1. **Adaptive User Interface evaluation By Objective**

This chapter presents our approach to evaluate AUIs. Our approach divided into two main contributions: a) generate evaluation rules, b) propose an automatic tool that detect problem of AUIs. In the first section, we introduce our approach overview. Second, we describe how we formulated the evaluation of AUIs task as a multi-objective optimization problem. Then we describe our proposed tool. First we apply an adjustment mechanism to update the evaluation rules. Then we describe *AUI\_Evaluator* plug-in Eclipse proposed to automate and generic the detection of problem.

* + 1. **Aproach overview**

We propose an approach that evaluates automatically AUIs. Our solution composed of two steps. First, we started by generating automatically evaluation rules. In fact, we formulating this task as a multi-objective optimization problem that using combination of context, quality metrics, and problem to generate the evaluation rules. This approach takes as input set of context (e.g., age, education level, interest, motivation, etc) and set of quality metrics (e.g., density, grouping, sequence, etc) and set of proposed problem that can be detected (e.g., loaded interface, disorder interface, low guidance, etc) , and it generate as output rule composed of two parts. The if-part (premise) includes an intersection between context and quality metric, while the then-part (conclusion) includes the problem. The approach uses base of example to evaluate the possible combination of rules.

Second, we proposed tool which used to detect automatically problems from an adaptive user interface. This tool using rules previously generated, it perform them and detect the existing problem on the interface. This approach takes as input the interfaces to be evaluated, its measure of quality metrics, and evaluation rules, and it generate as output detected problem. The tool proposed is an eclipse plug-in that automates a technique for the assessment of AUIs and the problems detection.

The approach is basing on multi-objective optimization problem which allows exploring a large search space and increasing the probability to generate an optimal solution of evaluation rules. Figure 3.1 illustrate our contribution.

Contexte

Measure

Peoblem

Trace

Adjustemrnt

**Evaluation rules generation**

**1**

**Automatic detection of problem**

Plugin

Source code

MC900331015.WMFMC900331015.WMF

Multi-objective

Search-based

Evaluation

Problem

detected

Measure of quality metrics

**2**

**Figure 3.1:** Proposed approach.

* 1. **Non-dominated Sorting Genetic Algorithm Overview**

During the last years, many Evolutionary Algorithms (EAs) were suggested to solve multi-objective optimization problems. Therefore, multi-objective optimization algorithm searching to generate a set of optimal solution or near-optimal solution in contrasts of mono-objective algorithm which looking for a single optimal solution. These set of near-optimal solution called also non-dominate solution and presented in the objective space with a curve called Pareto front.

Multi-objective optimization problem (MOP) is constrained by more than one objective which are typically a conflecting objective (Deb et al., 2002). And these objective functions can be maximized or minimized.

In addition, MOP create the condidate solution based on non-domination principale. When, the Pareto front is composed of the optimal solutions that doesn’t dominated by any others. So the main keys of the multi-objective optimization problem are the dominance principle and the optimal Pareto front.

**Definition 1: dominance** is calculated by comparing two solutions. According to (Deb., 2011) solution *X* is said to dominate another solution if the both following condition are true:

1. The solution *X* is no worse than *Y* il all objective. The solutions are compared based on their objective function value.
2. The solution *X* is strictly better than *Y* in at least one objective

**Definition 2: optimal Pareto front** *“The Pareto-optimal front corresponds to global optimal  
solutions of several scalarized objectives”* (Deb., 2011).

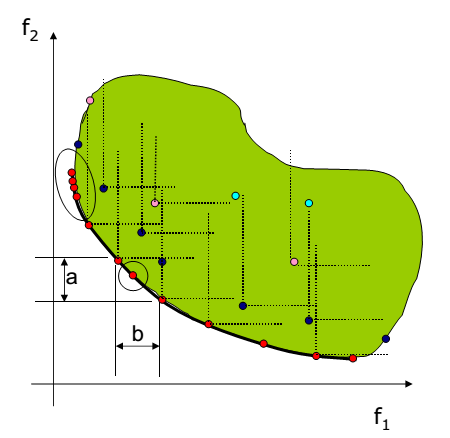
Where:

PF\*: is a Pareto optima front.

F(x): is a given multi-objective problem.

P\*: is a set Pareto optimal generated by the MOP.

Optimal solution



Dominated point d

Pareto front

**Figure 3.2** multi-objective space representations

Several Multi-objective optimization problems were proposed in the literature, they known by their ability to seek to provide a well-convergence and well-diversity by searching many Pareto-optimal solutions (Kessentini et al., 2015). Many Multi-objective EAs has been suggested such us NSGA II. It is one of the most known algorithms that have been suggested to solve multi-objective problem. It is the second version of the “Non-dominated Sorting Genetic Algorithm” (Deb et al., 2002). NSGA II is characterized with:

* **The sorting non-dominated procedure:** in which the solution is sorted based on the level of non-domination into each front.
* **Implementing elitism:** which used to enhance the convergence properties of a multi-objective evloutionary algorithm because it stores all non-dominated solution. Elitism can speed up the performance of the GA and can help to keep the good solutions once they are found.
* **The crowding distance parameter:** which is used to measure distance between each individual and their neighbors. This parameter enhence diversity and spread of solutions. ( **note** we can add how it enhance diversy for example: high Crow.Dist what does means??)
* **The constraints notion:** which allows to modifie the definition of dominance without the use of fittness functions.

NSGA-II uses an evolutionary process to solve problem that constrained by more than one objective which are typically conflecting objective. It provides to coverge toward a set of near-optimal solutions. In what follows, we show the basic steps of NSAG-II represented in the Algorithm 3.1:

1. Firstly, the population P0 is initialized randomly with a set of individual, based on the problem range and constraints if any (line 1).
2. The offspring population Q0 is generated (line 2).
3. Then the offspring population Q0 is combined with the current population P0 in new population R0 (line 5).
4. NSGA II uses Fast-non-dominated-sort algorithm to sort population based on non-domination principle (line 6). This algorithm compares each individual “*r*” with every other individual in the population until it is dominated by one of them. If no individual dominates it, the individual *r* will be considered non-dominated and will be selected by the NSGA-II to be a member of the Pareto front. All non-dominated individual constitute the optimal solutions of the problem (Pareto set ).
5. Once the non-dominated sort is completed the crowding distance calculation process is assigned, where NSGA-II uses a crowding-distance-assignment algorithm to calculate it. In fact, crowding distance of each solution is calculated by finding the Euclidian distance between each solution in the Pareto fron Fi. Crowding distance parameters used to determine diversty, so in this step each solution assigned a diversity score (line 9).
6. The steps 8 to 12 are performed repeatedly until the parent population Pt+1 is filled with N solutions.
7. Once the solutions are ranked based on non-domination and each one of them assigned a diversity score, fitness function can be performed. In fact, the fitness function in NSGA-II represented with couple (rank, crowding distance). Solutions that has the best rank would be emphasized. If solutions has the same rank, them which has a larger crowding distances would be emphasized. The Front Fi sorted in descending order (line 13), and the first (N-|Pt+1|) elements of Fi are chosen (line 14).
8. Finally, the genetic operators (selection, mutation, crossover) are used to create a new population Qt+1(line 15).
9. The steps 4 to 17 are performed repeatedly until some stopping criteria is fulfilled.

1. Create an initial population *P0*

2. Generate an offspring population *Q0*

3. t=0;

4. **while ( ¬** stopping criteria) **do**

5. Rt = Pt ∪ Qt;

6. F = fast-non-dominated-sort (Rt);

7. Pt+1 = ∅ and i=1;

8. **while** | *Pt+1*| +|*Fi*| ≤ *N* **do**

9. Apply crowding-distance-assignment(*Fi*);

10.Pt+1 = Pt+1 ∪ Fi;

11. i = i+1;

12. **end**

13. Sort (Fi, ≺ n);

14. Pt+1 = Pt+1 ∪ Fi[1 : (N-| Pt+1 |)];

15. Qt+1 = create-new-pop(Pt+1);

16. t = t+1;

17. **end**

Algorithm 3.1: High-level pseudo-code of NSGA-II.

* 1. **Non-dominated Sorting Genetic Algorithm Adaptation**

We propose an approach that generates rules to evaluate AUIs. The evaluation of adaptive user interface by considering the context of use is a complex task, and has different constraints. For this reason we proposed to solve it as a multi-object optimization problem using NSGA-II. In this section we are going to introduce how we adopted NSGA-II to solve our problem.

Adapting NSGA-II to evaluate AUIs, achieved by performing its different steps with taken into account our own objectives and constraints. Our method has five phases: the first phase defines how to encoding our solution, the second phase presents the generation of initial solution, in the next phase we describe the fitness function used to evaluate the candidate solutions, then we describe the selection of the fittest solutions, finally we how we change operators to derive new solutions from existing ones.

* + 1. **Solution representation**

We aim to detect quality problem that can be appear in AUIs, to achieve this task we propose to generate rules which would be used to detected the problems. So our individuals should be composed with two parts. The first part (premise) represents an intersection of context and quality metrics which represent the condition. The second part represents the detected problem. In our case an individual is a set of IF – THEN rules:

*IF (condition) THEN (conclusion: problem detected)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IF | Ci = <H, L,M> | ˄ | Mi = <H, L, M> | THEN | Pi |

Ri:

Ci: represent one of the characteristics of the context (user profile, platform, environment)

Mi: represent the quality metrics.

Pi: represent the problem

* + 1. **Creation of the initial population of solutions**

The initial population is created randomly with *N* population. The number of population fixed from the beginning which is based on the problem range and our objective. And each population composed on set of solution (individual) which has different size *S*. The size of solution determined randomly between Min-Solution-Size and Max-Solution-Size values.

* + 1. **Objective functions**

With NSGA II, after creating the solution we should evaluate it, to quantify its ability to solve the problem. Since we have two objectives to optimize, we are using the following two objective functions:

* ***Quality objective function:*** used to evaluate the quality of solution.We considered the solulution has high-qulaity if it detect the maximun of problem. This function is calculated as follow :

n: size of solution.

* ***Size objective function***: which evaluate the solution based on their size. A higher number of rules per solution do not necessarly mean that the result will be better. In other word, a small number of rules can be sufficient to provide good Solution. So this fuction help to minimize the number of rules used to detect the problem.

Therefore, these two objective functions used to generate solutions that has the minimum number of rules which able to detect the maximum number of problem.

* + 1. **Selection**

Once the solution is evaluated, the selection process of NSGA-II is carried out. This algorithm selected individuals by using binary tournament selection which is based on dominance and crowding distance. NSGA-II classifies solutions into different dominance levels. Then, it used comparison-operator to determine crowding distance to select potential individuals having the same dominance level.

* + 1. **Genetic operators**

To better explore the search space, we use this two genetique operator: crossover and mutation:

***Crossover :*** Crossover operator used to generate offspring. In our work we use single point crossover that selecting at random the crossover point and two parent solutions. First, crossover operator starts by selecting the two parents. Then it chooses the cut-point that used to split the parents into two parts. Finally, the two offspring was created, one them composed by the first part of the first parent and the second part of the second parent, the other one composed by the second one of the first parent and the first part of the socond parent. An example is shown in Figure 8

|  |
| --- |
| If (age = H) and (DM >= 0.7)  then woklod UI |
| If (Interest = L) and (SM <=0.2)  then disorder UI |
| If (motivation = L) and (GM <=0.2)  then Low guidance |
| If (age = L) and (RM <=0.2)  then disorder UI |

|  |
| --- |
| If (interest = H) and (DM >= 0.7)  then woklod UI |
| If (education level = L) and (RM <= 0.2) then disorder UI |
| If (Scren Size= L) and (SIM <=0.5)  then complex UI |
| If (Use Experience = L) and (DM<=0.2)  then woklod UI |

Parent 1 Parent 2

|  |
| --- |
| If (age = H) and (DM >= 0.7)  then woklod UI |
| If (Interest = L) and (SM <=0.2)  then disorder UI |
| If (Scren Size= L) and (SIM <=0.5)  then complex UI |
| If (Use Experience = L) and (DM<=0.2)  then woklod UI |

|  |
| --- |
| If (age = H) and (Density >= 0.7)  then woklod UI |
| If (Interest = L) and (Sequence <=0.2)  then disorder UI |
| If (motivation = L) and (GM <=0.2)  then Low guidance |
| If (age = L) and (RM <=0.2)  then disorder UI |

Child 1 Child 2

***Mutation:*** We use mutation operator to modify rule or to delete it from the set of rules. We modifing the rule by changing randomly the value of context, the value of quality metrics. An example is shown in Figure 8

|  |
| --- |
| If (age = H) and (DM >= 0.7)  then woklod UI |
| If (Interest = L) and (SM <=0.2)  then disorder UI |
| If (Scren Size= L) and (SIM <=0.5)  then complex UI |
| If (Use Experience = L) and (DM<=0.2)  then woklod UI |

|  |
| --- |
| If (age = H) and (Density <= 0.4)  then woklod UI |
| If (Interest = L) and (Sequence <=0.2)  then disorder UI |
| If (Scren Size= M) and (SIM <=0.7)  then complex UI |
| If (Use Experience = L) and (DM<=0.2)  then woklod UI |

Before Mutation After Mutation

* 1. **Problem detection**

In order to support the assessment of AUIs, we proposed an automatic tool that detect quality problem. So, this section describes the problem detection process. This process is a generic mechanism for analyzing quality of interface by using evaluation rules. We propose an approach to detect five different defect quality of AUI, namely complex UI, workload UI, disorder UI, low Guidance, and irregular UI. The automation through the use of tools can help to reduce the error of quality metrics calculation and allow analyzing a large number of adaptive interfaces in a quick and repetitive manner. The detection problem process has two steps: 1) First, we need to adjust quality metrics of evaluation rules, 2) then, perform rules and detected the problems.

**Figure 3.3:** Problem detection approach

* + 1. **adjustment of metric**

Quality problem detection has as input the pre-defined evaluation rules and the measurement of quality metrics of interfaces that should be evaluated. As highlighted in the previous section evaluation rules were composed on condition-part which is also composed on two parts context and quality metrics, and the conclusion-part which represent the problem.

The Quality metrics are quantification mechanisms that support the examination of interface component characteristics. They are an important means used to achieve the identification of quality problems of AUIs. Thus, evaluation rules combining metrics with logical operators and thresholds:

IF (age = H) AND (**DM >= 0.8**) THEN Workload UI

IF (Screen Size= M) AND **(SIM <=0.3)** THEN Complex UI

In other words, a quality metric result can be interpreted as a certain symptom of one or more problems. In fact, in the detection process we need to compare the value of quality metrics of interface to be evaluated with an adequate threshold value. The threshold value is represented by evaluation rules.

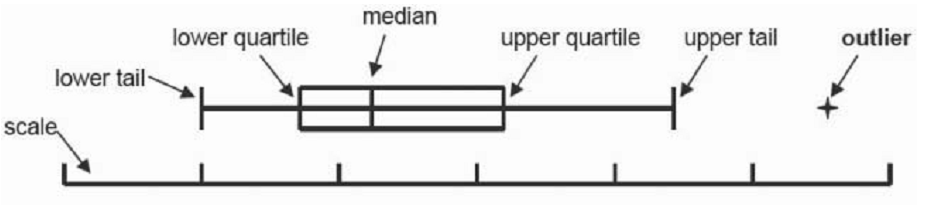
The pre-defined rules were generated automatically based on survey that describe the problem detected by users in three different applications. But to evaluate other interface we need to adjust the threshold of quality metric used in the evaluation rules. Each system has different number of interface and different number of component per interface.

Detection problem will vary depending on the selected value threshold. Increasing the value too much will cause more false negatives, while decreasing it in excess will cause more false positives. So, the use of quality metrics in the problem detection process needs an adjustment mechanism to improve accuracy of proposed tools.

In this context we propose to filter the value of quality metric used in the evaluation rules. “*Filtering is a statistical means by which a subset of the measurement results is extracted based on the particular focus of the measurement, in the context of the detection strategy.*” (Marinescu et al., 2004).

In our work we used box-plot technique to adjust the threshold of quality metrics represented in evaluation rules. It is one significant example of a statistical means for detecting the abnormal values in a data set (Marinescu et Lanza., 2006).

This technique analyze threshold represented by evaluation rules. Box-plot take as input data set that analyze it to compute firstly median value. Then, use it to determine two pair of thresholds, which are the lower and upper quartile. These two thresholds will be used to update the measurement of quality metrics.



**Figure 3.3** Box-plot

* + 1. **Plugu-in: AUI\_Evaluator Toll**

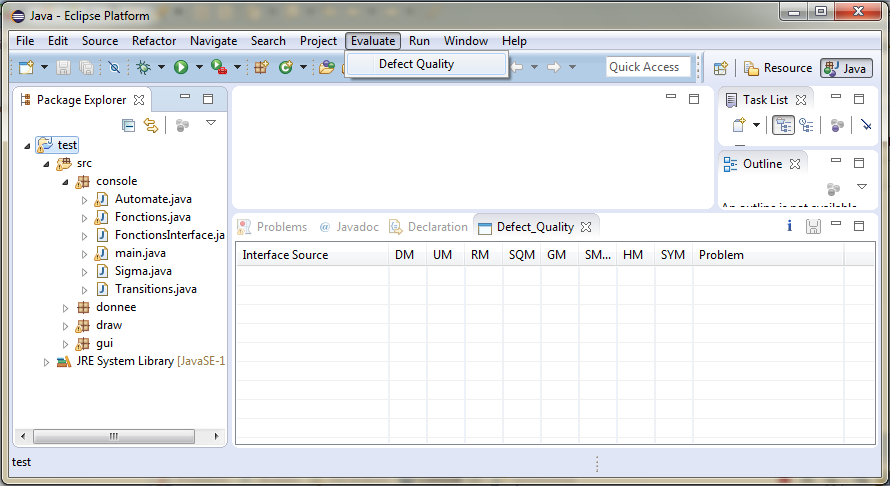
In this subsection we present an Eclipse plug-in *AUI\_Evaluator* that we proposed to detect problem of adaptive interface in Java project. The toll takes as input the adjusted evaluation rules and the measurement of quality metrics of different interface of system to be evaluated and generate as output list of detected problem of each interface.

AUI\_Evaluator extract interfaces from the Java project. For each interface the corresponding context are read, and the corresponding metrics are calculated. Then, we performing the evaluation rules that verify these context. There are two possible conditions:

* Evaluation rule use ≥ operator (greater than or equal to): In this case, if the measurement of quality metrics is above the defined threshold of evaluation rule then the problem is detected in this interface.
* Evaluation rule use ≤ oprator (less than or equal to): In this case, if the measurement of quality metrics is below the defined threshold of evaluation rule then the problem is detected in this interface.

AUI\_Evaluator is a generic and automatic toll that can evaluate any adaptive interface. To use it, the user should imports the adaptive system under study as a Java Project and opens Navigator View in Java Perspective. Then, the user should select the “Evaluate” item in the menu bar and triggers the “Defect Quality” action, which in turn opens the corresponding view. After pressing the “Identify Problem” button the Defect\_Quality view lists the different interface with their quality metrics value and the set of detected problem, as shown in Figure 1. The view contains seven columns:

1. Interface: show the name of interface evaluated.
2. DM : Density metrics measurment.
3. RM: Regularity metrics measurment.
4. SQM: Sequence metrics measurment.
5. GM : Grouping metrics measurment.
6. SMM: Simplicity metrics measurment.
7. Detected problems : showing the set of problem detected.



Action

View

Identify problem

Adaptive System

Figure AUI\_Evaluator output showing identified problem