VI SBQEE

21 a 24 de agosto de 2005

Belém – Pará – Brasil

Código: BEL 11 7752
Tópico: Modelagem e Simulações

A NEW FRAMEWORK FOR EVALUATING PQ EVENT IDENTIFICATION TECHNIQUES BASED ON ATP SIMULATIONS

ANDREIA SANTOS — LaPS-UFPA
JOSÉ BORGES — LaPS-UFPA
ANTONIO CARVALHO — GSEI-UFPA
MARCUS NUNES — GSEI-UFPA
SURYA SANTOSO — Univ. Texas, USA
ALDEBARO KLAUTAU — LaPS-UFPA

ABSTRACT

Real-world power quality (PQ) events usually are infrequent and composing a dataset of significant size is a difficult task. Using simulators to generate data in well-controlled conditions is a very useful and popular option. This work presents a framework based on the ATP (Alternative Transient Program) simulator, which allows for evaluating parameter extraction (e.g., wavelets) and data mining techniques. This paper concentrates on describing its use in fault identification.

PALAVRAS-CHAVE

Pattern recognition, ATP simulation, Time series classification, fault identification, data mining.

1.0 INTRODUCTION

Boosted by advances in fields such as digital signal processing and machine learning, current research in PQ covers a wide range of algorithms. Unfortunately, the area lacks freely available and standardized benchmarks and, consequently, comparisons among different algorithms are problematic. This situation is shared with other areas, such as time-series mining [1], but there is an aggravating element in PQ: real-world events usually are infrequent and composing a dataset of significant size is a time-consuming and costly task. Hence, the datasets of PQ events are often proprietary and reproducing previously published results is tricky or even impossible. This work presents a simulation-based framework that circumvents these problems and allows data mining of PQ events [2, 3].

In summary, this work presents a free software, called AmazonTP, which runs on top of the well-known ATP simulator. Among other things, the user can specify relatively few parameters and the software automatically generates (a possibly large number of) events and saves them into files. The software is an important part of a framework that has been developed at the Federal University of Pará (UFPA) using ATP-generated data.

There are many works in the literature that use ATP. The correspondent problems are, for example, automatic fault identification and relay testing [4-9]. Some of these follow the same methodology advocated in this work: given the lack of real-world data, “artificially” generate training and test files, and then evaluate techniques such as neural networks. However, there is still plenty of room for improving such methodology.

When the process of “driving” ATP is not easy and flexible enough, the users tend to create a relatively small database of events, and figures or merit such as the error rate cannot be estimated with an appropriate level of statistical significance. For example, in [7, 8], all the simulations led to zero errors, which could eventually hide interesting observations about specificities of the algorithms. Therefore, tools for efficiently creating a rich set of events are very needed and this works aims to contribute along this line.

The AmazonTP, as part of the proposed framework, makes much easier to compare and evaluate novel techniques. Besides, the labeled datasets can be made freely available, which is an incentive to reproducing results and facilitate scientific collaboration.

This work is organized as follows. Section 2 briefly describes the proposed framework for data mining of time series. In Section 3, the module responsible for easing the generation of events is discussed. Section 4 presents an example of using the framework and is followed by the conclusions.

2.0 PROPOSED FRAMEWORK

The whole framework tries to make easier the comparison of algorithms, such as neural network and Bayesian classifiers. The next subsection discusses the framework in a more general scenario, and after that it is particularized to the identification problem.

2.1 Mining time series

The framework allows for evaluating parameter extraction techniques, such as wavelets, and data mining algorithms, such as identification (also called classification) and clustering, as indicated in Figure 1. The PQ events are generated and organized by AmazonTP, which repeatedly invokes ATP to generate each event. The process depends on an ATP *master* file, which can be generated, for example, with ATPDraw. In this work, it was adopted ATPDraw version 4.1 and the Windows version of ATP tpbig.exe (Watcom, source code of December 2003).[i] AmazonTP modifies the master file to create a new ATP file, which is then used as input to the ATP simulator in order to obtain the correspondent new event.

The raw data (waveforms) of each PQ event is typically converted into new parameters. This can be done with Mathwork's Matlab (see, e.g., [10]). Alternatively, the user can bypass Matlab and invoke parameter extraction algorithms supported by the AmazonMiner, which has been developed at UFPA for mining time series and relies on Weka[ii] [12].

Both softwares AmazonTP and AmazonMiner were written in Java, according to the object-oriented paradigm. Through a DLL, AmazonMiner can pass parameters to Matlab, invoke Matlab routines and retrieve their results. Hence, besides feature extraction, Matlab can perform pattern recognition (similarly to [7]) and plot results. In such cases, AmazonMiner can be used to simply automate the process. Alternatively, all data mining operations can be performed through AmazonMiner (useful for users that do not have Matlab).

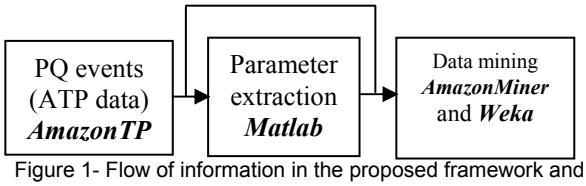

Figure 1- Flow of information in the proposed framework and the correspondent software.

Figure 2 (adapted from [4]) summarizes the process and indicates the files that are generated by each stage. The final result of AmazonTP is a set of amf files. Each amf file stores all sequences generated by one ATP simulation. The waveform samples are stored as real numbers, represented as the primitive type *float* in Java (more specifically: big-endian, 32-bits, IEEE-754 numbers). A header in text (ASCII) format precedes these samples. Among other things, the header indicates the sampling frequency, name of each sequence (based on its ATP name), number of events in each file, their location in time and *label* (description).

When comparing techniques or promoting one, in order to make the conclusions as general as possible, it is useful to test them with PQ events generated with different circuits (different ATP master files). In this case, the user may help by choosing the same names for the sequences of interest. If that is not possible, AmazonTP has a post-processing module that allows to rename the sequences stored in amf files. This module is also useful when the user wants to rename sequences representing real-world events obtained, e.g., from digital fault recorders (DFR).

[i] Both ATPDraw and ATP are available, for registered users (i.e., a password is required), from www.eeug.org.

[ii] Freely available at *http://www.cs.waikato.ac.nz/ml/weka/*.

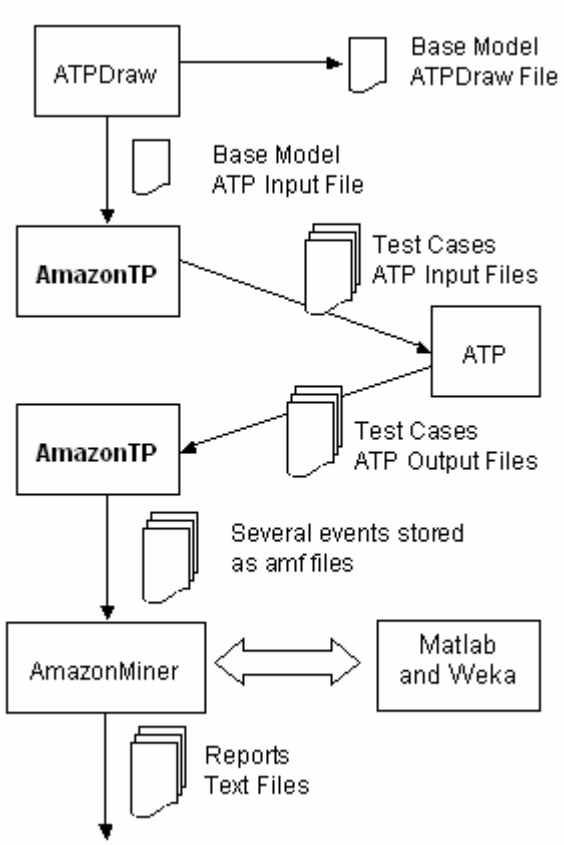

Figure 2- The files generated along the process.

The next subsection specifically discusses identification, which is one of the data mining tasks that the framework supports and helps to better illustrate its use.

2.2 Identification of Events

The proposed framework is capable of dealing with *sequence* data (also called *time series*) of variable length [1]. In contrast, most data mining packages, such as Weka, restrict the events to be represented by a fixed-length vector. In order to discuss such issue, some useful definitions are presented.

A sequence x of length (or duration) N is a discrete-time signal with samples x[n], n=0, 1,…, N-1. Each sequence is associated to a sampling frequency f_s. For a given experiment, it is assumed that all sequences share the same value of f_s. This condition can be imposed by using Matlab to resample the sequences that have f_s different than the chosen one.

As stated before, AmazonMiner allows for comparing sequences of different lengths. This can be done in two different ways. The first option is to use hidden Markov models (HMM) or dynamic time-warping (DTW) (see, e.g., [11]), which are algorithms that straightforwardly support input vectors with various lengths.

Alternatively, *windowing* can be used to convert each sequence into one or more subsequences (vectors) w of a chosen length L called *window* or *frame*. For example, in [7], each ATP simulation generated 6 sequences (current and voltage for the 3 phases), and the input of neural networks consisted of L=30 samples, obtained by extracting 5 samples from each sequence. Besides L, AmazonMiner allows the user to specify the window shift S, which is the difference, in terms of number of samples, between the index n of the first samples of two consecutive windows, and controls the amount of overlap between them. For a sequence of duration N, there are 1+floor((N-L)/S) subsequences z_i, where floor is a function that truncates its argument to an integer.

When the user chooses the windowing option, AmazonMiner performs the operation over all sequences in the experiment and generates a unique Weka file with its ARFF extension. For generating training and test ARFF files, the user should use apply the same windowing procedure to two disjoint sets of sequences. At this stage, the time-series problem becomes a conventional pattern recognition problem, which is based on a set of *examples* (z, y) for training and another for testing (a third set, called *validation*, can be used for model selection) [12]. For regression or identification problems, the label y is a real or integer number, respectively. For unsupervised learning, such as clustering, Weka internally represents y as an interrogation mark when it is undefined.

The trickiest part of windowing a sequence that represents one or more PQ events is the proper labeling of each example (z_i, y_i). For instance, assuming a fault started in z_k and its label is 2, corresponding to, e.g., a AT-fault (phase A to ground), it may be the case that subsequences z_{k+1} and z_{k+2} also have label 2, while all other subsequences have label 0 (e.g., No-fault). AmazonMiner provides some alternatives for labeling, which are all based on the information obtained from the header of each amf file. For example, giving a fault occurred in z_k, all neighboring subsequences could be associated to a label y_1 while the others to some other label y_2.

After creating Weka ARFF files, the user can count on several state-of-art identification (and other data mining) algorithms. Some of the most

prominent are: neural networks, support vector machines, Bayes' classifiers and decision trees.

AmazonMiner automatically evaluates the techniques, generating results such as misclassification (error) rate and standard statistical tests (e.g., t-paired and McNemar's) for the given significance level [12].

The next section describes the main module of AmazonTP, which is responsible for creating ATP files representing events of interest.

3.0 EVENTS GENERATION MODULE

Given a network represented by an ATP master file, GenEvent is the module of AmazonTP that generates files for ATP simulations. GenEvent provides the user with two different ways for generating events, which are called *user-defined* and *semi-automatic*. The former requires more (manual) intervention, while the latter tries to be "smart" enough to create events with minimal interaction with the user.

The user-defined generation is very similar to the BGEN (batch generator) software described in [4]. Among other differences, it should be noticed that BGEN is not in public domain, in contrast to AmazonTP.

GenEvent organizes the interaction with the user through submodules called *agents* (unfortunately, an overly used term in computing). Each agent is responsible for generating a specific PQ event. For example, the shunt-fault agent helps the user to generate a specified number of such events. When designing the experiment, the user can count on more than one agent.

Given an ATP master file, AmazonTP invokes a module called Parser, which interprets the file, and organizes the information about the circuit in a way that can be conveniently manipulated by agents' algorithms. The agent can retrieve information such as the number of nodes and their labels, manipulate such data and pass to the module FilesWriter, which saves an ATP input file and starts organizing the header of the correspondent amf file.

Splitting the task of generating events among agents helps to smoother the software evolution. For example, a user with knowledge of Java programming can incorporate its own agent to attend a specific need. Also, some components of ATP are harder to deal with, and support to these components can be incrementally added. For example, an agent can give up of supporting a JMarti LCC in favor of a simpler algorithm that deals only with distributed Clarke lines. The next paragraph discusses a more concrete situation, of user-defined fault generation.

Figure 3 shows a dialog window displayed by this agent, which was made equivalent to the one in [4]. It generates user-defined events as it relies on user input for defining all parameters related to a fault. In other words, everything is under the user control and there is no randomness in the process. Based on information provided by the Parser, the agent allows the user to select the fault location, its type, and the parameters of the RLC and switch components that will be used to simulate the fault using ATP. Each simulation parameter is a real number (resistance, time, etc.) or a string (AT_fault, BT_fault, etc.), which can be represented by an integer number. For each real number r, the user selects the initial value r[0], the increment (or step-size) Δ and the number λ of samples for this parameter, such that $r[n] = r[0] + \Delta n$, for $n=1, 2, \ldots, \lambda-1$. Considering one can vary R parameters and r_d is the d-th dimension, user-defined simulations typically create one ATP fie for each point in the Cartesian product $r_1[n] \times r_2[n] \times \ldots r_R[n]$, where $r_d[n] = r_d[0] + \Delta_d n$, for $n=0, 1, \ldots, \lambda_d-1$.

In contrast, an agent that uses the semi-automatic behavior releases the user from the task of explicitly specifying all the points in the Cartesian product, and relaxes the restriction of having uniformly sampled parameters.

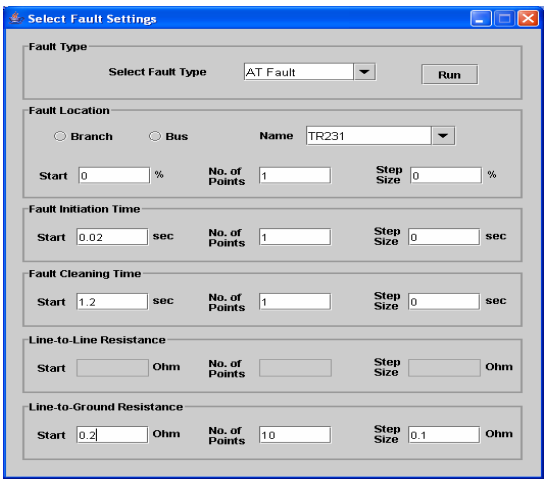

Figure 3- Dialog window of the user-defined fault generation agent. The user can select the type of fault, its location and other parameters.

The semi-automatic generation allows the user to specify a uniform or Gaussian probability density function for sampling a parameter. That is, the user can choose to create, for example, an experiment varying the location of a fault according to a Gaussian with a given mean (e.g.,

50% of a line length) and variance. This helps to include expert-knowledge of the network operation through previously collected statistics. Another aspect that improves the user interaction upon the user-defined behavior (as in [4]) is the following. The agent assumes by the default that the fault can occur at any node or line of the circuit, and the user can eventually select exceptions. Based on that, the user can simply say, for example, the experiment will have 1000 faults uniformly spread over the network. In such cases, AmazonTP tries to find a proper way of automatically labeling the events based on the network topology. If these labels are not adequate, the user can rename them in a post-processing stage.

This work mainly discusses the generation of events given an ATP master file. It should be noticed that a similar methodology can be adopted to automatically generate variants of circuits, that is, various ATP master files. The module GenNet is responsible for this task. It was designed to do simple tasks such as changing the size of a capacitor-bank, while being “smart” enough to generate realistic electric circuit data.

The next section illustrates the use of the framework through a simple example.

4.0 SIMULATION RESULTS

The framework was used to generate fault events based on the ATPDraw circuit presented in Figure 4. This circuit is based on the Eletronorte Tramoeste system. To keep the fault identification experiment simple, it was assumed the same duration (indicated by Tmax in ATP) for all simulations. In this case, conventional (or *static*) classifiers can be directly used, as previously discussed, but the length L of the input vectors would be equal to the number N of samples of a sequence, which is prohibitively large in this case. Hence, windowing was used to generate Weka ARFF files, following the procedure adopted in [7].

AmazonTP was used to generate 12,000 examples (z, y) of 7 types of faults (AT, BT, CT, ABT,ACT, BCT, ABCT). Hence, the number of distinct classes y is 8, counting with the no-fault class. Figure 5 shows a zoom of waveforms at the moment of a BT-fault. The faults were generated considering they could happen at any position along the lines represented by the three ``Z-T line'' elements in Figure 4 (C1 shows up on top of them). The value of the fault resistance to the ground was draw from a uniform pdf $U(0, 0.2)$ with support from 0 to 0.2. The begin and duration of the fault were draw from $U(0, 0.4)$ and $G(0, 0.07)$, respectively, where $G(\mu,\sigma)$ represents a Gaussian pdf.

The waveforms generated by the ATP simulations had a sampling period equal to 0.5 microseconds (delta T = 5E-5 in ATP). The signals were decimated in order to create versions with sampling frequency equal to f_s = 10 kHz, a relatively low value (in order to test the algorithms under this condition). From each of the three phases, 5 samples (window length) of current and voltage at the node identified by TR230 in Figure 4 were collected, resulting in an input vector of length L=30. The window shift was made equal to 5 samples (no window overlap). The heuristic used for labeling was the following: the intersection between the time interval of the fault and the given window was computed, and such window would be labeled as no-fault in case the intersection is less than 50%, or as a fault (of the given type) otherwise.

The whole procedure led to two ARFF files, one for training and another for testing (disjoints sets). After that, it was a (relatively) simple matter of choosing Weka classifiers to be compared. Table 1 shows some of the results in terms of misclassification error on the test set. The entry ANN corresponds to a multilayer perceptron trained with backpropagation and having 19 neurons in the hidden layer. A SVM with a Gaussian kernel was used, and its model selection was conducted through cross-validation. It can be seen that the J4.8 decision tree [12], which is the Weka implementation of Quinlan's C.45 algorithm, outperformed the other methods, but more simulations are needed in order to establish solid conclusions.

Table 1- Classification results for the Tramoeste system using several Weka learning algorithms.

Classifier	Error (%)
Naïve Bayes	14.3
Decision tree (J4.8)	1.5
ANN	6.4
SVM	13.1

5.0 CONCLUSIONS

With the evolution of simulators such as ATP, which are able to provide reliable results, the artificially generated signals can effectively help to guide the research towards algorithms that are effective in real-world situations. The proposed framework makes much easier to compare and evaluate novel techniques. Besides, the software

is in public domain and the labeled datasets can be made freely available, which is an incentive to reproducing results and facilitate scientific collaboration.

There are several improvements that need to be made and both AmazonTP and AmazonMiner should be considered ongoing work. One of the improvements is the support for the PL4 (ATP) binary file format. Currently, AmazonTP asks ATP to generate ASCII files, and can then read them in. However, the files are bigger than necessary and I/O operations become slower. The PL4 format is not in public domain but the managers of the ATP software have been contacted in order to provide the documentation and grant permission to use this format.

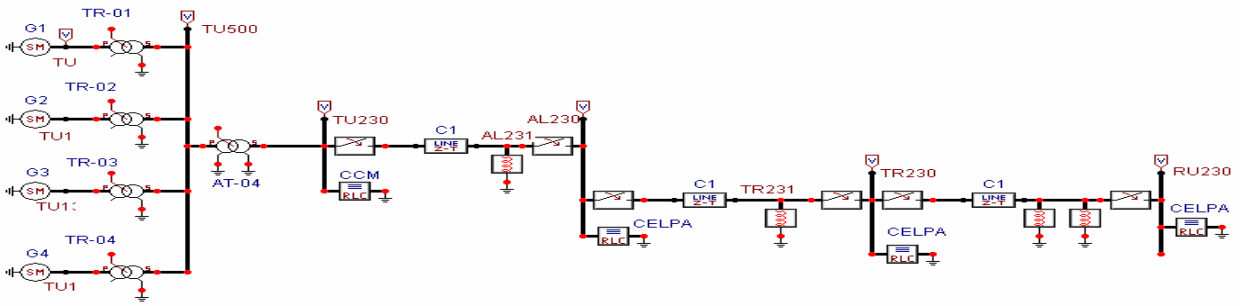

Figure 4– The Eletronorte's Tramoeste system, which was used as the master ATP file for the illustrative example.[iii]

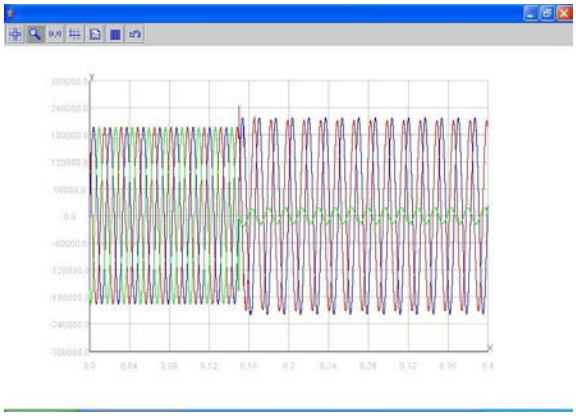

Figure 5- Snapshot of a plot generated by AmazonMiner: zoom of waveforms at the moment of a BT-fault generated by AmazonTP.

REFERENCES

[1] Keogh, E. and Kasetty, S.. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In the 8th ACM SIGKDD., 2002, Canada. pp 102-111.

[2] Santoso, S.; Lamoree, J.D. Power. Power quality data analysis: from raw data to knowledge using knowledge discovery approach. Engineering Society Summer Meeting, 2000. IEEE, Volume: 1, 2000. Page(s): 172 -177 vol. 1.

[3] Santoso, S.; Lamoree, J.D.; Bingham, R.P. AnswerModule: autonomous expert systems for turning raw PQ measurements into answers. International Conference on Harmonics and Quality of Power, vol. 2, pp. 499 –503, 2000.

[4] M. Kezunovic, T. Popovic, D. Sevcik, H. DoCarmo, Transient Testing of Protection Relays: Results, Methodology and Tools, IPST, 2003.

[5] S. Vasilic, Fuzzy Neural Network Pattern Recognition Algorithm for Classification of the Events in Power System Networks, Ph.D. Thesis, May 2004, Texas A&M University

[6] P. da Silveira. Identificação e localização de faltas utilizando análise por decomposição wavelet para relés de linhas de transmissão. Tese de doutorado. Agosto de 2001. UFSC.

[7] B. A. de Souza et al. Classificação de faltas via redes neurais artificiais. V SBQEE, Aracaju, Brasil. pp. 163-8.

[8] O. Delmont Filho et al. Utilização da transformada wavelet e RNAs para caracterização de distúrbios na qualidade da energia. V SBQEE, Aracaju, Brasil. pp. 381-6.

[9] L. Soares e H. de Oliveira. Wavelets na detecção, classificação e localização de faltas em linhas de transmissão. V SBQEE, Aracaju, Brasil. pp. 405-10.

[10] A.M. Gole and A. Daneshpooy. Towards Open Systems: A PSCAD/EMTDC to MATLAB Interface. IPST, 1997, Pp. 145-149.

[11] L.Rabiner and B.H.Juang. Fundamentals of speech recognition. Prentice Hall, 1993.

[12] Ian H. Witten and Eibe Frank: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 2nd edition, 2005

[iii] The authors would like to thank Eletronorte for providing access to this information.