**SERBIAN JOURNAL OF ELECTRICAL ENGINEERING**
**Vol. 10, No. 1, February 2013, 185-198**

**UDK: 004.438 "JAVA"** **DOI: 10.2298/SJEE1301185R**

# Java Based Tool for Fault Detection Processing and Result Visualization

**Dušan Radivojević[1], Dražen Drašković[1], Zaharije Radivojević[1], Miloš Cvetanović[1]**

**Abstract:** This paper examines a possible application of Java technology for an implementation of a software tool for processing and visualization of input signals originating from a thermal power plant's coal-supply system. Performance of the tool is evaluated from the aspects of algorithms, architecture, and scalability. During the evaluation the performance indicators were CPU and memory load, while varied input signals were sampling rate and a level of user interaction. All measurements are performed on both real and synthetic load.

**Keywords:** Java tool, Fault detection, Data visualization, Performance analysis.

## 1 Introduction

The main reasons for use of Java technology for processing and visualization of signals are a wide range of available tools and free open source libraries which enable fast and easy application development. Another advantage is the ability to run the same application on different platforms without the need for adaptation. In addition, running an application under the Java virtual machine enables easy control of access rights to the actual resources that might be important in safety-critical systems. The negative effects of use of Java technology are mainly related to increased CPU and memory loads.

This paper examines the possibility of using a Java-based technology for the implementation of a tool for signal processing and visualization. The developed prototype, named PRODI, is a monitoring tool for a thermal power plant's coal-supply system. The tool, in real time, periodically check critical input signals and by using the built-in expert system decides whether there has been a failure or not, and also visualize input signals and calculated values. The tool is designed according to the configuration and input signals that correspond to the ones obtained at power plant "Nikola Tesla". The tool is used for testing purposes only and has never been deployed in the production environment. The goal of this paper is to illustrate empirically obtained the tool's performance

[1]School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11120 Belgrade, Serbia; E-mails: dusan.radivojevic.belgrade.serbia@gmail.com; drazen.draskovic@etf.rs; zaki@etf.rs; cmilos@etf.rs

results and to describe certain design decisions made during the development of the tool.

The paper has the following structure. Section two gives insight into background and motivation for the implemented fuzzy logic algorithms. The tool description and implementation details are given in Section three, while Section four examines the performance of implemented algorithms, used architecture, and expected scalability. Section five evaluates the obtained results, while section six concludes the paper.

## 2 Background and Motivation

In modern complex systems, especially safety-critical systems like thermal power plants, it is very important to incorporate reliable fault detection and isolation (FDI). Assuring early detection of faults has many benefits such as avoiding subsystem deterioration, maintaining performance and optimality or avoiding damage to main machinery. In addition to early detection, it is very important to provide for quick error isolation so that operators can take proper corrective action and restore the system to its normal operating point, avoiding unnecessary shutdowns.

The coal-supply subsystem is one of the most important subsystems in modern thermal power plants, responsible for adequate fuel delivery. Any failure to maintain its normal operating condition can lead to operating losses and production delays. The coal stored in hoppers falls onto the first and second conveyors called the feeders, which delivers the coal to the mill. The mill crushes the coal into fine dust, which is then added to preheated air. The resulting air-fuel mixture is blown into the burner. This process is monitored by means of the air-fuel mixture temperature. Since the temperature of the preheated air is well regulated, with small variations, the temperature of the air-fuel mixture is mainly defined by the amount of coal dust. If the amount of coal reaching the mill decreases (e.g., if a large chunk of coal blocks the supply to the feeder), the temperature of the air-fuel mixture increases because it mainly consists of preheated air (coal dust decreases the temperature of the mixture because its temperature is much lower than that of the preheated air). When the air-fuel mixture temperature exceeds a 250$^{o}$C, the burner has to be shut down. The main idea is to construct an FDI system able to detect any coal-shortage as early as possible, using the data from the process.

The vast number of FDI algorithms found in the literature can be divided into two groups: model-based algorithms and data-driven algorithms. Model-based algorithms assume the existence of a mathematical model of the system, which makes them very powerful. However, this assumption also represents the main drawback [1, 2], as model-based algorithm design needs to be insensitive to disturbances and include noise impact suppression mechanisms, while being robust to modeling errors with sufficient sensitivity to faults [1]. Modern power plants represent complex systems with large numbers of unknown inputs, rendering mathematical modeling very complicated or even impossible. Data-driven algorithms, on the other hand, are based on the statistical decision theory and can be further divided into two main groups: sequential algorithms and fixed size sampling (FSS) algorithms [3, 4]. Data-driven algorithms also have drawbacks. Sequential algorithms are well developed for stationary systems, but in variable structure systems like power plants, forming a decision strategy is a highly sophisticated problem. FSS algorithms are much easier to implement but their main drawback is the introduction of a large number of assumptions which they cannot readily satisfy in a real-life application [5, 6].

A two layer FDI system [7] for coal-shortage detection in a thermal power plant coal-supply unit is shown in Fig. 1. Layer 1 consists of a knowledge-based fault detection and isolation intelligent system (FDIIS) and an algorithm based on an FSS strategy, proposed [8], running in parallel. Outputs from these systems (residuals) are then passed on to Layer 2 of the intelligent system for evaluation and a final decision on whether a coal-shortage really occurred. As a result, a lower probability of miss and faster detection are accomplished. The motivation for this FDI structure lies in the knowledge that no perfect FDI algorithm exists and that reliable and in-time fault detection and isolation in complex systems can be achieved by combining different methods, and this combined FDI method offers several "opinions".

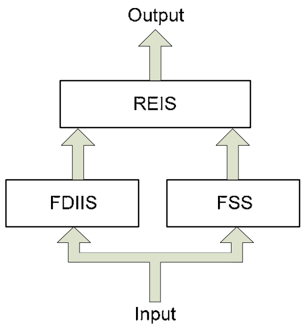

**Fig. 1 –** *Two-layer system. Layer 1 consists of an FDIIS running in parallel with an FSS algorithm. Layer 2 consists of an intelligent system which evaluates residuals (REIS), generated by Layer 1 algorithms.*

## 3 Tool Description

The software tool is implemented upon the previously presented theoretical background. The algorithms conform to the requirements of the Nikola Tesla Thermal Power Plant, Obrenovac, Serbia. The problem addressed concerns coal-shortage detection at the mills of this plant. The system comprises three parts: Data Acquisition, OPC Server and PRODI tool (Fig. 2).

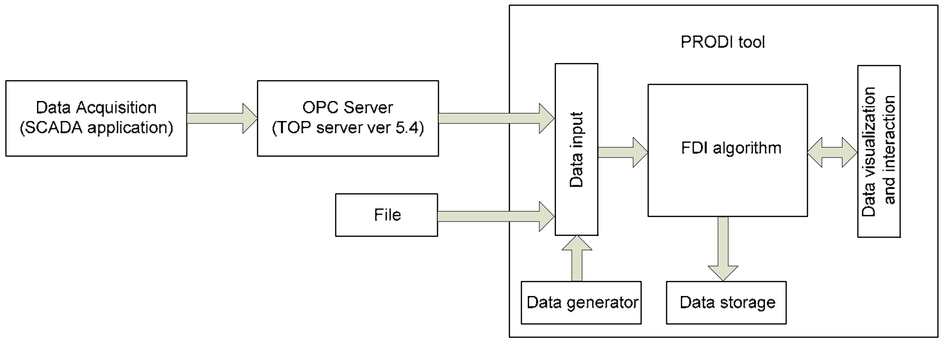
PRODI tool
Data Acquisition (SCADA application)
OPC Server (TOP server ver 5.4)
File
Data input
FDI algorithm
Data visualization and interaction
Data generator
Data storage

**Fig. 2** – *Structure of PRODI tool.*

The Data Acquisition represents the boiler of the thermal power plant. It is a SCADA application, including a schematic representation of the process and on-demand simulation of faults. It consists of modules for combustion chamber, six mills, control unit, turbine controller, fuel oil system, and superheater. Within modules, parameters like mill velocity, mill current, feeder torques, and others can be interactively changed.

The OPC (Object Linking and Embedding (OLE) for Process Control) ensures communication of real-time plant data between control devices from different manufacturers. OPC was designed to provide a common bridge for Windows based software applications and process control hardware.

The PRODI tool (Fig. 2) consists of five modules named: data generation, data input, FDI algorithm, data storage, and visualization. Data generation module produces data streams that statistically corresponds to the real measurements and enables PRODI tool to be used independently for simulation purposes. Data input module reads data streams from either OPC Server, Data generation module, or file, and converts them to a form suitable for the FDI algorithm.

The FDI algorithm module processes data according to algorithms described in the previous section. The algorithm is organized in two layers where the first layer comprises of intelligent system based on fuzzy logic [4] and system based on fixed size sampling [5]. The systems at the first layer run in parallel and forward their results to the second layer. The second layer contains an intelligent system based on fuzzy logic that decides on whether a system works correctly or a fault occurred [8, 9]. The processing is implemented as a real-time system meaning that the all calculations and visualization have to be performed within a sampling cycle. The system supports variable sampling cycle and the signals that could be sampled and processed encompass feeder moment, mill current, feeder and mill torque. All three algorithms were firstly developed using MATLAB environment and that implementation was used as a referent for testing purposes. Afterward, a second implementation using Java programming and available fuzzy logic libraries was developed. The Java implementation was further optimized, bringing up the third implementation, having in mind fixed size of the sampling buffer needed for the algorithm and iterative nature of processing.

Data storage module collects both input and output data used by the FDI algorithm module and stores them in external files or databases in such a way that enables reconstruction of complete history or any part of it. Stored data is signed using private-public key encryption for the purpose of non-repudiation and data integrity preservation. Stored data can be also used for simulation and algorithm verification purposes.

Data visualization module displays input signals, variables and decision functions, and supports on-line parameter tuning. Each graphic (Fig. 3) supports manipulation of range size, provides an instant overview of signal levels, and displays the domain in which a supply system failure was recognized. Apart from the visual display of selected signals, additional options, including display parameter tuning, signal status change notification and help options are available.

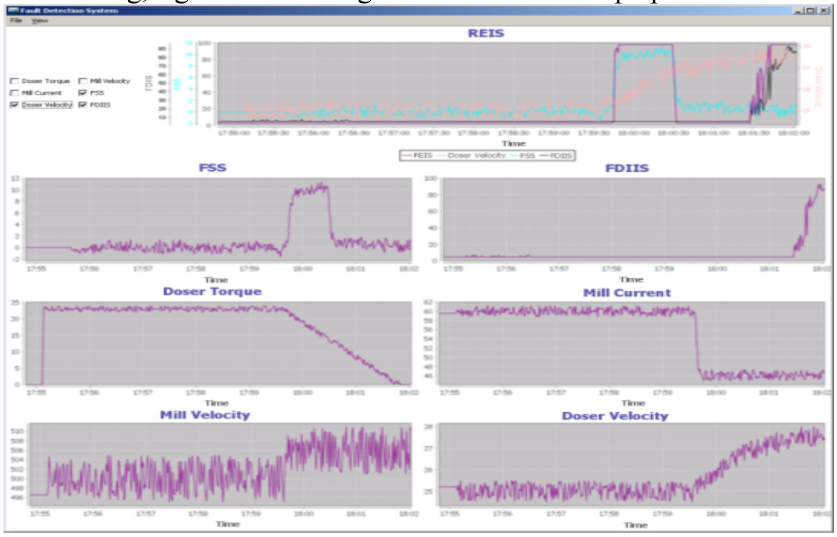

**Fig. 3 –** PRODI *tool: visualization of input signals and decision functions.*

The PRODI tool was developed using integrated development environment (IDE) Eclipse v3.6.1, with additional plug-in WindowBuilder v8.1 for user interface design and Test and Performance Tools Platform v4.7.1 for performance evaluation during development process. Development process relayed on using existing solutions, such as open source libraries, as much as possible. The JFuzzyLogic library was used as a support for fuzzy logic processing and some of the frequently used methods encompass fuzzification, defuzzification, and interpretation of Feature Code List (FCL) formatted text files. The JFreeChart library was used for data visualization using various types of charts, pies, graphs, and supporting objects (e.g. labels, markers, annotations). The JEasyOpc library was used for accessing the OPC server and retrieval of signal values sampled by the Data Acquisition part of the system.

## 4 Performance Analysis

Insight into performance of a tool enables quality and usability assessment of the tool and technologies used for its development. Besides, performance analysis can pinpoint the potential bottlenecks that might be resolved by changing or optimizing the source code.

Performance analysis and results presented in this paper are obtained on Intel Core i3 M350 (2,27GHz), with 3GB DDR3 physical RAM memory, 64-bit Windows 7 SP1 operating system and 32-bit Java virtual machine version 1.6.0_26. Tools used for performance and resource measurements are standard Windows monitoring tool Resource Monitor and Java profiler VisualVM version 1.3.2.

This section is organized in three subsections dealing with different aspects of performance analysis. The first subsection presents the results regarding the different implementations of the FDI algorithms used in the tool. The different approaches to the software architecture of tool are considered in the second subsection, while the third subsection deals with scalability of the tool.

### 4.1 Algorithms

Performances of the FDI algorithms have a great impact on the overall performances of the tool because they represent core of the signal processing. Obtained results illustrate differences among three implementations and used technologies.

The version developed using MATLAB was used as a referent implementation and was not considered for production because of the requirements to support true real-time environment. Two other implementations are developed using Java programming language. Even though Java supports real-time execution using specific virtual machines such as jRate, Apogee, or IBM WebSphere Real-Time, this paper considers performance results using conventional Java virtual machine. The first Java version is developed using existing open source libraries for fuzzy logic. The second Java version, which solely relies on standard Java libraries and specially developed methods, is optimized with regard to the problem addressed and domain knowledge.

The evaluation procedure measures the processing time from the moment of data entry into the FDI algorithm module to the moment of release of the corresponding output. The input data contains 7195 samples and the sampling rate is set at a value of one second. Execution time of the algorithm is calculated as average time spent per sample.

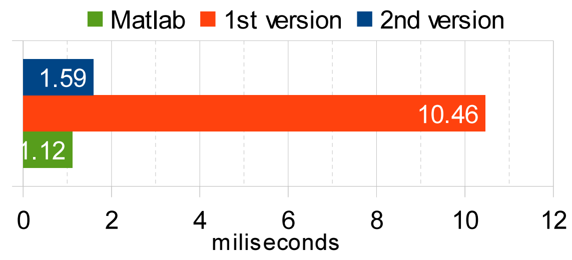

**Fig. 4 –** *Comparison of signal processing algorithm performance.*

The execution time, given in Fig. 4, shows that the referent implementation exhibits the best processing time coming from the fact that is executes using native mode. Even though the best in laboratory conditions MATLAB implementation was not considered for production because of the requirements to support true real-time environment. Comparison of implementations in Java shows that the processing time of the first version is increased by a factor of 6 compared to the second, optimized version, or 9 compared to the reference implementation.

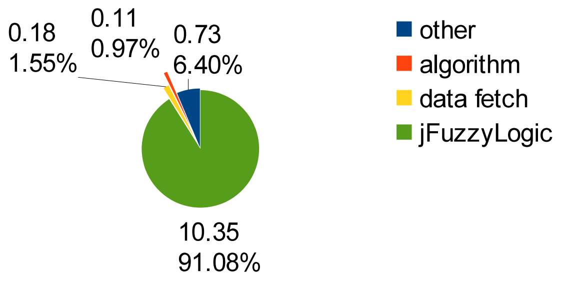

**Fig. 5 –** PRODI *tool execution time distribution in milliseconds* (*per sample*).

In order to understand the reasons for discrepancy between two Java versions the execution time distribution has to be analyzed. The percentages of time that the first version spends in the various parts of the code are unevenly distributed as shown in Fig. 5. The largest proportion of the time (91%) is taken up by fuzzy logic expert subsystems, or more precisely by fuzzy system processing methods.

In the JFuzzyLogic library used for the first version, the most demanding process during the implementation of the fuzzy logic controller is the implication process where the resulting membership function is computed in a pre-defined range, which is the standard approach in the implementation of a configurable controller. On the other hand, in the second version this is exactly where reference to a specific knowledge base and the nature of the membership functions are used. By computing important points only no time is lost on computing the entire range of the function. Since the range in the library is modeled for 1000 samples, and the resulting membership function is computed twice in both fuzzy logic subsystems (FDIIS and REIS), there are 4000 operations per sample, compared to the two operations required in the second version.

## 4.2 Architecture

In this subsection two versions implemented in Java are compared from the aspect of software architecture. The first version uses four threads while the second version uses two threads. The four-thread architecture relies on one thread for visualization and other three for data input, algorithm processing and data storage [10, 11]. The two-thread architecture displays results in the first thread, while data input, algorithm processing and data storage are in the second thread, as presented in Figs. 6 and 7.

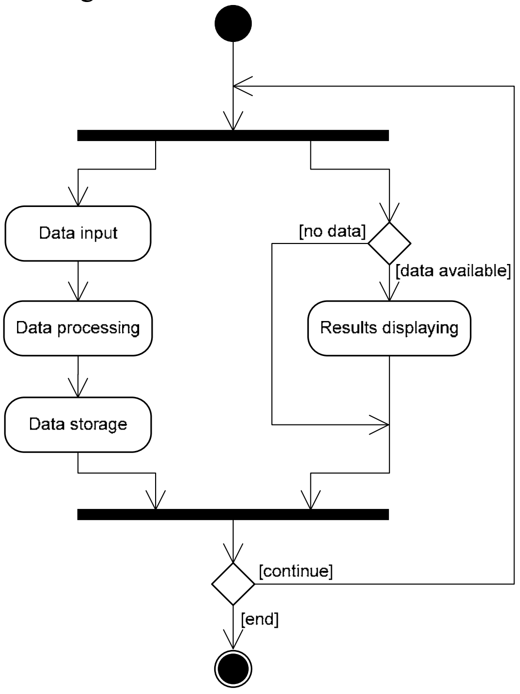

**Fig. 6** – *The two-thread architecture.*

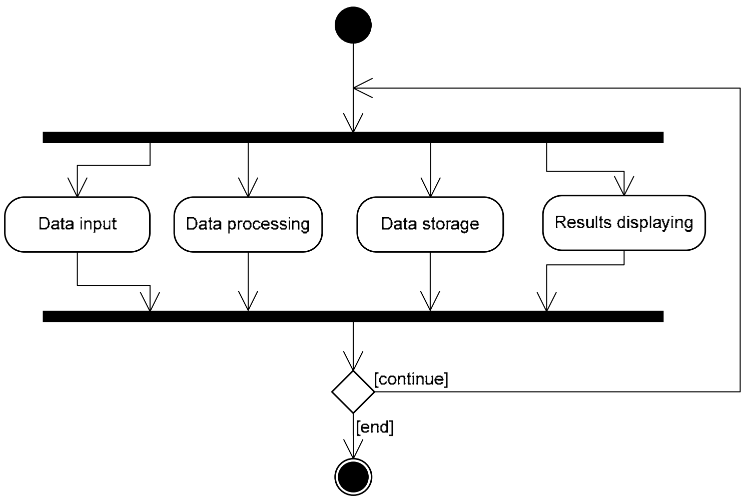

**Fig. 7 –** *The four-thread architecture.*

The processor load is measured with and without user interaction during the tool testing. Even though it is not an integral part of the processing algorithms, user interaction is considered because the time elapsed from the moment of acquisition of sample to the moment of displaying the result of processing may depend on the level of interaction and demanding graphical operations. It is extremely important to understand this behavior very well because if the processor load during user/application interaction is at its maximum, this means that it might not be possible to execute the processing algorithm at the planned rate. Fig. 7 shows average and maximum loads when there is no user interaction, as well as maximum loads with user interaction, for both implementation versions. The first version generates a slightly lower CPU load, and therefore better overall performance.

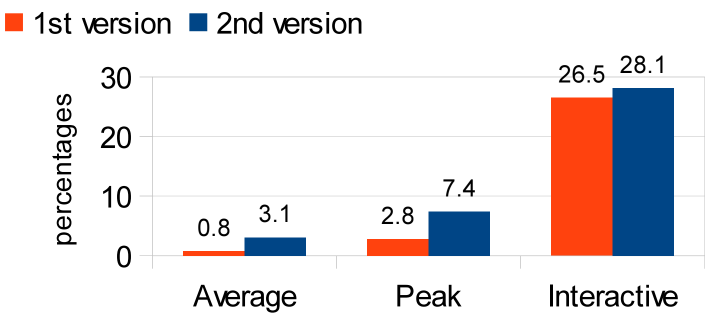

**Fig. 8 –** *Comparison of processor load depending on the implementation version.*

A characteristic of the maxima during user interaction, which is apparent in the processor load graphic (Fig. 8), is that they usually occur at the beginning of measurement and later rarely achieve that level. This type of behavior in the present case may be attributed to the time when the user interface, as a result of user action, initializes the graphical components not yet displayed.

The conducted tests also considered memory load with and without user interaction. The reasons for this are the same as when considering processor load. However, contrary to processor load, in addition to the current use of memory (active), attention in this case also needs to be paid to the memory no longer in use. The memory which is not in use is comprised of objects to which there are no references, but the garbage collector has not yet cleared it and, together with the active memory, it makes up the total memory. As shown in Fig. 9, the total memory may be more than double in size. To avoid this situation it is needed to provide explicit calls of the garbage collector in proper places in the source code. However, if explicit calls are too frequent, they may lead to unexpected processor load.

Based on the memory loads shown in Fig. 9, it is apparent that with user interaction the first implementation is more demanding than the second implementation and vice-versa when there is no interaction. This type of behavior may be explained by the fact that the second implementation, that uses only standard Java libraries, can handle better a configuration change.

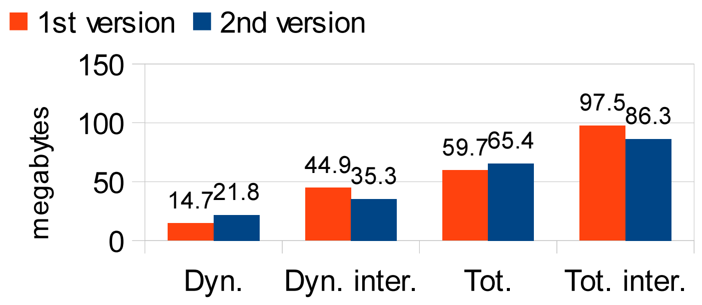

**Fig. 9** – *Comparison of memory load depending on the implementation version, where: Dyn. – Dynamic memory without interaction,*
*Dyn. inter. – Dynamic memory with interaction,*
*Tot. – Total memory without interaction,*
*Tot. inter. – Total memory with interaction.*

### 4.3 Scalability

Estimating scalability of the tool requires tests where input parameters are varied within their limits. Four out of five modules of PRODI tool predominantly depend upon two parameters, the sampling rate and a level of user interaction. Change in the sampling rate influence on a performances and functioning of data input, FDI algorithm, data storage, and visualization modules. Level of user interaction places more load on a visualization module that is identified as a critical one. Only module that does not depend on two mentioned parameters is data generation module, which is only used for testing purposes.

During experiments sampling rate is varied in range from 0.2 to 2 s, which corresponds to an expected human reaction time. Measurements are done with, and without user interaction, and this requires the user to go several times through all the options offered by the tool. Conditions during all measurements are identical or, more precisely, the tool is executed in the same environment and displays information as follows: one graphic with all series of normalized values, with the application window always set to the same size (1366×768) and the same set of applications active in the background. All measurements begin one minute after initiation of the tool and last for another five minutes.

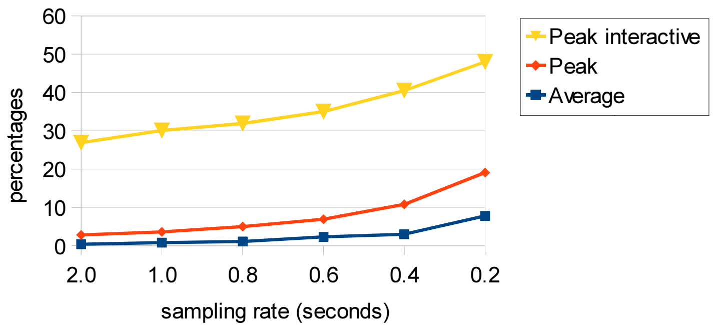

**Fig. 10 –** *Process load at for different sampling rates*.

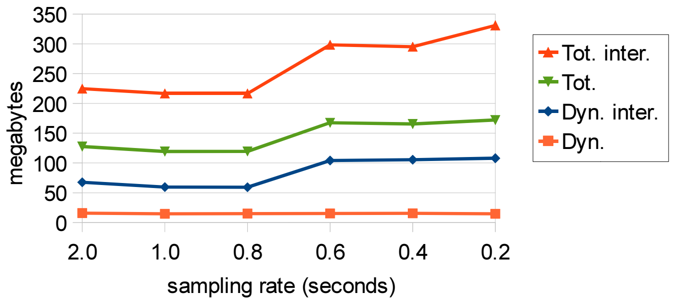

**Fig. 11 –** *Memory load for different sampling rates, where:*
*Dyn. – Dynamic memory without interaction,*
*Dyn. inter. – Dynamic memory with interaction,*
*Tot. – Total memory without interaction,*
*Tot. inter. – Total memory with interaction*.

Figs. 10 and 11 show processor loads and (peak) memory loads. Similar to previous measurements, the highest processor load at interaction occurs only once, during the first user action when graphical components are initialized. In some cases the usage of the dynamic memory is increased. Unexpected increase may be a significant problem and the cause lies in the Java virtual machine and its inner structure. Delay in the deallocation of the dynamic memory may lead to higher processor load due to the fact that allocating new memory space requires more processing power when memory is congested.

## 5 Results Evaluation

Special attention during the implementation of PRODI tool was dedicated to the optimization of source code in a way that would not affect its readability. Beside signal processing algorithm adaptation described in the previous section, steps taken regarding optimization are: avoiding instantiating temporary objects in dynamic memory, task parallelization where possible, and reimplementation of charts drawing library in a part regarding data storing.

The term temporary object relates to an object that has a relatively short life-cycle after which the object is discarded and in case of Java technology, left to garbage collector [12]. Garbage collector removes the object and frees the dynamic memory space that was occupied by the object. Activating garbage collector too frequently increases CPU load [13]. Activating garbage collector too rarely may increase memory load and also CPU load in cases when there is too much objects to be removed. Ways of dealing with temporally objects implemented in PRODI tool are usage of primitive types instead of objects, where possible, and "reusing" of temporary objects [14].

The usage of primitive types such as integers, floating point numbers, or logical values instead of classes decreases memory load and decreases CPU load by reducing garbage collector unpredictable executions [15]. Downside of primitive types usage is decreasing code readability by deviating from object oriented paradigm. Moreover, it is not always possible to use primitive types when classes are expected (e.g. library interfaces require object types).

Reusing temporary objects means keeping objects even after no longer needed with an idea to avoid creation of new objects of the same class. Reusable objects are usually implemented as a pool of objects whose size is determined upon estimated number of used and needed objects. Examples of objects that are implemented as reusable in the case of PRODI tool are objects for keeping values of signals or shared objects used for synchronization between working thread and graphical user interface thread.

Four threaded architecture shows how effective parallelization can be, and also how that allows easy extension of the tool's functionalities. Parallelization is obtained using multiple modules that are statically connected during initialization of the tool. Among modules there is one module that represents core of the tool and it interacts with the other modules, executing in separated threads, which are responsible for conducting specific tasks. The core module depending on a user activity starts appropriate modules. All modules are in a ready state after initialization of the tool, and can be started concurrently. Only core module is aware of existence of other modules. Such architecture allows easy extension of the system functionalities because adding a new module requires changes only in the core module. Suggested static linking of modules with the core module helps eliminating possible synchronization issues that could arise in architectures where dynamic, run-time, linking of modules is used.

Reimplementation of charts drawing library in a part regarding data storing is also conducted by using primitive types instead of classes. Floating point type is used for storing signal values, while integer type is used for storing absolute time shown on a chart. The library requires that both signal values and time values, are kept in form of data series. Instead of existing data series as dynamic structures, reimplementation introduced solution based on ring buffers. The ring buffers are implemented as static arrays whose sizes are based on expected number of samples for the observation period and are reallocated automatically when needed.

## 6 Conclusion

This paper presents an implementation of a software tool for monitoring of thermal power plant's coal-supply system. The tool processes input signals and visualizes both input signals and calculated values. In the essence the tool executes two stage fault detection algorithm based on fuzzy logic for the purpose of determining the correctness of the coal-supply and transportation system. The tool was developed using the Java programming language and some open source libraries.

Performance of the tool is evaluated from the aspects of algorithms, architecture, and scalability. During the evaluation the performance indicators were CPU and memory load. Evaluation of the algorithms showed that use of domain-specific knowledge and minimal changes in the open-source libraries can enable significant speedup of critical parts of the tool. Evaluation of the architecture showed that four threaded architecture creates lower CPU load then two threaded counterpart, but may introduce additional memory load in the presence of user interaction. Evaluation of the scalability showed that Java based tools can be used for real-time applications but beside a signal sampling period a level of user interaction with the tool has to be considered as an important parameter that may affect the overall performance.

## 7 Acknowledgment

Work on this project was partially cofounded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia (III44009 and TR32047). The authors gratefully acknowledge the support.

## 8 References

[1] J. Gertler: Fault Detection and Diagnosis in Engineering Systems, Marcel Dekker, NY, USA, 1998.

[2] S.X. Ding: Model-based Fault Diagnosis Techniques: Design Schemes, Algorithms and Tools, Springer-Verlag, Berlin, Germany, 2008.

[3] M. Bassevile, I. Nikiforov: Detection of Abrupt Changes: Theory and Application, Prentice Hall, NY, USA, 1993.

[4] T.L. Lai: Sequential Multiple Hypothesis Testing and Efficient Fault Detection-isolation in Stochastic Systems, IEEE Transactions on Information Theory, Vol. 46, No. 2, March 2000, pp. 595 – 608.

[5] L. Fillatre, I. Nikiforov: Fixed Size Sample Strategy for Change Detection and Isolation of Non-orthogonal Faults, 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Barcelona, Spain, June 30 – 03 July 2009, pp. 283 – 288.

[6] I. Nikiforov: A Simple Recursive Algorithm for Diagnosis of Abrupt Changes in Random Signals, IEEE Transactions on Information Theory, Vol. 46, No. 7, Nov. 2000, pp. 2740 – 2746.

[7] V. Todorovic, P. Tadic, Z. Djurovic: Expert System for Fault Detection and Isolation of Coal-Shortage in Thermal Power Plants, Conference on Control and Fault Tolerant Systems, Nice, France, 06 – 08 Oct. 2010, pp. 666 – 671.

[8] P. Tadic, Z. Djurovic, G. Kvascev, V. Papic: Coal-shortage Detection in Power Plants by Means of a Fixed Size Sampling Strategy, IFAC Conference on Control Methodologies and Technology for Energy Efficiency, Vilamoura, Portugal, 29 – 31 March 2010.

[9] P. Tadic, M. Stanković, S. Stanković, Ž. Djurović: An Application of Decentralized Estimation in a Fault Detection Problem, Serbian Journal of Electrical Engineering, Vol. 6, No. 3, Dec. 2009, pp. 373 – 387.

[10] Z. Radivojević, M. Cvetanović, Z. Jovanović: Reengineering the SLEEP Simulator in a Concurrent and Distributed Programming Course, Computer Applications in Engineering Education, (Early View).

[11] Z. Radivojević, M. Cvetanović: Dizajn simulatora diskretnih događaja opšte namene, ETRAN Conference, 06 – 08 June 2006, Belgrade, Serbia, Vol. 3, pp. 146 – 149. (In Serbian).

[12] H. Inoue, D. Stefanovic, S. Forrest: On the Prediction of Java Object Lifetimes, IEEE Transactions on Computers, Vol. 55, No. 7, July 2006, pp. 880 – 892.

[13] A. Corsaro, D. Schmidt: The Design and Performance of Real-time Java Middleware, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 11, Nov. 2003, pp. 1155 – 1167.

[14] R. Krapf, L. Carro: Efficient Signal Processing in Embedded Java Systems, International Symposium on Circuits and Systems, Bangkok, Thailand, 25 – 28 May 2003, Vol. 4, pp. IV-61 – IV-64.

[15] Y. Chang, A. Wellings: Garbage Collection for Flexible Hard Real-time Systems, IEEE Transactions on Computers, Vol. 59, No. 8, Aug. 2010, pp. 1063 – 1075.