International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:6, 2010

Modelling Silica Optical Fibre Reliability: A Software Application

I. Severin[1], M. Caramihai[1], R. El Abdi[2], M. Poulain[2] and A. Avadanii[1]

Abstract**—**In order to assess optical fiber reliability in different environmental and stress conditions series of testing are performed simulating overlapping of chemical and mechanical controlled varying factors. Each series of testing may be compared using statistical processing: i.e. Weibull plots. Due to the numerous data to treat, a software application has appeared useful to interpret selected series of experiments in function of envisaged factors. The current paper presents a software application used in the storage, modelling and interpretation of experimental data gathered from optical fibre testing. The present paper strictly deals with the software part of the project (regarding the modelling, storage and processing of user supplied data).

Keywords**—**Optical fibres, Computer aided analysis, Data models, Data processing, Graphical user interfaces.

I. INTRODUCTION

FIBER aging has been the subject of numerous studies leading to theoretical models for lifetime assessment. While ground observations do not contradict these predictions, the accuracy of the models is questionable due to the complexity of the aging mechanism. In this respect, experiments implemented on a long time scale are likely to bring new information. The better understanding of the factors ruling aging and reliability of optical fibers should lead not only to scientific advances, but also to economical spin-offs.

The technology evolution and the research for low cost optical fiber solutions lead to use new fibers and new components. Thus, polymeric fibers are being considered for the local distribution, while Bragg grating fiber components are now largely used in optical amplifiers. However, the reliability of these new components has still to be evaluated.

In practice, optical fiber aging depends on various factors that may decrease effective fiber strength: residual applied stress, temperature and water. It is assumed that surface flaws are enlarged, consequently promote crack growth. Maximum water activity is in aqueous solutions and it is expressed by the relative humidity (RH) in current atmosphere.

Various theoretical models are applied for mechanical characterization of optical fibers [1, 2], but the most common one is based on Weibull's statistics.

The Weibull law expresses the failure probability ***F*** of a fiber with a length ***L*** subjected to an applied stress:

$$Ln\left[\frac{1}{L}\left\{Ln[\frac{1}{1-F}]\right\}\right] = m\left[Ln(\sigma) - Ln(\sigma_o)\right] \qquad (1)$$

where ***m*** is a size parameter and σ_o is a scale parameter.

The evolution of $Ln\left[\frac{1}{L}\left\{Ln[\frac{1}{1-F}]\right\}\right]$ in function of *Ln (σ)* is known as the Weibull plot.

The values of ***m*** and σ_o are calculated from the slope of the curve and the intersection with the stress axis. The ***m*** parameter characterizes the defect size dispersion [2]. A high ***m*** value indicates that the distribution of the defect size is homogeneous while a low ***m*** value means that surface defects are varying in size. When the curve appears as a broken line with two distinct slopes – one small for low stress and the second one large, respectively – one has assumed two different families of defects, the first one corresponding to large extrinsic defects, and the second one relating to intrinsic flaws. Other plots encompass several straight lines relating to different groups of defects. The failure probability ***F*** is calculated from the relation:

$$F_i = \frac{i - 0.5}{N} \qquad (2)$$

where ***i*** represents the rank of the measurement and ***N*** the total number of values. The σ_o parameter represents the stress corresponding to the fiber cumulative fracture probability ***F*** of is 50%.

Comparing the Weibull plots traced for different testing conditions of optical fibers simulating harsh environmental conditions allow identification of potential damage responsible factors and understanding of reliability behaviour. That's why, a software application has been developed in order to trace effectively the Weibull plots registered for different aging schemes and to inter-compare the results.

The goal of the experiment is to analyze the behavior of various types of commercial optical fibers when subjected to different stress tests. In order to study this behavior we resorted to a statistical analysis based on the study of the fracture behavior of the optical fiber depending on stress and test conditions. Correlation between fiber tension and the probability of crack propagation for controlled conditions is conducted by our software, which is referred to as PDEFO. This correlation is represented in the form of curves called Weibull plots.

The purpose of the application is to allow the introduction, validation and sorting of data measured in the experiment (in this case it is the stress in the optical fiber in each test run), the obtainment of the probability P_k related to each test run, then draw the graph that has as its axes the two values (adapted for

easy interpretation of the graph). The application must also allow grouping of up to 50 experimental data sets (for each test) and generate graphs for up to 12 simultaneous data sets.

Other features that the applications offers in order to facilitate the input and manipulation of data:

- keyboard shortcuts for the most frequently used features;
- multiple/inverted selections;
- importing/exporting from/to Microsoft Excel documents (.csv);
- ergonomic design.

II. SOFTWARE TECHNIQUES USED

Firstly, we chose to develop a stand-alone application, which does not require other programs on the current machine (e.g. the application does not require that Microsoft Office be installed on the current PC).

In order to do this and save time on the GUI development we turned to Microsoft's .NET Framework (hence the application requires .NET Framework 4.0, which is still an acceptable compromise from our point of view since it is free software, therefore there are no licensing issues).

The programming language chosen for the backend of the application is C# because, for an application of such lengths, automatic memory management allows us to write code quickly and easily at the expense of a slight drop in performance [3, 4, 5].

One of the most important factors in choosing the programming language for the backend was the existence of a free, preferably open-source, library that allows easy plotting of graphs. The development of our own solution for charting graphs would have taken too long and would have made us miss our deadline.

Another argument that weighed heavily in our choosing the C# programming language was the existence of extensive online support for C#: from the most commonly encountered issues to the large number of users that can answer specific questions or help with solving certain errors encountered during the development of the application.

Also, readily available classes for serialization and complex data types (such as DataSets, ObservableCollections, Hashtables, Dictionaries etc.) were a big plus. These features helped us implement data saving, importing/exporting from/to Microsoft Excel files with ease, as well as optimize memory usage and response times.

Last but not least, being able to guarantee a consistent experience on any Windows platform by using the .NET framework was a welcome addition.

The Graphical User Interface (GUI) was made using Windows Presentation Foundation (WPF), since the installation of an additional library was not required.

Other factors we took into consideration when choosing WPF were [6, 7]:

- WPF is a relatively new technology from Microsoft which allows for easy creation of graphical interfaces, at least as fast as the Windows Forms;
- WPF has a mechanism called binding, which makes an automatic correlation between user inputted values in the GUI and the mathematical model implemented in the backend; this mechanism allows automatic validation of data, its processing, automatic posting etc;
- Thanks to the way WPF is designed, it allows for complex calculations related to the GUI to be performed by the processor of the video card, sparing the CPU of these calculations and thus ensuring a more rapid functioning of the program (although this mechanism was taken into account when we chose WPF for the GUI implementation, it ended up not being used in the final implementation);
- The possibility to rewrite the templates for all of the basic controls (from statusbar to menus) - the present application contains mostly modified controls in order for it to have the desired look;
- Code for WPF is written in XAML, an XML-based language defined by Microsoft, which made it intuitive and easy to learn;
- Because we chose to edit the code in Visual Studio, its IntelliSense feature has helped very much with the development, decreasing the time required to write the code.

Due to the fact that the application has a pronounced mathematical character, we chose to start off by developing a mathematical model of an experiment and implement it in C #, planning to make the changes required for the communication with the GUI later.

The backend programming consists of mainly two C# files (Experiment.cs and ExperimentList.cs), each of them implementing a class with the same name.

ExperimentList

The ExperimentList class implements a collection (ObservableCollection) of Experiment objects (instances of the Experiment class we will treat later on) and defines methods for their management [8, 9, 10]

This class is only instantiated once (since one session can only contain one list of experiments) and it is mainly used for communication between the graphical interface and the backend (data set deletion, document saving etc).

Experiment

This class is the application core. Since the rest of the program consists of mainly GUI definitions and handlers, we can say that the Experiment class implements the low-level functions that execute at the atomic level.

The Experiment class implements:

- variables used for data storage (like fiber diameter, temperature, duration etc.);
- variables used for enabling/disabling each of the variables described above;
- a *static* structure of Dictionary type for P_k values' storage (this way, every set of N P_k values is only stored once in the memory and the set is accessible by the very key N); this approach assures minimum

memory usage and minimum process time spent on computing these values;

- a structure of type ObservableCollection which contains objects that store the data the user inputs (the stress readings) during the experiment; the object that store these values are actually instances of another class called DoubleObject, used to automatically communicate with the GUI via binding-type mechanisms that are WPF-specific;
- a complicated ObservableCollection structure to store the computed data (C, ln(C), P_k etc);
- a constructor without parameters;
- a constructor that accepts a CSV line parameter and parses the data, easing the creation of a new Experiment object from a CSV file;
- a static constructor which allocates the memory for the static variables;
- properties that make the variables available for reading and/or writing depending on the specific needs for each variable;
- public methods for experiment data alteration, deletion, data recalculation or data export;
- the implementation of theInotifyPropertyChanged property, needed for using this class instances within the GUI directly (by binding to them);
- the class also implements the ISerializable interface, which makes saving or reading the data to/from a file almost trivial;
- the memory used by the instances of this class represents quite a big percentage of the total memory the application uses, therefore optimizing this class in terms of memory usage was mandatory in order to keep it under a decent limit.

The front-end programming (the graphical user interface) represents quite a large part of the code itself and it is located in the UI directory [6, 7]. Although the software design pattern we chose is not modular (like MVC for example), we tried to separate the backend programming (mainly consisting of mathematical calculations and data manipulation) from the GUI, which is usually not difficult, but complex because of its size.

The *.xaml* files located in the UI directory, along with the *.cs* files associated, implement the interface and the specific functions (windows, controls, handlers, security mechanism like preventing exit if the file was changed etc.)

III. FUNCTIONALITY AND SCREENSHOTS

The main application window is presented below:

The four areas highlighted in the image are as follows:

1. Status bar – Indicates the number of selected experiment sets, the total number of experiments in the current session as well as the outcome of the last import/export command executed;

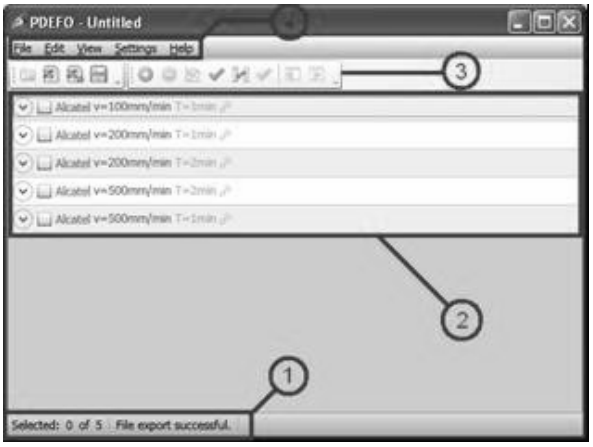

Fig. 1 The application main window

2. Experiment list – Contains all the experiments uniquely identified by their name and special testing conditions (if any are applicable). Within this list, every experiment can be expanded in order to display the data associated with it. Also, multiple experiments can be selected by left clicking on the checkbox or by right clicking them. To the right of each experiment there is a greyed out wrench icon which opens the experiment editing window.

Fig. 2 Multiple experiment selection

3. Toolbar – Contains shortcuts to the most used menu entries (open, import/export, save, add experiment etc.). These buttons are only available when the associated action makes sense.

4. Menu – Used to access every feature of the application. The menu entries are only available when the associated action makes sense.

Further below you can see the Add/Edit Experiment window (Fig. 3):

Fig. 3 Add/Edit Experiment Window

Each test condition in a particular experiment can be enabled or disabled via a checkbox called "Enabled". Once a property has been activated, it can also be changed. The only mandatory property is the breaking speed of the current test. The data is introduced at the bottom of the window (the values of the stressing fiber). If the program detects an error regarding the consistency of the data, the value is flagged as being wrong.

After inputting several sets of data, the graph containing the Weibull curves can be plotted for one or more of the sets. To do this, in the main window, we select the experiments for which we wish to plot the graph by either right-clicking or ticking their checkboxes, and then we call the Plot (Ctrl+F2) function. The result is shown below (in Fig. 4):

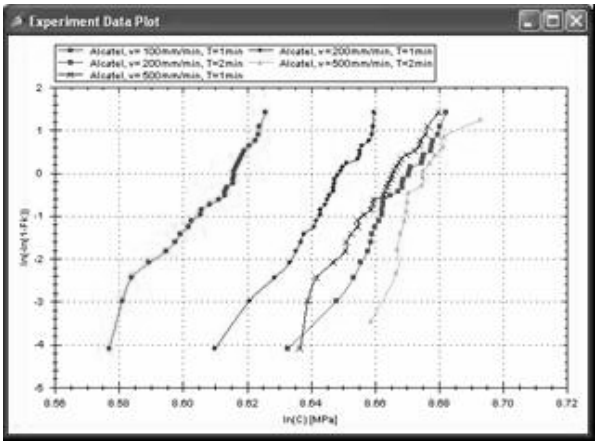

Fig. 4 Experiments data plot

To plot this graph we used the ZedGraph.dll open source library, which offers many options, such as:

- Possibility of zooming in / out, resetting the zoom;
- Automatic scaling of the axes;
- Resizing the graph when the parent window is resized;
- Possibility of printing the graph;
- Possibility of saving the graph in various formats

ACKNOWLEDGMENT

The following students were involved in the software application development: Alexandru Avădănii (Team leader), Iordache Florin (GUI programmer), Petrea Adrian (C# programmer).

REFERENCES

[1] J. Zarzycki, Les verres et l'etat vitreux, Ed. Masson, (1982)
[2] A. C. Wright, The structure of some simple amorphous network solid revisited, J. Non-Cryst. Solids, 129, p. 213 (1991)
[3] R. L. Mozzi, B. E. Warren, The structure of vitreous silica, J. Appl. Cryst., 2, p. 164-172 (1994)
[4] J. K. West, L. L. Hench, Silica fracture: part I, A ring contradiction model, J. Mat. Sci. 29, p. 3601-3606, (1994)
[5] S. H. Garofalini, D. M. Zirl, Onset of alkali adsorption on the vitreous silica surface, J. Vac. Sci. Tech. A6, p. 975-981, (1988)
[6] http://msdn.microsoft.com/en-us/default.aspx
[7] http://zedgraph.org/wiki/index.php?title=Main_Page
[8] http://www.codeproject.com/?cat=3
[9] http://bea.stollnitz.com/blog/
[10] http://www.codeplex.com/wpf