A Domain-Specific Language for Fault Diagnosis in Electrical Submersible Pumps

Gustavo Epichim Monjardim, Alexandre Rodrigues, Flávio Miguel Varejão, Vítor E. Silva Souza
Federal University of Espírito Santo – Vitória, ES – Brazil
gustavo_monjardim@hotmail.com, alexandre.rodrigues@ufes.br, {fvarejao, vitorsouza}@inf.ufes.br
Marcos Pellegrini Ribeiro
Petroleo Brasileiro SA - Petrobras – Rio de Janeiro, RJ – Brazil
mpellegrini@petrobras.com.br

***Abstract*—Electrical submersible pumps are devices frequently used in off-shore oil exploration. Vibration signals analysis and expert systems technology are used for detecting faults on these motor pumps. Fault diagnosis classifiers may need to be updated or expanded. This paper proposes a domain specific language for enabling non-programmers engineers to create and adjust rule-based fault diagnosis classifiers of electrical submersible pumps.**

***Index Terms*—Fault diagnosis, Electrical submersible pumps, Domain specific language, Expert systems**

I. Introduction

An Electrical Submersible Pump (ESP) is an efficient and reliable artificial-lift method for off-shore oil and gas exploration [1], [2]. This process involves the use of very sophisticated and expensive equipments. The most important component of this system is a multi-stage centrifugal pump driven by electric motors. The installation, removal and replacement of this equipment is highly costly, since it requires the use of a drillship (ship designed for use in exploratory offshore drilling of new oil and gas wells) and the interruption of oil production. Hence, these equipments are carefully examined in tests carried out before the installation.

The pre-installation tests are performed in a controlled environment and aim to diagnose failures in the equipment. In these test procedures, accelerometers are attached to the pump and vibration signals are collected during the operation of the equipment. Two accelerometer sensors are orthogonally (X-axis and Y-axis) attached in several points distributed along the main components of the pump. This distribution results in a total of 36 sensors that simultaneously collect signals of vibration in the time domain. After the collection, the signals are taken to the frequency domain via conventional Fourier Transform to be analyzed by an expert engineer. Based on this analysis, the expert decides whether the conditions of the equipment are suitable or not for installation.

This process of analyzing and deciding about the conditions of the equipment requires a tacit knowledge that is only acquired along years of experience solving similar problems. This knowledge cannot be easily taught or transfered and, as a result, new engineers need years of study and practice to be able to perform the analysis. In order to depersonalize the expert knowledge, computational and statistical methodologies for the storage, processing, visualization and automatic analysis of the vibration signals were developed and packed into an expert system called RPDBCS.[1] Machine learning based classifiers [3]–[7] were developed and implemented in the artificial intelligence module of the system to detect four fault conditions (unbalance, misalignment, rubbing and sensor failure) and the normal condition operation. RPDBCS performs the data analysis automatically and enables non specialized personnel to assess the conditions of the equipment.

The artificial intelligence (AI) of the expert system RPDBCS was built based on interviews and discussions with the domain expert. However, as widespread in the expert system literature [8], the knowledge acquired by the expert is not easily transferred and it is reactive (in the sense that only when faced with a specific situation the knowledge emerges). Therefore, the features used in RPDBCS's AI module may not completely represent the tacit knowledge needed to analyze the data acquired. In fact, something frequently observed throughout this research project is that the human expert eventually changed his opinion about some patterns when confronted to new ones. In addition, new type of faults (or faults not yet considered in RPDBCS) might appear and the classifiers must be updated or expanded.

Since training machine learning classifiers is not an easy task and demands many examples of the new concepts to be classified, it is not likely that domain engineers would be able to adjust and expand the classifiers by themselves. However, this recurrent need of adjustment and creation of new classifiers highlights the need of a high level language with which the engineers may describe the features and patterns relevant for classifying existing faults or new ones.

Whereas general purpose rule based systems like Drools [9] support a declarative, rule-based approach for problem solving, rules are still implemented in a Java-based way, which is difficult to implement and understand for non-programmers. Domain Specific Languages (DSLs) [10] allow for implementing rules in a language that is closer to the user natural language. In a DSL, rules can be written, read, and modified much easier, even by non-programmers.

[1] Acronym for *Reconhecimento de Padrões de Defeitos em sistemas de Bombeio Centrífugo Submerso*, Portuguese for *Fault Pattern Detection in Electrical Submersible Pump systems*.

This paper proposes DSL-FDESP (Domain Specific Language for Fault Diagnosis in Electric Submersible Pumps), a new DSL that allows the construction of rule-based classifiers for diagnosing faults in ESPs based on vibration signals analysis.

The use of DSLs in industrial settings have been proposed in the past. For instance, Preschern et al. [15] present a DSL for modeling automation systems; Consel et al. [16] propose Spidle, a DSL for the development of streaming applications. To the extent of our knowledge, however, a DSL to represent classifiers for fault diagnosis based on vibration analysis has not yet been proposed. Our paper aims to fill this gap.

The remainder of this paper is organized as follows: Section II summarizes the process of detecting faults performed by the engineer expert. Section III describes in detail the DSL, which is validated in Section IV. Finally, Section V concludes and discusses future works.

II. Fault Diagnosis in Electrical Submersible Pumps (ESPs)

To categorize the mode of operation of an ESP, the expert engineer visually inspects a large amount of vibration spectra of frequencies looking for an error pattern. For some kinds of faults (misalignment and unbalance, for example) the faulty pattern must be captured by the sensors in both directions (X-axis and Y-axis) to validate the faulty condition, whereas for other types of failures (rubbing, for example) a single detection of the faulty signature characterize a defective condition.

Typical examples of relevant features from the vibration spectra used by the expert to identify possible faults in the equipment are the rotation frequency, peaks around the harmonics of the shaft rotation frequency and the shape of magnitudes in the low frequencies. For example, Figure 1 illustrates a typical fault signature in the frequency domain for the misalignment fault. Misalignment is characterized by high amplitudes at the first and second harmonics of the shaft rotation frequency.

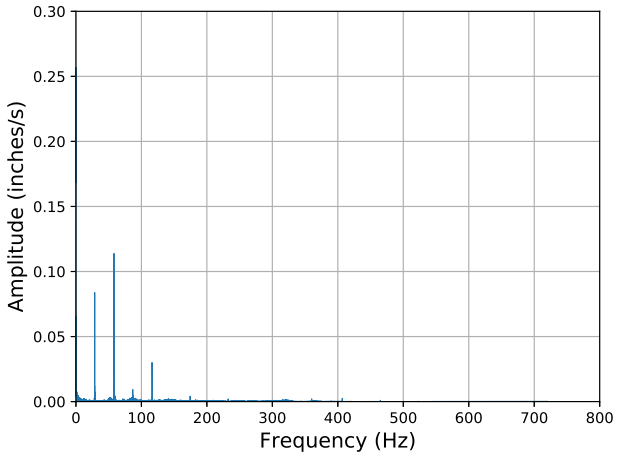

Fig. 1. Fault signature in the frequency domain for the misalignment fault.

III. DSL-FDESP

In this section, we present the proposed Domain-Specific Language (DSL) for fault diagnosis in electrical submersible pumps. Subsection III-A presents the Model-Driven Development paradigm, which was used to build the DSL; subsection III-B provides an overview of the language, how it was designed and how it represents fault classifiers; subsection III-C lists the operators, constants and functions that are part of the DSL; subsection III-D explains how classifiers written in the language are validated; subsection III-E then shows how these classifiers are interpreted and transformed into Java code, in order to integrate them into the expert system RPDBCS.

A. Model-Driven Development

Model Driven Development (MDD) [11] is a software development approach that focuses on models: instead of building software using programming languages separately from the software design artifacts, system functions are specified using models, which can be validated and transformed from one stage to another, until the level of code.

In order for these models to be transformable to code, independent from software, reusable in different contexts and more easily processed by machines in general, they have to be specified with a clear abstract syntax, with well-defined rules for its interpretation. As models play a pervasive role in MDD, the approach advocates the representation of the models themselves as "instances" of some more abstract models, called *meta-models*. Meta-models constitute the definition of a modeling language, given they provide a way of describing the whole class of models that can be represented [12].

Languages defined by meta-models in MDD can be *Domain-Specific Languages* (DSLs) or *General-Purpose Languages* (GPLs). The former are designed specifically for a certain domain, context or company to aid the description of things in that particular domain (e.g., BPMN for business process modeling or HTML for web page design), whereas the latter can be applied to any sector or domain for modeling puposes (e.g., the UML modeling language or the Java programming language) [12].

In another dimension, modeling languages can be textual (HTML, Java) or graphical (BPMN, UML), depending on which form of representation is more adequate for the uses of the language. The choice of visual notation defines the language's *concrete syntax*, whereas its *abstract syntax* is defined by the meta-model. Finally, when designing a language, one should also define (formally or not) its semantics, i.e., the meaning of the elements defined in the language's syntax.

B. Overview of the DSL

The Domain Specific Language for Fault Diagnosis in Electric Submersible Pumps (DSL-FDESP) is a textual DSL that allows domain experts to create new classifiers for fault diagnosis in the expert system RPDBCS.

DSL-FDESP was build with Xtext [13] on top of the Eclipse Modeling Platform [14], which runs on Java. Xtext allows the definition of textual DSLs using a powerful grammar

LISTING 1
GENERAL SYNTAX FOR THE DEFINITION OF A NEW DEFECT CLASSIFIER.

```
1  classifier <id> {
2    name <text>
3    description <text>
4    target <Single | Pair>
5    faultType <FaultType>
6
7    expression <id> {
8      eval <expression>
9    }
10
11   faultEval <expression>
12 }
```

LISTING 2
EXAMPLE OF A CLASSIFIER TO DETECT SMALL PEAK ANOMALIES.

```
1  classifier SmallPeak {
2    name "SmallPeak"
3    description "Detects small peak anomalies."
4    target Single
5    faultType othersSP
6
7    expression Exp1 {
8      eval harmonicPeak(Signal, 1) < 0.07
9    }
10   expression Exp2 {
11     eval harmonicFreq(Signal, 1) > 55
12   }
13
14   faultEval Exp1 && Exp2
15 }
```

language, providing reusable infrastructure composed of a lexical analyzer, which converts a sequence of characters into a sequence of tokens; a parser, which performs syntactic analysis to make sure the sequences of tokens form valid statements; and a standard in-memory representation of the parsed program in a structure called the Abstract Syntax Tree.

Listing 1 shows the general syntax for the definition of a new fault classifier using DSL-FDESP. A valid definition starts with the classifier keyword, followed by an identifier and curly brackets to determine the scope of the definition (lines 1 and 12). Inside this scope, attributes of the classifier are defined in a header (lines 2–5) and the keyword faultEval (line 11) precedes an expression that determines the presence of the fault this classifier will look for (therefore, should evaluate to true or false). For complex expressions, the language allows you to modularize by defining one or more expressions (lines 7–9), assigning them identifiers and reusing them in the definition of other expressions (these can evaluate to non-Boolean values).

In the header, the following keywords are used to specify a few attributes of the classifier being defined:

- name: a short name of the type of fault this classifier will look for, written between "quotes";
- description: a longer description of the type of fault (for human use), also between "quotes";
- target: Single, if the classifier will analyze each signal individually; or Pair, if the signals (X, Y) are analyzed together as a pair;
- faultType: defines the icon that will be shown in the graphical user interface of the RPDBCS tool, to be chosen among the following possibilities: rubbing, unbalance, sensorFault, misalignment, othersDHP, othersM, othersH60P, othersSP, othersAUC, others.

Finally, expressions are written based on the operators and functions presented in the next subsection. As an example, Listing 2 shows the definition of a classifier to detect small peak anomalies. Analyzing each signal individually, this classifier will indicate a possible fault in an electrical pump if the magnitude in its vibration signal's first harmonic is less than 0.07 (represented by expression Exp1) and the rotation frequency of that same harmonic is greater than 55 (Exp2).

C. Operators, Constants and Functions

To compose expressions, DSL-FDESP supports Boolean, relational and arithmetic operators, listed below in order of precedence (lower to higher). Parentheses can be used to change precedence in an expression, in the same fashion they are used in most programming languages. All operators are binary and in-fixed, with the exception of ! (not), which is unary and, despite its position in the list below, has the highest precedence among all operators.

- Boolean operators: ! (not), $\&\&$ (and), $||$ (or);
- Relational operators: $==$ (equal to), $!=$ (not equal to), $>$ (greater than), $>=$ (greater than or equal to), $<$ (lower than), $<=$ (lower than or equal to);
- Arithmetic operators: $+$ (addition), $-$ (subtraction), $*$ (multiplication), $/$ (division).

As operands, the language allows the use of constants (such as Signal in Listing 2) and functions (such as harmonicPeak and harmonicFreq in Listing 2), all of them elicited from domain experts in the area of fault diagnosis in electrical submersible pumps. Table I lists the available functions, along with their expected argument and their semantics. The available constants, which can also be used as arguments in functions, are listed below:

- Signal: a single vibration signal, in any axis, used only in classifiers of target Single;
- SignalX, SignalY: vibration signals in the X and Y axis, respectively, used only in classifiers of target Pair;
- Signal.rotation: the rotation frequency of the vibration signal;
- Acquisition: the set with all 36 vibration signals from an electrical submersible pump.

An important aspect regarding operators, constants and functions is their type. DSL-FDESP uses four types inherited from the Java platform, namely Boolean, Double, Double[] (array of Double) and Integer; two classes created specifically for the DSL, Signal and Acquisition; the enumerated type FlowType with values MAX, MIN and BEP (Best Efficient Point) representing acquisition flows; and, finally, List⟨Signal⟩, a list of Signal objects using Java's Collection API.

TABLE I
FUNCTIONS AVAILABLE IN DSL-FDESP (IN ALPHABETICAL ORDER), WITH THEIR EXPECTED ARGUMENTS, SEMANTICS AND TYPES.

Function name	Function arguments	Semantics	Type
acquisitionInterval	Acquisition, p (Double)	This function first calculates a matrix with frequencies and modes of an Acquisition. Then it extracts the modes and returns an array containing two numbers representing an interval of frequencies. For instance, if there are 20 signals with $33.2Hz$ and 20 signals with $34.5Hz$, it will return $[33.2 \times (1-p),\ 34.5 \times (1+p)]$.	Double[]
checkFlow	Acquisition, FlowType	True if Acquisition has the indicated FlowType (BEP, MAX, MIN), otherwise false.	Boolean
contains	signalList (List⟨Signal⟩), Signal	True if a Signal is contained in signalList, false otherwise.	Boolean
faultySensorKNN	harmonicPeak1x (Double), harmonicPeak2x (Double), quadraticError (Double)	True if there is a sensor fault, false otherwise, according to a K-Nearest-Neighbor algorithm (usually, $K = 1$) comparing the harmonicPeak1x, the harmonicPeak2x and the quadraticError with records of sensor faults from the past in the database.	Boolean
findPeak	Signal, interval (Double[])	The highest Signal amplitude in the given frequency interval.	Double
frequencyInterval	firstFrequency, lastFrequency	An array with two elements [firstFrequency, lastFrequency] representing an interval of frequencies, which is an input required by many other functions.	Double[]
frictionError	Signal, interval ($[f_0, f_n]$), slope (Double), intercept (Double)	$\frac{\sum_{i=f_0}^{f_n} [ln(Signal.peakList[i]) - (slope \times Signal.freqList[i] + intcpt)]}{f_0 - f_n}$ (1)	Double
frictionSearch	Signal, interval	Proportion of times that the Signal median magnitude grows considering 30 equally divided parts in the given frequency interval.	Double
harmonicFreq	Signal, harmonic (Integer)	Signal rotation frequency in the indicated harmonic.	Double
harmonicPeak	Signal, harmonic	Signal magnitude in the indicated harmonic.	Double
logisticRegression	target (Double), beta0 (β_0, Double), beta1 (β_1, Double)	$\frac{e^{(\beta_0+\beta_1 \times target)}}{1+e^{(\beta_0+\beta_1 \times target)}}$ (2)	Double
medianMagnitude	Signal, interval	Median of the Signal magnitudes in the given frequency interval.	Double
peakThreshold	Signal, interval, threshold (Double)	True if there is a peak above the indicated threshold in the given frequency interval of the Signal's shaft rotation frequency, otherwise false.	Boolean
quadraticError	Signal, interval, partition (Integer)	The quadratic error of the set composed by the highest Signal amplitudes for every partition of the given frequency interval.	Double
regressionA	Signal, interval	Intercept (a) of the linear regression of the logarithm of the Signal's frequency magnitudes (Mag) over the interval of frequencies (Fq), c.f. Equation (3). $log(Mag) = a - b \times Fq$ (3)	Double
regressionB	Signal, interval	Slope (b) of the linear regression of the logarithm of the Signal's frequency magnitudes (Mag) over the interval of frequencies (Fq), c.f. Equation (3).	Double
rMS	Signal, interval	Root mean square of the Signal magnitudes in the given frequency interval.	Double
signalsOutOfInterval	interval (Double[]), Acquisition	A list with all signals containing magnitudes that are outside an Interval.	List⟨Signal⟩
verifyAreaUnderCurve	Signal, interval	Size of the area under the curve of a Signal's vibration chart, in the given frequency interval.	Double

Starting with the constants, Signal, SignalX and SignalY are defined as being of type Signal; analogously, the Acquisition constant is typed Acquisition; lastly, Signal.rotation is of type Double. The type of each function and their arguments is shown in Table I. Finally, as intuitively expected, Boolean and relational operators are of Boolean type, whereas arithmetic operators are typed Double.

The definition of this small type system for the DSL is the first step towards the language's validation and interpretation, which are presented next.

D. Model Validation

DSL-FDESP performs a series of validations on classifiers written in the language, displaying error messages to users if any problems are encountered. Some of these validations are automatically performed by Xtext. For instance, it checks that the syntax of the language (as shown in Listing 1) was respected. It also verifies that all identifiers are unique, i.e., there are no two classifiers, or even two expressions within a classifier, with the same name.

LISTING 3
VALIDATION METHOD FOR FUNCTION HARMONICPEAK.

```
1 def checkType(HarmonicPeak harmonicPeak) {
2   checkExpectedSignal(harmonicPeak.signal,
        RPDBCSPackage.Literals.HARMONIC_PEAK__SIGNAL)
3   checkExpectedInt(harmonicPeak.harmonic,
        RPDBCSPackage.Literals.HARMONIC_PEAK__HARMONIC
        )
4 }
```

A few validation methods were implemented specifically for the DSL. For each function and operator of the language, there is a method that uses the type system described in the previous subsection to check if the arguments of such functions/operators are of the expected types. For instance, the harmonicPeak function expects as arguments a Signal and a harmonic (of type Integer), therefore its validation method checks that the expressions passed as parameters to this function satisfy these constraints. Listing 3 shows how this validation method was implemented in the DSL.

Another validation method implemented for the DSL concerns the target keyword at the header of the classifier. When the target is defined as Single, meaning that the classifier will analyze vibration signals individually, constants SignalX and SignalY should not be used, as they refer to the X and Y parts of a pair of signals. Analogously, when the target is defined as Pair, the Signal constant cannot be used as the user must specify which axis from the pair is being used in the expression.

E. DSL Interpretation

Once a classifier has been validated, it is ready to be interpreted. For DSL-FDESP, this means generating Java code that evaluates the expressions written in the DSL using the data available in the RPDBCS tool.

For each classifier defined in the language, the interpreter generates a corresponding Java class using the classifier's identifier as name for the class. Inside this class, the interpreter generates the following methods:

- Methods that return the classifier's name, description and faultType (information from the classifier's header, to be used in the tool's graphical user interface);
- A method for every expression defined in the classifier, using the expression's identifier as method name. Its return type depends on the type to which the expression evaluates;
- The method faultEval() with return type Boolean, which refers to the classifier's fault expression (the one after the faultEval keyword);
- The method classify(SignalXY pair), which is the method used by the RPDBCS tool when applying this classifier to the data.

The classify() method receives as parameter a SignalXY pair, i.e., a pair of signals, one in the X axis and one in the Y axis. For classifiers with target Single, this method will call the faultEval() method twice, once for each signal of the pair, as the fault may be identified in one of the signals independently. On the other hand, for classifiers with target Pair, the classify() method will call faultEval() only once, as the fault is identified by analyzing the signals as a pair. The classify() method returns a list of Fault objects, one for each fault identified, empty if no fault is found.

The contents (Java implementation) of the faultEval() and other expression methods consist of a mapping that the interpreter performs from DSL-FDESP to Java, using the API provided by RPDBCS. The interpreter works recursively when confronted with expressions that use operators, such that a call to $interpret\,(\langle left\rangle\ \langle operator\rangle\ \langle right\rangle)$ is resolved as $interpret\,(\langle left\rangle)\,interpret\,(\langle operator\rangle)\,interpret\,(\langle right\rangle)$, which breaks expressions down to the level of operators, constants and function calls. Operators are interpreted as their obvious Java counterparts, whereas constants and function calls are mapped to methods in RPDBCS's API.

For instance, the DSL constant Signal.rotation is mapped to function call dslUtils.signalRotation(s), where dslUtils is an instance of RPDBCS class DSLUtils, s is the signal being analyzed (received as argument by classify() and passed along as parameter in all method calls it is needed) and signalRotation() is a method from DSLUtils that, given a signal, returns its rotation.

LISTING 4
INTERPRETATION (IMPLEMENTATION) FOR CLASSIFIER SMALLPEAK.

```
1  class SmallPeak extends BinarySignalClassifier {
2    DSLUtils dslUtils = new DSLUtils();
3
4    public SmallPeak() {
5      super(FaultType.othersSP);
6    }
7
8    public List<Fault> classify(SignalXY s) {
9      List<Fault> fs = new ArrayList<>();
10     Fault f = classify(s.getX());
11     if (f != null) fs.add(f);
12     f = classify(s.getY());
13     if (f != null) fs.add(f);
14     return fs;
15   }
16
17   private Fault classify(Signal s) {
18     if (s == null) return null;
19     if (faultEval(s)) {
20       Fault f;
21       f = new Fault(s, getRecognizableFaultType());
22       f.setDescription(this.getDescription());
23       return f;
24     }
25     return null;
26   }
27
28   public boolean faultEval(Signal s){
29     return (Exp1(s) && Exp2(s));
30   }
31
32
33   public boolean Exp1(Signal s){
34     return (dslUtils.harmonicFreq(s, 1) > 55);
35   }
36
37
38   public boolean Exp2(Signal s){
39     return (dslUtils.harmonicPeak(s, 1) < 0.07);
40   }
41
42   public String getName() { return "Small Peak"; }
43   public String getDescription() {
44     return "SmallPeak";
45   }
46 }
```

For instance, Listing 4 shows an excerpt (parts of the code omitted for brevity) of the interpretation of classifier SmallPeak, defined in Listing 2, i.e., the Java code generated by DSL-FDESP in order to integrate this classifier into RPDBCS. Expressions Exp1 and Exp2 derive homonymous methods (lines 33 and 38), which are used by faultEval() (line 28) to determine if there is a fault in a signal. Being of target Single, the faultEval() method is called twice, once for each axis in the pair, as can be seen in classify(SignalXY) (line 8).

IV. VALIDATION

To evaluate the DSL, we used data from several sweep tests that have been conducted on ESPs belonging to the RPDBCS project's data set. A sweep test consists on feeding the pumps with frequencies $40Hz$, $45Hz$, $50Hz$, $55Hz$ and $60Hz$ and, for each frequency, conducting tests with flows

TABLE II
CLASSIFIERS IMPLEMENTED WITH DSL-FDESP.

Classifier	Fault description	DSL Dependencies
Acquisition-Mode	A signal with a rotation frequency in the first harmonic outside an interval, which is calculated by first getting the mode of frequencies in the acquisition then applying $[mode - 1\%, mode + 1\%]$ to create the interval.	Signal, Acquisition, acquisitionInterval, signalsOutOfInterval, contains
Area-Under-Curve	The size of the area under the curve of a signal's vibration chart between the interval of frequencies $[3, 10]$ is greater than 0.21.	Signal, verifyAreaUnderCurve, frequencyInterval
Distant-High-Peak	A high peak between the interval of frequencies $[2x, 1000]$.	Signal, Signal.rotation, peakThreshold, frequencyInterval
Faulty Sensor	High noise and absence of significant amplitude at the rotation frequency.	Signal, harmonicPeak, quadraticError, frequencyInterval, faultySensorKNN
Friction	Rubbing that occurs due to contact between stationary and rotating surfaces, which leads to low frequency noise with amplitudes decaying exponentially.	Signal, Signal.rotation, frictionError, frictionSearch, frequencyInterval, logisticRegression
High60-Peak	A high peak between the interval of frequencies $[59.2, 60.2]$.	Signal, Signal.rotation, peakThreshold, frequencyInterval
Misalignment	A high peak in the $1x$ frequency, where x is the rotation frequency of the shaft.	SignalX, SignalY, harmonicPeak, logisticRegression
Small-Peak	A magnitude in the first harmonic of the rotation frequency smaller than 0.07 and rotation frequency greater than $55Hz$.	Signal, harmonicFreq, harmonicPeak
Unbalance	An abnormal high peak in the $2x$ frequency.	SignalX, SignalY, harmonicPeak, logisticRegression

MIN, MAX and BEP (Best Efficiency Point), as specified for the equipment.

All of the test data was imported into RPDBCS and each test was diagnosed using the expert tool's original, built-in classifiers. Each diagnosis was recorded, establishing a benchmark against which the DSL would be tested. Then, the original classifiers were removed from RPDBCS and replaced by the ones generated by DSL-FDESP and the test data was imported once again for diagnosis. The DSL classifiers produced the same diagnostic results as the benchmark, with no significant difference with respect to performance.

A total of nine built-in classifiers were replaced by DSL-FDESP versions, which were validated by the procedure described above. Table II lists the classifiers that were implemented, describing the faults they diagnose and specifying the DSL elements they use to perform such diagnosis.

V. CONCLUSIONS

This paper presented DSL-FDESP, a new DSL for allowing engineers without background in software development to adjust or add new classifiers in the domain of ESP fault diagnosis. The DSL was validated by replacing the original expert system's classifiers by the ones generated using the DSL. The DSL classifiers produced the same diagnostic results with similar computational performance.

Currently, the DSL generates Java code that is manually integrated to the system. One future work consists of implementing the automatic integration of this Java code for not requiring a Java programmer for performing this task. Another future work consists of evaluating the language usability, i.e. if the language is intuitive and easy to be used by the expert engineers. Tests should also evaluate the classifiers scalability, i.e., if the classifiers performance scales well to large datasets.

ACKNOWLEDGMENT

Work supported by CENPES-Petrobras under Grant Termo de Cooperação 0050.00070332.11.9 Petrobras-UFES.

REFERENCES

[1] H. Toliyat, S. Nandi, S. Choi, and H. Meshgin-Kelk, *Electric Machines: Modeling, Condition Monitoring, and Fault Diagnosis*. CRC Press, 2016.

[2] H. Pasman, *Risk Analysis and Control for Industrial Processes - Gas, Oil and Chemicals: A System Perspective for Assessing and Avoiding Low-Probability, High-Consequence Events*. Elsevier Science, 2015.

[3] T. W. Rauber, F. M. Varejao, F. Fabris, M. Pellegrini, and A. Rodrigues, "Automatic diagnosis of submersible motor pump conditions in offshore oil exploration," in *39th Annual Conference of the IEEE Industrial Electronics Society*. Vienna, Austria: IEEE, nov 2013, pp. 5537–5542.

[4] F. Boldt, T. W. Rauber, F. M. Varejao, and M. Pellegrini, "Performance analysis of extreme learning machine for automatic diagnosis of electrical submersible pump conditions," in *12th IEEE International Conference on Industrial Informatics*. Porto Alegre, Brazil: IEEE, jul 2014, pp. 67–72.

[5] ——, "Fast feature selection using hybrid ranking and wrapper approach for automatic fault diagnosis of motorpumps based on vibration signals," in *13th IEEE International Conference on Industrial Informatics*. Cambridge, UK: IEEE, jul 2015, pp. 127–132.

[6] T. Oliveira-Santos, T. W. Rauber, F. M. Varejao, L. Martinuzzo, W. Oliveira, M. P. Ribeiro, and A. Rodrigues, "Submersible Motor Pump Fault Diagnosis System: A Comparative Study of Classification Methods," in *28th International Conference on Tools with Artificial Intelligence*. San Jose, CA, USA: IEEE, nov 2016, pp. 415–422.

[7] F. Boldt, T. W. Rauber, F. M. Varejao, T. Oliveira-Santos, A. Rodrigues, and M. Pellegrini, "Binary Feature Selection Classifier Ensemble for Fault Diagnosis of Submersible Motor Pump," in *26th International Symposium on Industrial Electronics*. Edinburg, UK: IEEE, jun 2017, pp. 1807–1812.

[8] J. Cullen and A. Bryman, "The knowledge acquisition bottleneck: time for reassessment?" *Expert Systems*, vol. 5, no. 3, pp. 216–225, 1988.

[9] M. Proctor, "Drools: A rule engine for complex event processings," in *4th International Conference on Applications of Graph Transformations with Industrial Relevance*. Springer, 2012, pp. 2–2.

[10] M. Fowler, *Model-Driven Software Engineering in Practice: Second Edition*. Addison–Wesley, 2011.

[11] O. Pastor, S. España, J. I. Panach, and N. Aquino, "Model-driven development," *Informatik-Spektrum*, vol. 31, no. 5, pp. 394–407, 2008.

[12] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice: Second Edition*. Morgan & Claypool Publishers, 2017.

[13] L. Bettini, *Implementing Domain-Speci c Languages with Xtext and Xtend: Second Edition*. Packt Publishing, 2016.

[14] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed. Addison-Wesley Professional, 2009.

[15] C. Preschern, A. Leitner, and C. Kreiner, "Domain-Specific Language Architecture for Automation Systems: An Industrial Case Study," in *Workshop on Graphical Modeling Language Development at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012)*, Lyngby, Denmark, 2012.

[16] C. Consel, H. Hamdi, L. Réveillère, L. Singaravelu, H. Yu, and C. Pu, "Spidle: A DSL Approach to Specifying Streaming Applications," in *2nd International Conference on Generative Programming and Component Engineering (GPCE 2003)*. Springer, 2003, pp. 1–17.