METROLOGY AND MEASUREMENT SYSTEMS
VOL. XV, NUMBER 2 (2008)

TOMASZ BARSZCZ

AGH University of Science and Technology
Department of Robotics and Mechatronics
Kraków, Poland
e-mail: tbarszcz@agh.edu.pl

PROPOSAL OF ARCHITECTURE OF DISTRIBUTED VIBRATION MONITORING SYSTEM WITH FLEXIBLE ARCHITECTURE AND REMOTE ACCESS SUPPORT

With advances in IT technologies even complex machines with several gears and bearings, running at varying rotational speed can be efficiently monitored with an algorithm consisting of a combination of signal envelope and synchronous resampling.

For implementation, robust advanced software development technologies are applied, based on an object-oriented approach, supported by formal methodologies, like UML. Another concept important for distributed systems is middleware, like RPC or DCOM.

The paper presents a proposal of the architecture of the distributed vibration monitoring and diagnostic system, based on the approaches mentioned above. Principal components are described and their functionality is discussed. Special interest was put on flexibility of architecture and optimization of remote access. XML was proposed as universal and efficient means of configuration storage. The structure of the monitored machine should also be stored in XML. A proposal of such a data structure is described for a gas compressor.

Another key component is the database server. It needs to handle multiple databases and multiple users, delivering fast and reliable data. Vibration signal handling can be more efficient when signals are processed on the server side. Efficiency is essential when remote access to vibration data is required and data link is limited. Such a mechanism, based on plug-in approach, is proposed and described.

Finally, the paper presents a case study which describes the installation of the system on a gas compressor. The measured data are presented after different processing algorithms. High resolution envelope order spectrum was able to detect the bearing failure.

Keywords: distributed system, vibration monitoring, configuration, database

1. INTRODUCTION

Fast development of machinery state assessment over the last decade leads to rising demand for more accurate monitoring and diagnostic systems for rotating machinery. Since in almost all cases such an assessment is based on dynamic state measurements, those systems are vibration monitoring and diagnostic systems (later in the paper referred to as VMDS). At the same time one can observe quick improvement of the processing capabilities (understood first of all as the CPU speed and memory and disk volumes), which causes a drop of costs of VMDS. On the other hand, it is now possible to implement even very sophisticated diagnostic algorithms in such systems and apply them for a very wide spectrum of machinery [1]. An universal algorithm for vibration monitoring of a varying-speed machine with gears and bearings was presented in [2]. This algorithm is shown in Fig. 1.

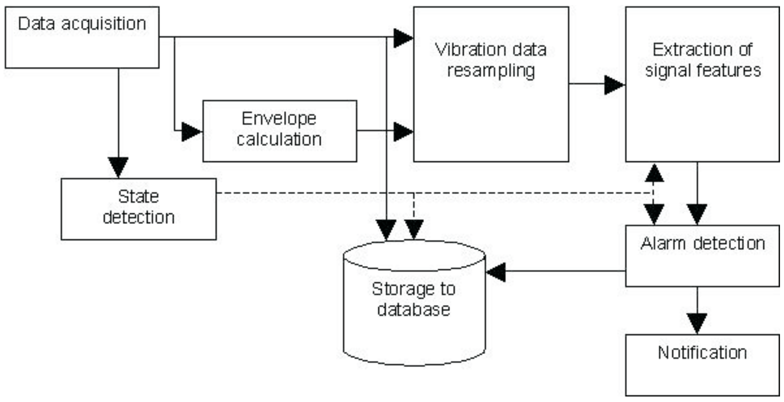

Fig. 1. Data procesing algorithm of the monitoring and diagnostic system [2].

The algorithm starts with data acquisition, both vibration and process values. For the detection of bearing faults, envelope calculation is performed [3]. For supervision of varying speed machines, both vibration signals and their envelopes are resampled. During resampling linear interpolation is used, since it provides an optimal balance between processing power and spectrum distortion [4]. Several signal features are then extracted from all signals (normal and envelope, both in the time domain and in the order domain). A discussion of selection of such features can be found in [3]. A special element of state assessment is state detection. Machine behavior depends strongly on process parameters like speed, load, temperatures and like. It should be possible to configure several states based on those values. If a state is detected, a set of limits for this state is activated and a separate alarm detection algorithm checks calculated features against limits.

The software implementing the algorithm should follow modern object-oriented methodologies and should fit a possibly wide spectrum of various machines. This goal can be achieved if the system architecture will be distributed and if appropriate software technologies will be used. In the next chapter modern object-oriented technologies will be presented. Then, distributed and flexible software architecture will be proposed. Since development and behavior of the system depends on the structure of its configuration and data storage solutions, they will be discussed in the following chapters. The paper is concluded with final remarks.

The proposed algorithm was implemented as the computer software in the VMDS and will be described in this paper. The work was performed within the EU co-funded project number WKP_1/1.4.4/1/205/6/6/569/2006, "The functional model of the distributed system for the vibration diagnostics". The test installation was commissioned on the natural gas compressor.

2. MODERN SOFTWARE DEVELOPMENT TECHNOLOGIES

Nowadays modeling of complex software systems uses a number of advanced software development ideas and technologies, most important of those is object methodology [5]. It provides means for achieving direct compatibility between the problem structure and tasks and a program code organization, being a very important feature from the developers point of view. A class concept plays a fundamental role in object methodologies. A class is a grouping of both properties and methods relevant to a well-separated part of a problem (an object or a process). Properties describe a state of an object/process, methods are essentially procedures which are allowed to execute on an object to perform certain actions. Both properties and methods can be privately or publicly accessible, thus allowing to control the rights to execute certain procedures. For modeling of complex systems the inheritance mechanism is used. It allows to extend/redefine the behavior of base class, through adding of new properties and methods, or writing virtual versions of existing methods.

In the software design process, the strong requirement for a modeling technology independent of the programming language became obvious [6]. This requirement is growing, as projects become bigger and the expected robustness level of systems increases. The role of such a technology is now served by UML (Unified Modeling Language). It is a meta-language, used for describing the structure of the whole software system, its parts called packages and their relations. The UML model is created using a graphical shell. It contains classes and connections representing relations between them. The UML model may consist of diagrams of the following types: static class diagram, collaboration/interaction, state, sequence and others. Most of them can be automatically translated to code in a chosen language (typically C++ or C#).

A most recent approach to the design and implementation of software systems is the component object model, closely connected with the concept of the middleware [7]. The idea of components is to embed in the binary code of a component the information on every interface element, which can be called by external software components. This information has a special format which can be automatically read and translated into text files, defining the interface to the component. This is an input information for programs which generate the code of "wrapper classes" in the chosen language. This allows for using the component by other components or applications. So far, a few standards of component object model are established: COM family from Microsoft Corp. which is native for the Windows operating system, and CORBA, which is more independent from an operating system and therefore more flexible. In many applications simpler technologies, like RPC, can be also used with less overhead [8].

Very important features of the object-oriented middleware approach were: encapsulation, hiding unnecessary variables from other objects, inheritance, which allowed to create hierarchies of objects of increasingly complicated features or different variants and polymorphism, where methods can be required by a base class, but are defined (or implemented) by a derived class at a later time. This innovation was followed by new design tools (based on UML technology), where objects could be defined, their interactions optimized and tested. This situation was quickly followed by another important feature – reusability, when objects became called components. Such components would be developed once and then used in a variety of projects. This brings benefits of decreased time and cost of development and higher reliability, as components are well tested. Currently, most of development environments come with a variety of ready-to-use components. A variety of components are available from independent software companies.

Another important change happened when computer networks became popular, which resulted in the development of distributed systems. In such systems, particular tasks of the monitoring and diagnostic systems were divided into several processes. These processes could be running on separate computers, what gave advantages of higher reliability, shorter development times and better scalability, to name just a few. In results of trends described above, a distributed VMDS consists of several processes, which in turn are built from components. Distributing system components also resulted in flexibility, as the system could be suited for a particular task by reconfiguration or modification of its components. An example of such flexibility will be presented in next chapters.

3. PROPOSAL OF A DISTRIBUTED SYSTEM ARCHITECTURE

The following chapter will present a proposal of the software architecture of a vibration monitoring and diagnostic system. First of all the design of the software architecture of the VMDS system should be flexible and open for future extensions. To achieve this, the system must be modular. It must be possible to modify the number of cooperating modules depending on the application and to add new modules when necessary. Figure 2 presents the proposal of such an architecture.

Each of the components in the picture above plays an important role in the system. Table 1 presents a short description of their functionality:

The **Data Hub** (DH) is the component which is the main data exchange hub in the system. Any data exchange between any components in the VMDS always needs the DH as the connection point. Such an approach allows to develop each module independently from the others, since it is enough to follow the software interface to be able to attach a new module to the VMDS. If we have a number of software modules

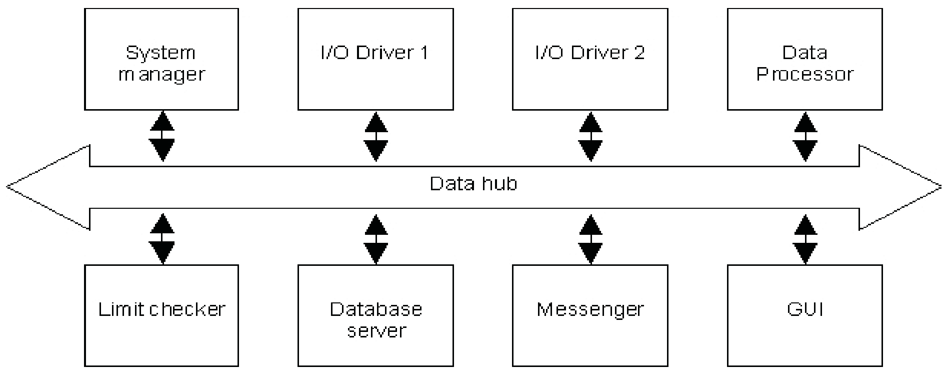

Fig. 2. Propesed software architecture of the distributed vubration monitoring and diagnostic system.

Table 1. Brief description of the distributed vibration monitoring and diagnostic system.

Data hub	Software component, setting standards for inter-module communication. It must be network-transparent, so the system can be distributed. There are several existing middleware standards on which the library can be built (e.g. RPC, CORBA, COM).
System manager	Main component, starting up all the other modules and supervising the whole system. Must have watchdog functionality, i.e. periodical checks whether all the modules are “alive”. The system manager is also responsible for management of the configuration.
I/O Driver(s)	Data input/ output modules. Depending on hardware, it can be a plug-in card, VME/ cPCI card, CAN or Ethernet module etc.; it detaches all hardware-specific functionality from the rest of the system
Data processor	Various data processing depends very much on application. In simplest cases this can be a statistic calculation, in cases more complicated e.g. system identification. There could e.g. exist the possibility of using procedures from MATLAB/ SIMULINK (through RTW package).
Limit checker	The component which detects if any limit was violated. If such a violation is detected, an event to other components should be sent.
Database server	The largest components implementing the database. This component will be described in a separate chapter.
Messenger	Sends messages when predefined condition occur. Messages can be: a TCP/IP packet, e-mail or SMS. This component is very useful in monitoring systems which are not supervised all the time by on-site operators.
Graphical User Interface	A separate application, presenting data in the system on a variety of plots. GUI can be also run on a remote machine. This component will be described in a separate chapter.

which share a common VMDS interface, it is possible to exchange a module, or even to change the structure of the whole system without changes in the software, but only in the configuration.

The basic function of the Data Hub is data exchange. The DH is not meant to store the history of channels, but only current value buffers. Various modules can share data by writing them in and from those buffers. Sometimes some components require short historical buffers, e.g. to calculate statistical values. Those buffers have only auxiliary meaning, to calculate the resulting values. The following basic data types are supported by the DH:

– scalar values (double precision floating point),
– binary values,
– events,
– dynamic values (i.e. vibration waveforms).

Scalar and binary values represent a single value from a sensor, like a single temperature value or pressure readout. It is the most popular type of data, existing also in e.g. DCS or SCADA systems. Events are data type describing anything that can happen in the system, typically not having any particular value, but having a defined moment in time. Good examples can be starting down the VMDS or detecting an alarm in one of channels. Events are used to inform selected modules about e.g. the completion of a task by a module. Events are also used to synchronize the whole system. Dynamic values occupy the largest space in memory, as they store waveforms of vibration signals. In the general case, however, such a value is defined as a buffer, which has no particular structure. Such a structure is known only to modules which are writing or reading such data. For the DH it is treated as a block of binary values of given length. For future modules this data type can be used for many other data, like e.g. images. In the current implementation it is used only for vibrations, where one dynamic data record holds the result of one sampling session. For example it can hold 25000 samples taken during one second from one channel.

Each data type stored in the Data Hub has three fields, listed below:

– time stamp
– status
– value

The time stamp holds the time of the readout. It is clear for scalar or binary values, but for dynamic data it is the beginning of sampling of the record. The status holds additional information about the data. Possible status values with descriptions are presented in Table 2.

A measurement error can take place when the read value is out of the measurement range or when a sensor failure is detected. In such a case, no further data processing makes sense. The N/A status is typically given in cases of more advanced calculations, when input data are correct but the resulting value cannot be calculated. One example of such a case is calculation of phase, when the amplitude is very close to zero. Then, the value of phase does not make sense and should not be evaluated.

Table 2. Possible status values in the Data Hub.

Status	Description
OK	Correct data, no limit violations
ALERT	Correct data, alert detected
ALARM	Correct data, alarm detected
ERR	Measurement error
N/A	Value cannot be evaluated

The **System Manager** (SM) component controls the operation of the whole system: it starts the whole system, i.e. launches processes of all defined components and executables and verifies the correctness of the startup process. After that is begins to check periodically the operation of all the modules. When a module does not respond, it will be deleted and restarted. In other words, this component performs the typical operation of a software watchdog. Another function of the SM is configuration management. It is the central place in the whole system, where all configuration information can be obtained. Since the structure of this information is very important, it is described in a separate chapter.

I/O Drivers (I/O) are modules which connect to the hardware and collect measurements. On the input side they deal with hardware-specific commands, typically provided by the hardware manufacturers. On the output side they respond to the Data Hub with the recently acquired data. Data are provided on a configured, timely basis, i.e. one complete data packet is sent to the Data Hub every defined period. Such a packet consists of samples from vibration channels and values from process channels (e.g. temperatures, pressures etc.). One has to add that the name of Driver can be misleading, and it should not be mixed with hardware drivers, provided by hardware manufacturers. Hardware drivers are libraries, which are intended to be used by applications using the hardware. In this case, this application is also called the Driver. This application is specific to the particular system.

Data Processor (DP) is the data analysis module, which creates new channels calculated from channels acquired by the Driver. There are three groups of calculations performed by the standard implementation of the DP:

– statistical calculations, like: mean, standard deviations,
– algebraic calculations, which are user-defined operations to calculate channels which are not directly available, like: "rpm = Freq * 60", "ΔT = T_deep – T_shallow" or similar; it is very helpful to define such calculations during the configuration phase, without changing the source code of the system,
– vibration signal parameters, like: peak-peak amplitude, rms value, amplitude of N^{th} harmonic; this calculation is typically the most CPU-demanding, as it requires FFT; often in order to monitor varying speed machines it is necessary to resample vibration signals to get from the time-to the order-domain; for bearing monitoring it

is necessary to obtain the envelope of the signal; all these operations are performed within the module.

After calculation, channel values are returned to the Data Hub for further processing. It is important that it is possible to run more than one Data Processor module. Apart of standard calculations listed above, using the DP approach is a simple way to prototype new diagnostic algorithms, unless they require very sophisticated data processing. If this is the case, it is possible to develop a new module dedicated for the new method. In such a new DP module (which could have a different name) more complicated algorithms, e.g. system identification would be implemented. Also, there could e.g. exist the possibility of using procedures from MATLAB/ SIMULINK (through the RTW package).

Limit Checker (LC) is the module which performs two basic operations: detection of the machine state and detection of alarms. Since the behavior of the machine often depends on the operating state more than on its technical state, it is very important to divide the operation of the machine into states and analyze data from each state. Good examples are turbosets, which have two distinct states: a transient and a steady-state. Another one are wind turbines, where some types operate on two levels of power most of the time. The method of state definition is in fact only one of possible methods of monitoring machines which change their operating point. Such states are typically defined by rotational speed and output power, but it is better not to restrict the definition of state to any specific set of parameters. The method of states is simple to understand and yields good results.

When states are defined, for each scalar channel value (including calculation results) one can define limits. Parameters of each limit are presented in Table 3.

Table 3. Limit parameters in the Limit Checker module.

Parameter	Description
Name	String uniquely describing the limit
State	Machine state in which the limit is to be active
Channel	Measurement channel which is evaluated
Value	Numerical value of the limit to which the value read from the channel will be compared
Operator	Comparison operator („¡” or „¿”)
Importance	Two levels are available: Alert and Alarm
Hysteresis	Protects against multiple detections, when a value crosses the limit
Detection delay	Protects against random peaks in the signal

All presented parameters are almost self-explanatory. Channel name is a link of limits to channels, and state is the link to states, if we want to describe limits in terms of relational databases. In this approach, limit is a many-to-many relation between states and channels.

The number of limits is quite high, as there are separate limit objects for each state and each channel. Each limit can have a severity level assigned, like: Alarm, Alert, Info. During system operation the Data Hub sends periodically vectors with current channel values to the Limit Checker module. In response the Limit Checker returns the same vector with a set status filled set. In parallel, when the start or end of a limit violation is detected, the Limit Checker sends the event object back to the Data Hub, from where events are forwarded to other modules.

Data Storage is the module which encapsulates the central database of the system. The Data Storage module is called periodically by the Data Hub with requests to write the recent data, which were tagged as "to be written". Apart of that, graphical user interface(s) asynchronously read various data, depending on users' requests. To achieve high performance the Data Storage module was implemented as the proprietary, high speed database.

The **Messenger** module sends messages to the remote users, when a predefined condition occurs. This component is very useful in monitoring systems which are not supervised all the time by operators on site. Messages can be sent through: TCP/IP packets, e-mail or SMS. It is possible to send messages in two modes:
- immediate, in case of critical alarms
- periodic, to send statistics of system operation and to ensure that it is running correctly

If notifications are to be sent through the e-mail, the module establishes the TCP/IP connection (unless there is a permanent one), then connects to the e-mail server and sends the message. The mail contains all important information describing the event. When there is no possibility to use mail, the module can send a SMS message via a specialized modem. Using SMS brings some limitations, since the size of a single SMS cannot exceed 160 characters. Also, to send it the modem must be equipped with a SIM card which must be activated and periodically paid.

4. STRUCTURE OF CONFIGURATION DATA

An important part of the project was to propose an universal and efficient method for configuration storage. Storage of configuration is performed by the System Manager module. The configuration of the system is divided into two branches: configuration of the measurement and configuration of the machine. In the beginning the first part will be shortly described, then the second one will be described in more detail. All configuration of the system is stored in XML format. The XML (eXtensible Markup Language) is an open standard, recommended for files portable among different platforms [9]. In its general form it can be applied for numerous tasks, such as data and configuration of various computer systems. Internal structures of XML suit the storage of graphs very well. Additionally, application of XML is efficient, because many ready-to-use software components for XML handling are available. Configuration of the VMDS consists of several XML files, describing parameters necessary for those modules. All modules store the configuration in tree structures, where e.g. I/O drivers store nodes describing measurement channels and their parameters, the Limit Checker stores node parameters with defined limits.

Apart of application modules, the configuration must describe the monitored machine also in XML format. Machines can be divided in a natural way into components like: AC motor, gearbox, pump etc.. Such a division is multi-layer, as often components are divided into sub-components. Another group of configuration information are measurement channels, as they are sources of data. They can be organized as a separate branch in the configuration tree, or they can be assigned to particular machine components. Since more than one component can be described by a single measurement channel, the optimal solution is to store all channels as a separate branch and to place references (or links) to these channels in parts of XML describing machine components. Based on this approach, configurations of several machines were prepared. Here is an example of such a configuration for the gas compressor.

a) File header
b) Root nodel
c) Installation – name, description, location
  - Flat list of measurement channels
  - Monitored compressor – name, description etc.
    - ■ Process channels node
      - • PV Channel 1 with its parameters
      - • PV Channel 2 ...
      - • ...
      - • PV Channel n ..
    - ■ Vibration (dynamic) channels node
      - • VIB Channel 1 with its parameters
        - ♦ List of all defined analysis with its parameters
      - • VIB Channel 2 with its parameters
        - ♦ List of all defined analysis with its parameters
      - • ...
      - • VIB Channel m with its parameters
        - ♦ List of all defined analysis with its parameters
  - Machine structure – list of components
  - Gas compresso
    - ■ AC motor
      - • List of assigned measurement channels, with assigned analysis
      - • ...
    - ■ Gearbox
      - • List of assigned measurement channels, with assigned analysis
      - • ...
    - ■ Compressor

- List of assigned measurement channels, with assigned analysis
- ...

▪ Bearings
- List of assigned measurement channels, with assigned analysis
- ...

The proposed configuration structure has several advantages. It is logically divided into parts which are directly read by appropriate modules. Apart from dedicated configuration tools, such a file can be also edited with an external XML editor, like e.g. XML Edit (freeware XML editor). Storage of configuration in a tree structure brings clarity of data presentation. In the user interface it is possible to display the tree structure, as presented above. This can serve for quick navigation if a channel should be opened. Also, if a problem is detected (e.g. limit violation or sensor failure), then such an information can be propagated from a source node in the tree to the root node. This gives a very efficient tool for the user to quickly find the reason of the problem.

5. DATABASE SERVER WITH PLUG-IN MECHANISMS

Data Storage is the module which encapsulates the central database of the system. It is possible to use various database types, e.g. a relational SQL database, flat file – based or a dedicated one. Regardless of the implementation, the database must store different types of data. The main data types are:

– **vibration data**: very large, carry most information, but quickly consuming disk space; there are several ways to limit the amount of data, e.g. data can be compressed (or only spectra are stored), data can be written rather seldom,
– **scalar data**: standard float type data, can be stored every few seconds/ minutes; such a form is used to store vibration signal parameters and the output from diagnostic modules,
– **event data**: smallest size, stored asynchronously when an event occurs (e.g. an alarm violation is detected).

The Data Storage module is called periodically by the Data Hub with requests to write the recent data, which were tagged as "to be written". Apart of that, graphical user interface(s) asynchronously read various data, depending on user(s)' requests. The structure of the Data Storage module is presented in Fig. 3.

The central point of the module is the database engine which manages all the databases in the system. There can be more than one database attached, as in the typical application the following databases are configured:

– alarm database – storing all alarms and data recorded when an alarm was detected,
– operational database – storing data periodically, based on time periods,
– reference database – storing data necessary for comparison with a known "good state"; data to this database are written only on user request, e.g. right after installation or modernization of the machine.

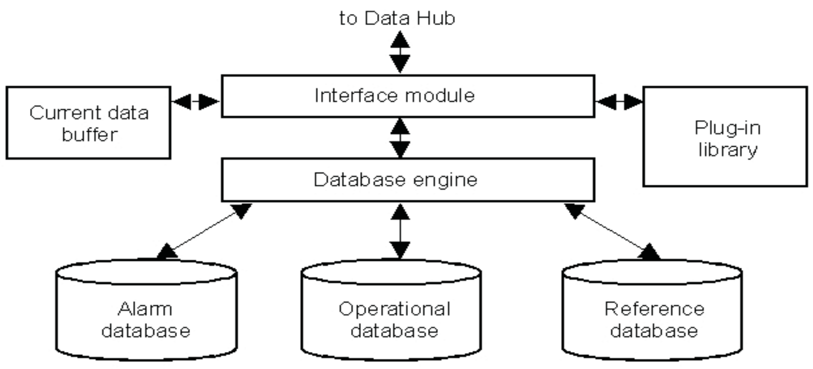

Fig. 3. Structure of the Data Storage module.

When several databases are attached, it is easily achievable to apply different circular buffer policies to these databases. The database server must implement a circular buffer, which will automatically overwrite the oldest data. Such overwriting must depend on the type of data. For example, operational data carry the smallest amount of diagnostic information and should be overwritten as the first ones. Alarm data are much more important and must be kept for longer periods. Reference data are very small, but very important and should never be overwritten.

Another important mechanism proposed for this database are **plugins**, which implement analysis of the data on the level of the database server. In plugins data are post-processed (if necessary) before they are returned to the requesting module (usually the User Interface). This technique is used predominantly for handling vibration data. When UI requests a spectrum, it is often very inefficient to send back the whole raw vibration data and then perform calculation of spectrum in the UI. The problem comes from the fact that raw vibration data can contain 1e6 samples (i.e. ca 2MB of data) and the UI only needs to display a maximum of 1000 lines of spectrum, as it is the horizontal size of most screens. The overhead in data transfer is in the range of 1000. Instead, the UI requests the spectrum with given input arguments (frequencies of first and last line, line resolution, ...) and calculations are performed at the server side by plug-in. This technique is crucial for **efficient access from remote locations** and will be presented in the case study.

For high performance the Data Storage module is a specialized, proprietary database. It does not use any standard SQL (Structured Query Language) engine. It implements indexes and basic queries, but several higher level features, typical for SQL, are not implemented. For example, transactions are not supported. It is a reasonable approach, as transactions are important in financial or management systems and have little application in data acquisition/ processing.

6. CASE STUDY – DETECTION OF A BEARING FAILURE IN A GAS COMPRESSOR

The described VMDS was installed on a 4 cylinder, natural gas compressor on an oil platform. The compressor was driven by an AC motor coupled directly to the compressor shaft. The compressor had several failures of rolling bearings, which led to forced outage and resulted in loss of production and profits. Bearing faults were ring defects, inner or outer. The role of the system was to perform high resolution online spectral analysis of the vibration signals in order to detect and track the frequency component generated by the faulty bearing. An additional requirement was the ability of efficient remote access, when the only communication medium was a GPRS connection with a practical data transfer speed of 20-30 kbps. Therefore, optimization of data transfer was essential.

The following plots will present an analysis of the compressor, when the fault in the rolling bearing was developing. The characteristic frequency of the outer ring fault was calculated as 8.14X (referenced to the fundamental shaft speed). This frequency was calculated based on measurements on-site, as no manufacturer data were available. The following formula was used to obtain the characteristic frequency [3]:

$$f = \frac{n}{2} f_r \left(1 - \frac{d}{D} \cos \alpha \right). \tag{1}$$

Such a procedure can have a significant error which will be discussed later in the paper.

Figure 4 presents the spectrum of the vibration signal from channel 8 whose sensor was mounted directly over the investigated bearing. The other vibration channels also showed the existence of this fault, but here it was the strongest. The spectrum has a resolution of 1 Hz and is strongly dominated by high harmonics of the fundamental shaft frequency. As the rotational speed was varying, the spectral lines showed some level of a "smearing effect" [10]. Also the influence of a structural response can be seen in the range 450 – 800 Hz. No bearing fault can be detected on the basis of this plot.

As the next step in the algorithm, the vibration signal is resampled, so that an equal number of samples per revolution is obtained. Next, the order spectrum is calculated (see Fig. 5). Since small variations of the rotational speed were cancelled, the harmonic lines are much sharper than in Fig. 4. Nevertheless, no sign of a bearing fault can be seen. There are small spectral lines around order 11-12, but there is not enough evidence to bind it to any kind of malfunction. Bearing faults are best visible in the envelope of the vibration signal (see algorithm from Chapter 1), therefore such an analysis was also implemented into the system. Figures 6 and 7 present the envelope spectrum and envelope order spectrum of the signal, respectively. Figure 6 already shows the existence of additional spectral lines around 100 Hz, which is equivalent to ca. order 8. In Figure 7 it can be seen that this component is slightly above the 8th

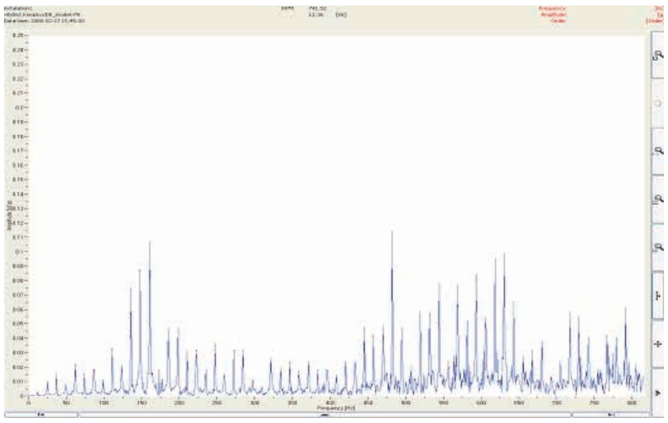

Fig. 4. Spectrum of the vibration signal on channel 8 (compressor bearing.

order. The zoomed spectrum, which is presented in Fig. 8, shows additional details. The frequency of the component is 8.34X and is slightly smeared, which leads to the conclusion that this line is caused by the outer ring ball pass. The frequency is slightly varying, since rolling bearing elements are not moving in a deterministic way but show some random behavior [11]. The frequency is slightly different than the one calculated from documentation, since there is a 2.4% frequency shift (8.34X measured vs. 8.14X calculated). This is most probably caused by bearing dimensions measurement errors. Moreover, other data sets showed some frequency shift which was probably caused by a change of the angle of the bearing.

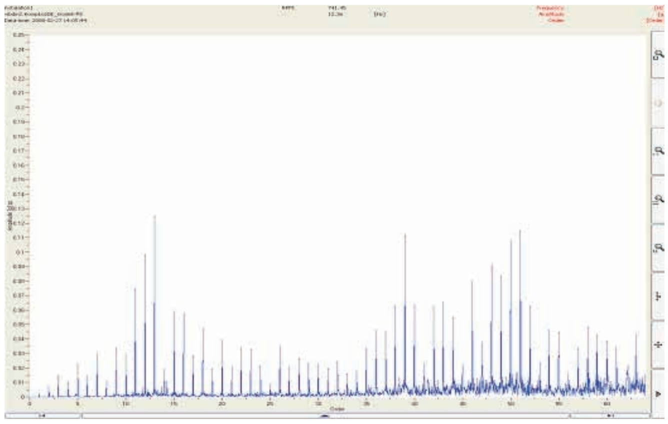

Fig. 5. Order spectrum of the vibration signal on channel 8.

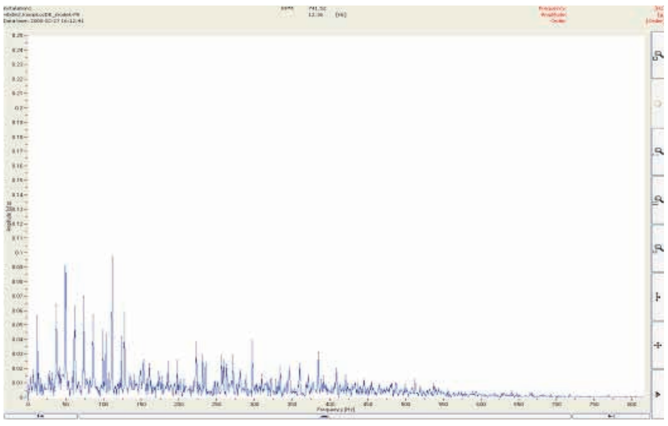
Fig. 6. Envelope spectrum of the vibration signal on channel 8.

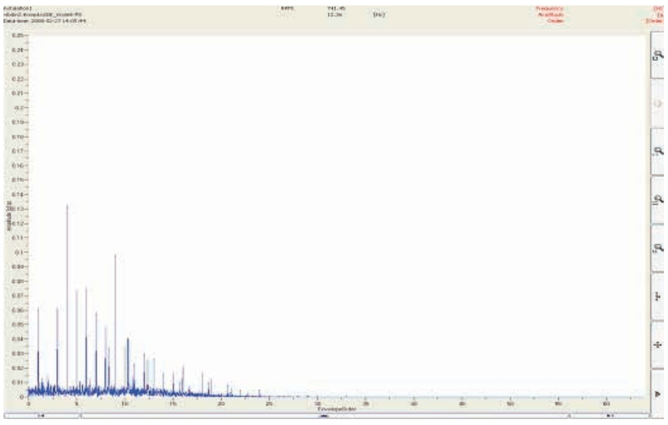
Fig. 7. Envelope order spectrum of the vibration signal on channel 8.

Figures 5-8 present the analysis of a single vibration signal. For a broader picture, showing the evolution of the malfunction over time, a trend plot is necessary. Figure 9 presents such a trend plot, showing the energy of the band in the envelope order spectrum, calculated around the 8.34X. The band was configured so that it does not include very strong harmonics of the fundamental frequency. The trend plot shows a stable vibration level assigned to the outer ring until around 30.12.2007, when the machine was stopped. After startup on 1.01.2008 machine showed an increased level of vibration in the monitored band. The level was varying, but in general was higher than in the period before stand-still. The machine was stopped on 9.01.2008 and the bearing was inspected. Inspection revealed increased wear of the outer ring race, but

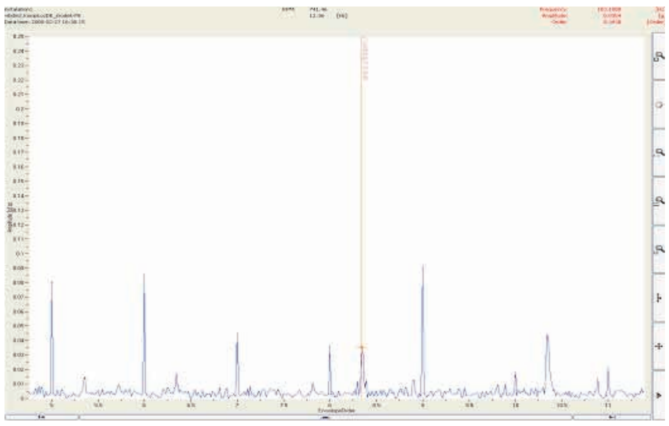

Fig. 8. Zoomed envelope order spectrum from Fig. 7.

the level did not require a repair. The machine was started again under constant human supervision.

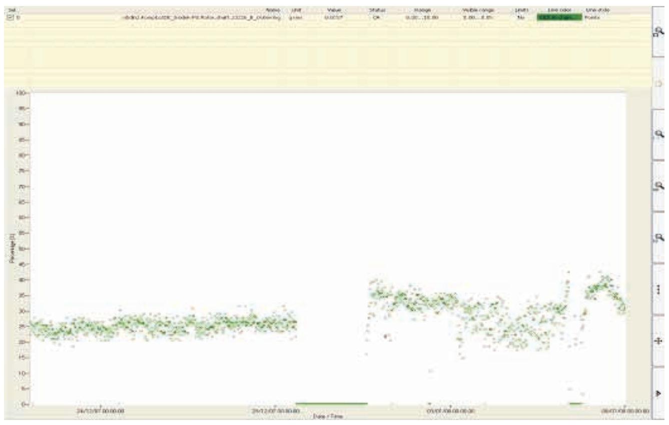

Fig. 9. Trend of the energy in the band in the envelope order spectrum, calculated around the 8.34X (outerring ball pass).

7. FINAL REMARKS

The paper presents an innovative architecture of the flexible distributed vibration monitoring system. Another important novelty is the plug-in mechanism which can be used to dramatically limit the amount of data sent to the graphical user interface.

This functionality is a must for remote access. The proposed architecture was efficient enough to implement very complex processing algorithms, including synchronous re-sampling and enveloping. The system worked for 16 channels with 25kHz sampling. The system was developed on hardware which was available off-the-shelf, so there was no need for development of any specific hardware. Further research over the proposed architecture of the system will be focused on the development of novel algorithms and their implementation in the data processing modules for specific purposes. Such modules can be e.g. new diagnostic support modules, i.e. for the determination of limit values, based on history of the data.

8. NOTATION

IT –Information Technology
UML – Unified Modeling Language
RPC – Remote Procedure Call
DCOM – Distributed Component Object Model
XML – eXtensible Markup Language
VMDS – Vibration Monitoring and Diagnostic System
CPU – Central Processing Unit
COM – Component Object Model
CORBA – Common Object Request Broker Architecture
DCS – Distributed Control System
SCADA – Supervisory Control And Data Acquisition
SQL – Structured Query Language
UI – User Interface
f_r – rotational speed of the shaft [Hz],
α – bearing load angle, D – bearing diameter, d – rolling element diameter, n – number of rolling elements.

REFERENCES

1. Barszcz T.: *Monitoring and diagnostic systems for machinery*. Wyd. ITE, Radom 2006. (in Polish)
2. Barszcz T., Uhl T., Hanc A.: “Proposal of concept of distributed vibration monitoring system for the set of fans”. *Proc. of 19th international congress COMADEM*, Lulea, Sweden, 2006, pp. 1–9.
3. Klein U.: *Vibrodiagnostic assesment of machines and devices*. Stahleisen Verlag, Duesseldorf 2003. (in German)
4. McFadden P. D.: “Interpolation techniques for time domain averaging of gear vibration”, *Mechanical Systems and Signal Processing*, vol. 3, no. 1, 1989, pp. 87–97.
5. Uhl T., Barszcz T., Hanc A.: “Mechatronics in design of monitoring and diagnostic systems”. *Key Engineering Materials*, vols. 245–246, Trans Tech Publications, 2003, pp. 381–390.
6. *Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns, Elements of Reusable Object- Oriented Software*, Addison-Wesley Pub. Co., 1995.

7. Birmann K. P.: *Reliable Distributed Systems: Technologies, Web Services and Applications*. Springer Science and Bussiness Media, 2005.
8. Bloomer J.: *Power programming with RPC*, O'Reilly, 1992.
9. St. Laurent S., Fitzgerald M.: *XML Pocket Reference*. O'Reilly, 2005. [10.] Ho D., Randall R. B.: "Optimization of bearing diagnostic techniques using simulated and actual bearing fault signals". *Mechanical Systems and Signal Processing*, vol. 14, no. 5, 2000, pp. 763–788.
11. Antoni J., Randall R. B.: "Unsupervised noise cancellation for vibration signals: part II – a novel frequency-domain algorithm". *Mechanical Systems and Signal Processing*, vol. 18, no. 1, 2004, pp. 103–117.