application.

network, to reduce the impact on the battery consumption [9]. On the other hand, there are other features onboard the mobile devices that can be tuned to extend the battery life, such as the type of connectivity, the type of visualization, the refresh rate, and so on. Such characteristics can be modeled according to the feature model of SPLs. The feature model can then be used to adjust at run-time what are the enabled features to minimizing the memory and battery usage, which increase the user experience.

Our use case considers an IAR application for carrying out the training of an operator through a mobile device. The use case is inspired by the one presented in [29], where the IAR application is meant to train operators for assembly operations on the real machines using real instruments for the interaction, guided by the mobile device. The IAR application provides instructions and feedback to the trainee, like audio cues or captions. In this use case, we consider also the possibility of connecting to a remote operator for asking for clarifications, in case the training instructions are not clear.

From a platform perspective, the IAR application requires the usage of some connectivity, which can be either through a Local Area Network connection or via a Bluetooth connection. The graphic interface can be tuned to different levels of quality, depending on the available resources.

Fig. 2 shows the extended feature model for the use case. We use two different kinds of features: *choices* and *variables*. *Choices*, which are shown in the figure as rectangles (e.g., Connectivity), 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 shown as ovals, can be evaluated as values of different types, for our case study they are positive integer numbers, i.e., $u_i \in \mathbb{N}$.

The extended feature models allow the specification of cross-tree constraints to delimit the degree of variability. These constraints are defined as relationships between different features of the feature model. The grey box of Fig. 2 shows the constraints of our case study. For instance, C1 states that having a remote operator that is available online for the training requires the network connectivity to be enabled (as it provides a higher bandwidth). It is possible to specify constraints involving the values of a variable feature. For instance, constraint $C3$ states that the value of $frameRate$ is between 40 and 60 frames per second.

B. Formulation of the use case

Once the extended feature model of the case is derived (see Fig. 2), the feature constraints $\mathcal{C}_t$ can be formulated as a set of linear constraints needed for the optimization problem (3). In the following, we show on the given use case how to formulate such constraints, based on the feature model.

We introduce the constraints on the variables u_i where i is the number of the feature in Fig. 2. Additionally, we have to consider the special mappings for variable features (i.e., the case of u_9 and u_{21}). Therefore, the resulting paternity constraints are:

$$\begin{aligned} &u_i \leq u_1, \qquad i = 2,\ldots,5 \\ &u_8 \leq u_3, \\ &u_9 \leq 21u_3, \\ &u_{21} \leq 129u_{11}. \end{aligned} \tag{4}$$

The mandatory constraints are the following:

$$\begin{aligned} &u_1 \leq u_i, \qquad i = 2,\ldots,5 \\ &u_3 \leq u_i, \qquad i = 8,9 \\ &u_11 \leq u_{21}. \end{aligned} \tag{5}$$

The group constraints are the following:

$$\begin{aligned} &u_2 \leq u_6 + u_7 \leq 2u_2, \\ &u_4 \leq u_{10} + u_{11} \leq u_4, \\ &u_5 = u_{12} + u_{13}, \\ &u_6 = u_{14} + u_{15} + u_{16} + u_{17}, \\ &u_8 = u_{18} + u_{19} + u_{20}. \end{aligned} \tag{6}$$

Finally, the cross-tree constraints are the following:

$$\begin{aligned}
&u_{13} \leq u_6, && \textbf{(C1)}\\
&u_{12} \leq u_7, && \textbf{(C2)}\\
&0 \leq u_9 \leq 21, && \textbf{(C3)}\\
&0 \leq u_{21} \leq 129, && \textbf{(C4)}\\
&u_{20} \leq u_{17}, && \textbf{(C5)}\\
&u_{14} \leq u_{18}. && \textbf{(C6)}
\end{aligned} \tag{7}$$

When the IAR application is running, then $u_1 = 1$. The set of constraints is therefore $\mathcal{C}_t = \{(4) \cup (5) \cup (6) \cup (7)\}$, that must hold true for every time instant in the prediction horizon.

The model (2) can be learned at design time from data logged from the running system, where the features $\mathrm{u} = \{u_1, \ldots, u_{21}\}$ are varied one by one to better capture their effect on the performance metrics y.

As these values can be obtained at runtime monitoring the application, we can test our proposal with real devices.

Finally, one can select the objective function to account for different costs of running the application to be simultaneously minimized. For IAR applications running in a smartphone, we have chosen as performance indicators $\mathrm{y} = \{y_1, y_2\}$, where y_1 is the current battery consumption; and y_2 is the memory occupation. The objective function can be therefore selected as $F(\mathrm{y}) = w_1 y_1 + (1 - w_1) y_2$, with $w1 \in [0,1]$; $w_1 \to 0$ will result in an adaptation that tries to minimize the memory occupation, while $w_1 \to 1$ will result in an adaptation that minimizes the battery consumption.

IV. Experimental results

A. Objectives and Research Questions

The methodology used in this study is the goal-question-metric approach as follows: "Analyze the feasibility of using ProDSPL for the adaptation of IAR applications". To achieve this goal we set the following research questions (RQ):

RQ1. How close are the solutions generated by ProDSPL to the Pareto frontier compared with a reactive strategy? This question studies the values of battery consumption and available memory of the configurations generated by the proactive controller. We compare our results with the configurations generated by a reactive approach, the IBEA algorithm presented in [15].

RQ2. Are the execution times of ProDSPL adequate for an interactive application? This question explores the feasibility of using our approach in contexts with different response time requirements. When it comes to user experience, and in the context of real-time interactive applications, this delay determines how users experience an AR-based training session. Normally, response times should be as fast as possible and this question studies if the time required to perform adaptation affects user experience. In interactive and real-time applications like IAR, the response time, a measurement of the total amount of time an application takes to respond to a user request for service or action, has a tremendous impact on user satisfaction. Specifically, in AR, the display of the personal device (e.g., an AR headset or a mobile phone) should show the changes in the environment immediately at the same time the user moves. Any significant delay causes a conflict in the brain which can result in nausea or motion sickness. High latency can cause misalignment of virtual elements in relation to the real world. According to [30], which defines three response-time limits, 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 seconds, but the user does lose the feeling of operating directly on the real-time data.

RQ3. When it is more advantageous to use a proactive strategy rather than a reactive strategy for IAR applications? To answer this question, we analyze the answers to the previous questions paying special attention to the number of reconfiguration executions required by each strategy.

B. Data collection

To answer the research questions, we study different aspects of our proposal in the context of mobile devices. With this goal, we have developed a framework that works with a set of benchmarks generated using random feature models from SPLOT[1].

Self-adaptations are made taking into account two important non-functional properties of IAR apps, to minimize battery consumption and to maximize available memory. The mobile phone starts the experiments fully charged and the benchmark requests a new configuration to an Adaptation Server (AS) providing battery consumption and the available memory every 30 seconds. The AS supports four self-adaptation strategies: random (i.e., random valid configuration of the benchmark FM), empty (i.e., all the adaptive components of the benchmark disabled), proactive and reactive. Each experiment runs until the battery is depleted (which is around 8 hours) and it is repeated three times. When the experiment finishes, the AS generates a report that includes battery consumption and available memory of each configuration of the benchmark. Benchmarks run in a Nexus 5X device and the AS is deployed in an Intel Core i7 with 3.40 GHz of CPU and 16 GB of RAM.

All the benchmarks have the same structure. They have an adaptor, which is in charge of interacting with the AS and adapting to the new configuration, two monitoring services for the battery consumption and the available memory, and a variable number of adaptive components that can be enabled or disabled at runtime. Monitoring is implemented using Android system libraries. However, they only provide battery consumption and available memory for the device (not per application), and it is very inaccurate. Therefore, the adaptation strategy has only an overall notion of the current behaviour of the app. A side effect is that it is very difficult to have a precise learned model for the adaptation of the system. The adaptive components of the benchmark are generated using the FMs. For each feature of the FM, the benchmark has an adaptive component that performs one of these tasks: to calculate the

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

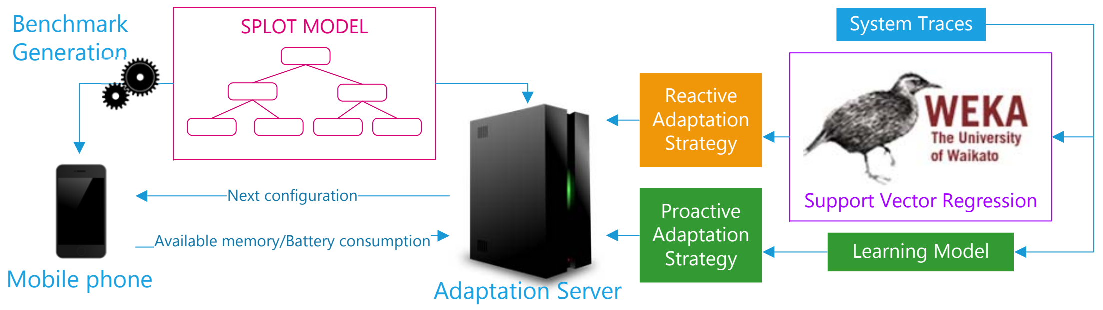
Benchmark Generation
SPLOT MODEL
System Traces
Reactive Adaptation Strategy
WEKA The University of Waikato
Support Vector Regression
Next configuration
Available memory/Battery consumption
Proactive Adaptation Strategy
Learning Model
Mobile phone
Adaptation Server

Fig. 3: Framework for data collection.

tangent, to determine if a number is prime or not, to compute the factorial of a random number, or to sum two numbers. The type of task is randomly determined in the generation of the benchmark using the Java random function.

The proactive strategy presented in Section III has been implemented using the Matlab API for Java and the Optimization toolbox. On the other hand, the reactive strategy is implemented using the MOEA for DSPLS presented in [15]. In this work, some of the MOEAs of the JMetal framework is adapted and evaluated for DSPLs. Specifically, we have used the IBEA algorithm, which generates better solutions for small and medium-sized FMs (20-100 features). MOEAs use a fitness function to determine the quality of generated configurations that needs to know the relationship between features and the available memory and battery consumption. To determine this, we have used the random models and the implementation of the Support Vector Regression of Weka[2] with a lineal kernel.

In the experiments presented in this section, we have used 3 FMs with a different number of features and number of possible configurations (20 and 2.268; 50 and 85.952.600; 100 and impossible to compute a number of possible configurations). 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. Regarding the branching factor, the minimum is 1 and the maximum is 6. The maximum size of the groups is 6. The cross-tree constraints are 3-CNF formulas that consider the 20% of features with a clause density of 1.

C. Answers to research questions

RQ1. How close are the solutions generated by ProDSPL to the Pareto frontier compared with the reactive strategy? To answer this question, we compare ProDSPL with MODAGAME. As we explained in Section III, ProDSPL generates a plan over a prediction horizon subject to the constraints derived from the DSPL. Our hypothesis is *the wider the horizon is, the better will be the battery consumption and available memory of the plan*. However, the duration of our experiments (around 8 hours) makes it impossible to test the behaviour of ProDSPL for all the prediction horizons and FMs. So, we made previous experimentation to determine a prediction horizon that works well with all the FMs, which resulted in being 6.

To compare the solutions of both strategies, we have to consider that they use multi-objective optimization, and the Pareto frontier is unknown. This is due to several factors: (i) the inaccuracy of measuring methods; (ii) the maximum and minimum values of available memory and battery consumption that the device can support are unknown, and (iii) the real optimal configurations of our FMs cannot be determined. However, we know that the application results when any feature is selected, i.e., the empty configuration, should be very similar to the Pareto optimal. Fewer features selected imply lower battery consumption and more available memory. Therefore, the strategy that generates solutions more similar to the empty strategy should be the best one.

We calculate the similarity of the solutions generated by each strategy using the distance between centroids of the solutions generated by each adaptation strategy and the empty strategy. To compute these centroids, the first 30 and the last 30 experiment steps are eliminated because when a lithium polymer battery (as the one of the Nexus 5x) is fully charged or nearly empty provides voltage values that introduce a lot of distortion in the results. The scale of the objective affects this distance, so data has been normalized. We can see the solutions generated by the three strategies for the three FM in the scatter plots of Figures 4-6. Centroids are highlighted using circles, and frontiers are marked using lines. The violet centroid corresponds to the Proactive Strategy, while light blue is for the reactive strategy, and the black one is for the empty configuration.

These scatter plots illustrate the inaccuracy of the information provided by Android about the state of the device. Although the configurations generated by the proactive and empty strategies are always the same, we can see a great dispersion in the results. Additionally, it is interesting to note that the behaviour of the strategies differs a lot depending on the size of the FM. For 20 and 100 features, the proactive strategy can maintain the system more stable. However, for 50 features, the situation is the opposite, but the proactive strategy can reduce battery consumption more than the reactive strategy. Considering the centroid, **results for both algorithms are rather similar but better for the proactive strategy for FM**

[2]https://www.cs.waikato.ac.nz/ml/index.html

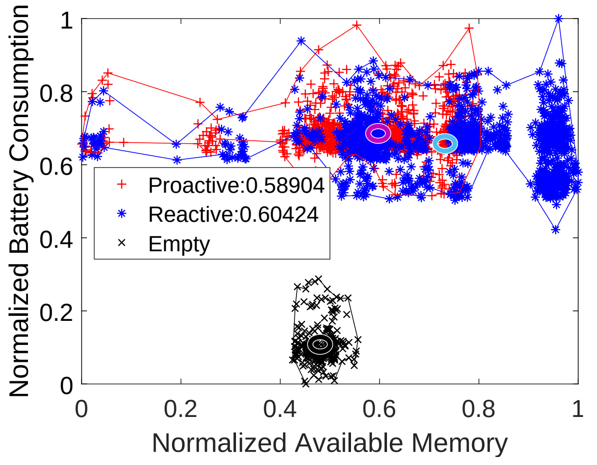

Fig. 4: Scatter plots of the results for FM of 20 features.

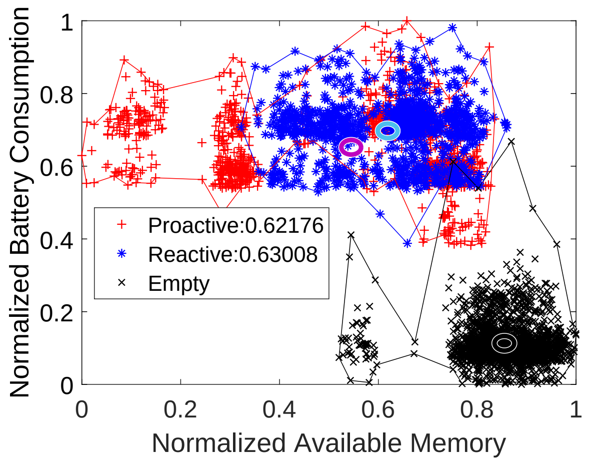

Fig. 5: Scatter plots of the results for FM of 50 features.

of 20 and 50 features, and better for the reactive strategy for FM of 100 features. As we stated in the introduction, the proactive strategy does not improve the results of the reactive one in all scenarios but provides other benefits discussed in RQ3.

RQ2. Are the execution times of ProDSPL adequate for an interactive application? To answer this question, we have measured the execution time of ProDSPL for different prediction horizons from 1 to 10 (see Table I). Experimentation stopped when the response time of the algorithm becomes unacceptable for the reconfiguration of an application that requires user interaction. This limit was reached when the prediction horizon was 12. If we exclude these cases, the execution times are in the order of milliseconds on average. However, in the worst-case execution, time can rise to 4 seconds. In any case, these results make our approach suitable for an interactive IAR application.

We can reduce the response time of our strategy to manage the worst case. Related to this, the real-time community has explored different options [22] that includes exploiting the simple properties of interior point algorithms to execute

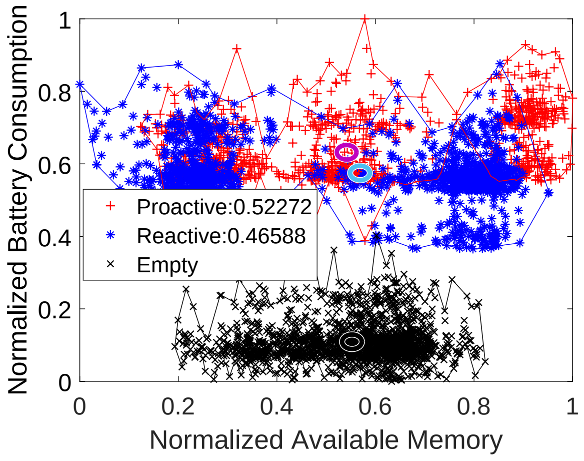

Fig. 6: Scatter plots of the results for FM of 100 features.

the next proposed step by the proactive strategy to avoid computing a new plan or reducing the complexity of the problem. We will explore this issue in future work.

RQ3. When it is more advantageous to use a proactive strategy rather than a reactive strategy for IAR applications?. Considering the results of our experiments, the ProDSPL can generate configurations closer to the Pareto optimal for FMs of 20 and 50 features. However, the execution time is the order of hundreds of milliseconds while MODAGAME can produce solutions in few milliseconds [15]. On the other hand, the number of reconfigurations required for ProDSPL is smaller than the numbers required by MODAGAME. For the experiments presented, ProDSPL maintains the same configuration during the execution of the experiment, while MODAGAME performs 17, 13, and 24 reconfigurations for the models of 20, 50, and 100 features, respectively.

In conclusion, ProDSPL is adequate for interactive applications like IAR because the frequent change of configuration could lead to user dissatisfaction. Finally, we see that the results of both strategies are similar concerning the Pareto optimality. It would be interesting to explore which kinds of FMs are better for each strategy as future work.

V. Conclusions

IAR is a key technology for the vision of Industry 4.0. However, its adoption faces several challenges because it requires the execution of heavy computation tasks in wearable devices (poor in computational resources) in a dynamic operational context. IAR application requires proactive self-adaptation to ensure an immersive augmented reality experience while ensuring computational resources. In this work, we have proposed the application of ProDSPL, which combines proactive control with DSPL, for the self-adaptation of IAR applications. To do so, ProDSPL has been adapted to operate in multi-objective application domains like IAR. We have explored the feasibility of the application of ProDSPL by analyzing the quality of the configurations obtained, the number of reconfigurations needed, and the response time. ProDSPL

TABLE I: Times statistics in milliseconds.

FM	20				50				100			
H	1	4	6	10	1	4	6	10	1	4	6	10
Median	140	239	288	429	267	456	572	805	536	847	1077	1483
Std. Dev.	355	486	295	457	354	329	386	628	631	380	213	553
Min	129	213	270	375	260	440	556	776	492	833	1066	1450
Max	2466	2766	2836	3308	2864	3013	3406	4776	3359	3439	4118	4948

has been compared with MODAGAME, a multi-objective reactive DSPL that generates nearly optimal configurations. Our approach has a reasonable response time and can generate system configurations near the optimal ones for feature models of 20 and 50 features. Additionally, PRODSPL minimizes the number of reconfigurations required to optimize the app.

In this work, we have shown that PRODSPL is advantageous in the self-adaptation of IAR applications. However, in our ongoing work, we are studying its applicability to other IoT systems. With this regard, it is of special interest the influence of the uncertainty, the cost of the reconfiguration, and the impact of the features in the multi-objective function.

ACKNOWLEDGEMENTS

This work is supported by the European Union's H2020 research and innovation programme under grant agreement DAEMON 101017109, 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 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] A. Rojko, "Industry 4.0 concept: Background and overview." *Int. Journal of Interactive Mobile Technologies*, vol. 11, no. 5, 2017.

[2] K. van Lopik, M. Sinclair, R. Sharpe, P. Conway, and A. West, "Developing augmented reality capabilities for industry 4.0 small enterprises: Lessons learnt from a content authoring case study," *Computers in Industry*, vol. 117, p. 103208, 2020.

[3] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[4] Ó. Blanco-Novoa, T. M. Fernández-Caramés, P. Fraga-Lamas, and M. A. Vilar-Montesinos, "A practical evaluation of commercial industrial augmented reality systems in an industry 4.0 shipyard," *IEEE Access*, vol. 6, pp. 8201–8218, 2018.

[5] L. Damiani, M. Demartini, G. Guizzi, R. Revetria, and F. Tonelli, "Augmented and virtual reality applications in industrial systems: A qualitative review towards the industry 4.0 era," in *IFAC Symp. on Information Control Problems in Manufacturing (INCOM)*, vol. 51, no. 11, 2018, pp. 624–630.

[6] P. Fraga-Lamas, T. M. Fernández-Caramés, O. Blanco-Novoa, and M. A. Vilar-Montesinos, "A review on industrial augmented reality systems for the industry 4.0 shipyard," *IEEE Access*, vol. 6, pp. 13 358–13 375, 2018.

[7] G. Orsini, D. Bade, and W. Lamersdorf, "Cloudaware: Empowering context-aware self-adaptation for mobile applications," *Trans. Emerging Telecommunications Technologies*, vol. 29, no. 4, 2018.

[8] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, Mar. 2016.

[9] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 83–92.

[10] K. Toczé and S. Nadjm-Tehrani, "Orch: Distributed orchestration framework using mobile edge devices," in *IEEE Int. Conf. on Fog and Edge Computing (ICFEC)*, 2019, pp. 1–10.

[11] D. Weyns, *Software Engineering of Self-adaptive Systems*. Cham: Springer Int. Publishing, 2019, pp. 399–443.

[12] D. Garlan, S. . Cheng, A. . Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[13] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[14] I. Ayala, A. V. Papadopoulos, M. Amor, and L. Fuentes, "Prodspl: Proactive self-adaptation based on dynamic software product lines," *Journal of Systems and Software*, vol. 175, p. 110909, 2021.

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

[16] C. Quinton, M. Vierhauser, R. Rabiser, L. Baresi, P. Grünbacher, and C. Schuhmayer, "Evolution in dynamic software product lines," *Journal of software : evolution and process*, vol. 33, no. 2, 2021.

[17] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *IEEE Software*, vol. 19, no. 4, pp. 58–65, 2002.

[18] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615 – 636, 2010.

[19] J. Ballesteros, I. Ayala, J. R. Caro-Romero, M. Amor, and L. Fuentes, "Evolving dynamic self-adaptation policies of mhealth systems for long-term monitoring," *Journal of Biomedical Informatics*, vol. 108, p. 103494, 2020.

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

[21] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Flexible and efficient decision-making for proactive latency-aware self-adaptation," *ACM Trans. Auton. Adapt. Syst.*, vol. 13, no. 1, 2018.

[22] K. Angelopoulos, A. V. Papadopoulos, V. E. S. Souza, and J. Mylopoulos, "Engineering self-adaptive software systems: From requirements to model predictive control," *ACM Trans. Auton. Adapt. Syst.*, vol. 13, no. 1, pp. 1:1–1:27, 2018.

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

[24] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[25] J. Cámara, W. Peng, D. Garlan, and B. R. Schmerl, "Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation," *Sci. Comput. Program.*, vol. 167, pp. 51–69, 2018.

[26] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. Schmerl, "Comparing model-based predictive approaches to self-adaptation: CobRA and PLA," in *SEAMS 2017*, 2017, pp. 42–53.

[27] L. Ljung, *System Identification: Theory for the User*, 1999.

[28] P. Hansen, "Methods of nonlinear 0-1 programming," in *Discrete Optimization II*, 1979, vol. 5, pp. 53 – 70.

[29] N. Gavish, T. Gutiérrez, S. Webel, J. Rodríguez, M. Peveri, U. Bockholt, and F. Tecchia, "Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks," *Interactive Learning Environments*, vol. 23, no. 6, pp. 778–798, 2015.

[30] J. Nielsen, *Usability engineering*, ser. Interactive Technologies. Cambridge, Mass: AP Professional.