Computer Standards & Interfaces, 2013, Volume 35, Issue 2, February 2013, Pages 238–246

Distributed Embedded Condition Monitoring Systems based on OSA-CBM Standard

T. Sreenuch[1,2], A. Tsourdos[2] and I. K. Jennions[1]

[1]Integrated Vehicle Health Management Centre, Cranfield University, Conway House, University Way, Cranfield, Bedford MK43 0FQ, UK

[2]Department of Informatics and Systems Engineering, Cranfield University – CDS, Shrivenham, Swindon SN6 8LA, UK

Email: t.sreenuch, a.tsourdos, i.jennions@cranfield.ac.uk

***Abstract*—This paper presents an approach to distributed condition monitoring systems that offers a reusable software architecture for a class of condition monitoring (CM) applications. The focus of this paper deals with an open software framework for development of CM applications stemming from 1) the Open System Architecture for Condition Based Maintenance (OSA-CBM) specification, which is an architecture promoting interoperability, and 2) a component framework that enables reuse, data process partitioning, configuration and rapid deployment. The publish/subscribe mechanism is the primary model used for both intra- and inter-module communications. The framework is developed using Java and Remote Method Invocation (RMI) distributed middleware, and its application is demonstrated through a gearbox CM system, where the CM software are deployed on the distributed embedded devices. This approach provides software enabled capability to distribute/re-configure the CM data process (through the OSA-CBM common interface and data model) across the hardware platforms to meet the given system configuration.**

1. Introduction

In today's globally competitive environment, operators are seeking to maximize their asset usage and hence to demand a comprehensive engineering asset management system. More reliance is now being placed on condition monitoring (CM) systems to extend functional life of the assets and to allow repairs to maximize affordability [1]. From the systems development view point, the major obstacles in implementing CM systems are: the development of software systems for platform distributed embedded devices, the integration of the existing/new health ready components and also the other maintenance systems. A software development framework that makes appropriate use of open standards would provide a great deal of benefit toward ease of development and integration between providers.

Several examples of distributed condition monitoring are reported in the literatures. [2] provides an example of the multi-agent approach which applies to measurement of power electronic systems. In [3]–[4], a modern software technology like .NET is used to create a generic particle-filtering based framework for fault diagnosis and failure prognosis. The framework aims is to enhance productivity in the development of CM applications. The approach has been applied to fault diagnosis/prognosis of bearing and fault diagnosis of brushless DC motor. [5] reports an example of distributed fault detection based on wireless sensor network. The system is developed for CM of machine cutting tools. In this system, raw data are transmitted using low-level communication protocol from sensor nodes and then processed using LabVIEW on a central computer. Neither standardized interface nor data model are used in [2]–[5], and hence a key disadvantage of using these approaches is of that limitations in term of interoperability.

IEEE 1451 smart transducer interface standard is an open standard for distributed measurement and control. In IEEE 1451 based implementations, a standardized software structure and interface called Network Capable Application Processor (NCAP) are used to enable the distribution of the measurement and control of the systems [6]. Most of the reported works implement the NCAP client/server and publish/subscribe models based on Web Service middleware. Its applications are found in the areas of water management [7]–[8], industrial automation [9] and environmental air pollution monitoring [10]. In addition to the Web Service, .NET can also be used to enhance productivity in the NCAP prototyping [11]. [7]–[11] are the examples of how the interoperability issue at the data acquisition level of CM systems can be solved using the IEEE 1451 standard.

IEEE 1232 standard for Artificial Intelligence Exchange and Service Tie to All Test

Environments (AI-ESTATE) is a standard providing a set of formation data specifications to be exchanged between diagnostic reasoners [12]. [13]–[14] reports application examples of AI-ESTATE which are on electronics test circuits. In the examples, Fault Tree, Fuzzy Set and Bayesian Belief Network reasoners are used to demonstrate the adequacy of AI-ESTATE data exchange model. In term of CM systems, AI-ESTATE addresses interoperability at the health assessment level (see section 2).

However, in CM applications, there are many more data processing steps apart from measurement and reasoning addressed by the IEEE 1451 and AI-ESTATE standards. Moreover, the data generated by the platform CM components should be able to seamlessly map into the maintenance database for further maintenance operations or data mining. Open System Architecture for Condition Based Maintenance (OSA-CBM) is a needed standard for developing and integrating CM systems [15]–[16]. Note that IEEE 1451 and OSA-CBM can co-exist and complement each other. The integration of network sensors is addressed by the IEEE 1451 standard, while the OSA-CBM specification addresses the integration of the software components performing the higher-level of CM data process. However, implementing the standard is by no mean a trivial task, and there are very few OSA-CBM implementation examples are available in the literature. Therefore, a need exists to create an OSA-CBM compliant open software framework which would enable adopting CM implementers to accelerate or ease the development (or prototyping) process of CM applications. An approach that allows rapid prototyping and re-configuration of CM data process without re-implementing most part of the system would be of particular interest. This and its usage example will be the contribution of this paper.

Open System Architecture for Enterprise Application Integration (OSA-EAI) is another CM related standard worth mentioning for completeness. OSA-EAI is a relational database specification for maintenance systems [17]. It focuses on information integration of Asset Lifecycle Management (ALM) applications through a common standardized maintenance database. Examples of OSA-EAI compliant commercial off-the-shelf (COTS) software are IBM Maximo Oil and Gas, Rockwell Emonitor Odyssey and Emerson Process Management RBMWare [18]. OSA-EAI addresses interoperability between ALM applications which is a level above CM systems. However, OSA-EAI and OSA-CBM are closely related. OSA-CBM format data can in fact be mapped to any OSA-EAI compliant databases [19]. This will essentially allow better integration between CM systems and enterprise ALM applications.

The outline of this paper is as follows: Section 2 describes the OSA-CBM specification and

its interpretation. Section 3 outlines the proposed software development framework and describes the technologies used for implementing the framework. An example of how the proposed framework is applied to a specific CM application, i.e. Gearbox, is described in section 4 and 5. Section 6 discusses the benefits of an open standard like OSA-CBM, and then we conclude the paper.

2. OSA-CBM Specification

The OSA-CBM specification [20] is an open standard architecture for moving information in a condition based maintenance system. Its goal is to enhance interoperability between multiple vendors' software components [21]. The architecture is divided into the interface specification and information specification or data model (see Fig. 1). These specifications are defined using the Unified Modeling Language (UML) and intended to be platform independent which can be mapped into various programming languages and middleware technologies. OSA-CBM is now managed and published by the Machine Information Management Open Systems Alliance (MIMOSA) standards body.

2.1. Data Model

The OSA-CBM data model is based on the concept and employment of metadata (data about data), i.e. OSA-CBM data are always identifiable and traceable. The aim is to have data that supports the database centric maintenance information management. In fact, OSA-CBM data can be directly mapped into any OSA-EAI-compliant relational database maintenance systems with ease. There are 4 primary OSA-CBM data classes: DataEvent, Configuration, Explanation and Extensible. Extensible class is still immature and hence is excluded the diagram shown in Fig. 1. Configuration gives information about an OSA-CBM module's input sources, a description of algorithms used for processing input data, a list of outputs and various output specifics such as engineering units, thresholds for alerts, etc. Explanation is the data or a reference to the data used by an OSA-CBM module to produce an output. Note that Configuration and Explanation Data are optional for embedded systems such as a small platform CM system.

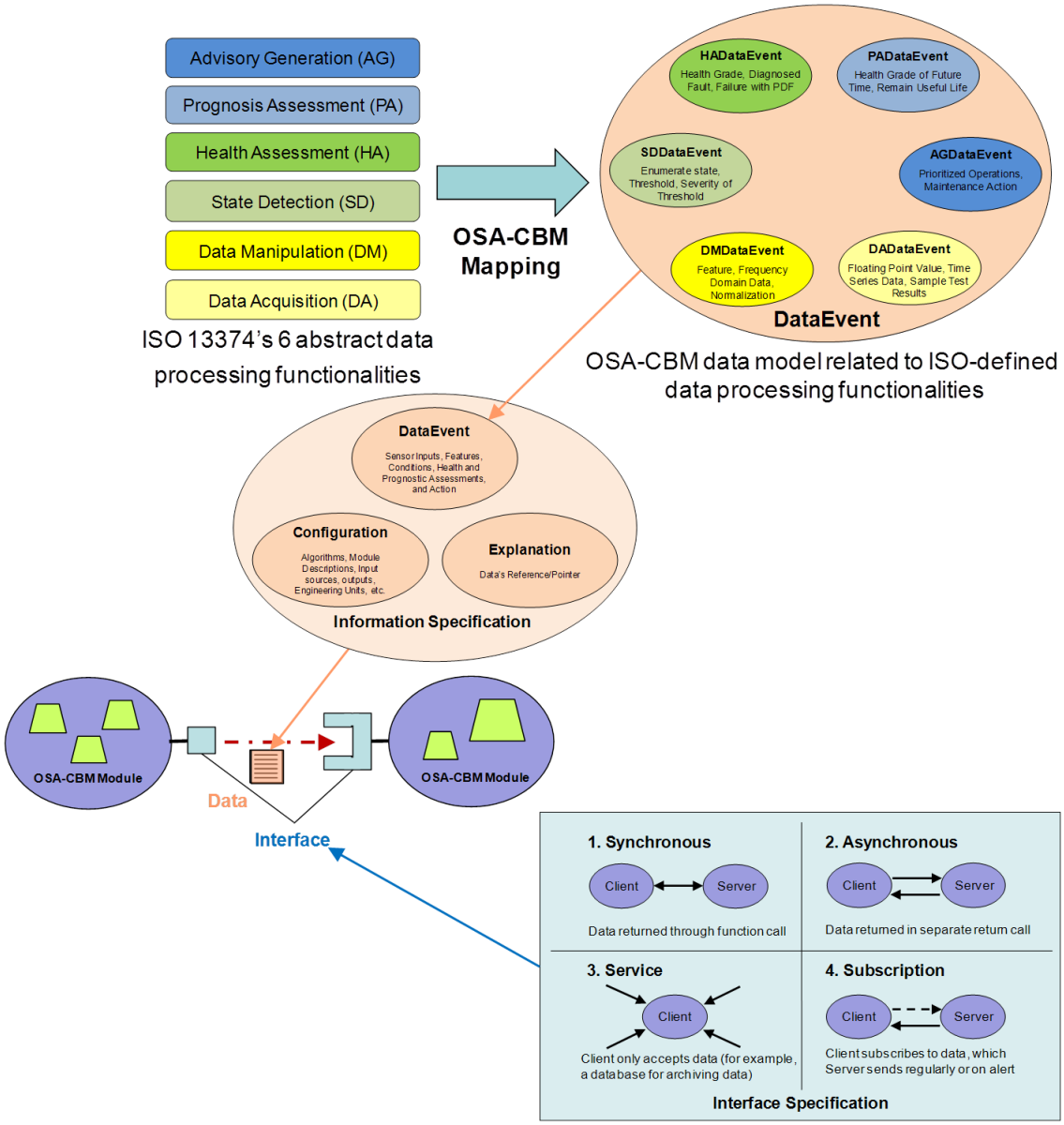

Fig. 1. OSA-CBM Interface and Information Specifications.

DataEvent is the dynamic data related to CM events generated by an OSA-CBM module such as measurements, manipulated or processed data, etc. The DataEvent class forms a substantial part of the OSA-CBM data model. The DataEvent child hierarchy is associated with particular abstract CM data processing functionalities defined in the ISO 13374 Condition Monitoring and Diagnostics of Machines – Data Processing, Communication and Presentation [22]. Those classes have child classes below them describing particular data types. The functional definitions can be viewed as abstractions of data into knowledge with higher and higher abstraction level of information. The DA data type is an acquired sensor

data which is formatted into a consistent form. The DA data is then transformed into one or more meaningful features, which are in the DM-type data format. The features are compared against expected values, and the resulting enumerated condition indicators are stored in the SDDataEvent class. The data in the HA, PA and AG formats are the data related to current health of the machine, predicted future failures and recommended action steps, respectively.

2.2. Interface Specification

The interface specification describes how information will be moved. There are 4 primary types of interface to accommodate different purposes and technological capabilities: Synchronous, Asynchronous, Service and Subscription. The synchronous interface returns data with the call. It models the Web XML over HTTP fetch technology. The asynchronous interface allows any number of higher modules to establish and maintain a two-way connection for duration of need. The data is returned either on request, on alert or by push all which pushes data to the higher connected modules every time it collects data without the need for a request beforehand. The service interface is for a one-way data input device. Two possible uses would be 1) data storage utility and 2) maintenance advisory receiver service. The subscription interface is similar to the return on alert or the push all modes of communication in the asynchronous interface. The upper module subscribes to data, which the lower module sends regularly or on alert. A given implementation will likely not implement every interface types, and it is also not compulsory that all Application Programming Interfaces (APIs) within a chosen type of interface are implemented.

3. Development Framework

3.1. Component Model

Unlike IEEE 1451 smart transducer interface standards [23], OSA-CBM does not specify the software component types or building blocks used to design and implement application systems. How the internal software structure should be is left to the OSA-CBM implementer. A component framework approach is adopted in this paper, illustrated in Fig. 2. The framework is aimed to simplify development of OSA-CBM applications. To allow reuse, configuration, extension, several key features are identified and built into the component

model: A lightweight hierarchical component model that stresses on modularity and extensibility. In this paper, the subscription interface (which has the least API methods) is selected for simplicity in implementation and also to naturally capture the event-driven characteristic in the CM applications. The main components are described as follows:

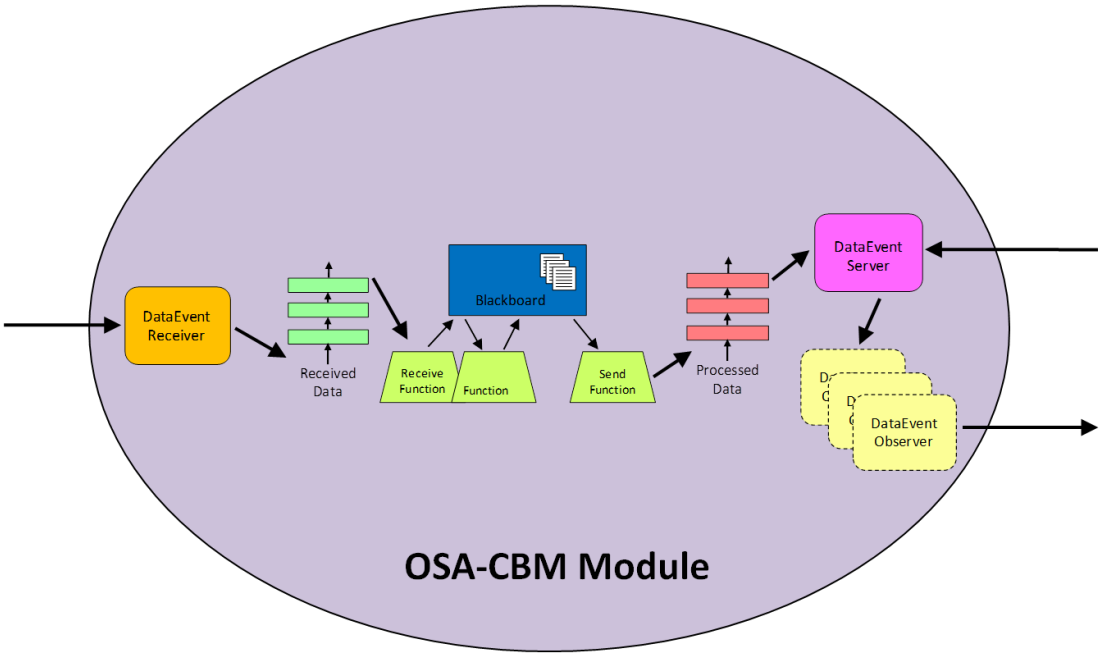

Fig. 2. OSA-CBM Component Model

3.1.1. Entry Points

The OSA-CBM interface specification's Subscription defines two interfaces for an OSA-CBM module (see Fig. 3): DataEventServer and DataEventReceiver. DataEventServer is the interface provided by an OSA-CBM module to other OSA-CBM modules that have an interest in getting data from it. When a module wants data from a server module, the requesting/client module uses the server module's DataEventServer interface to make a request. During the requesting process, the client module will provide a DataEventObserver to the server module. The DataEventObserver contains a reference to the DataEventReceiver of the client and have the notification method to be called by the server module when a new data is ready.

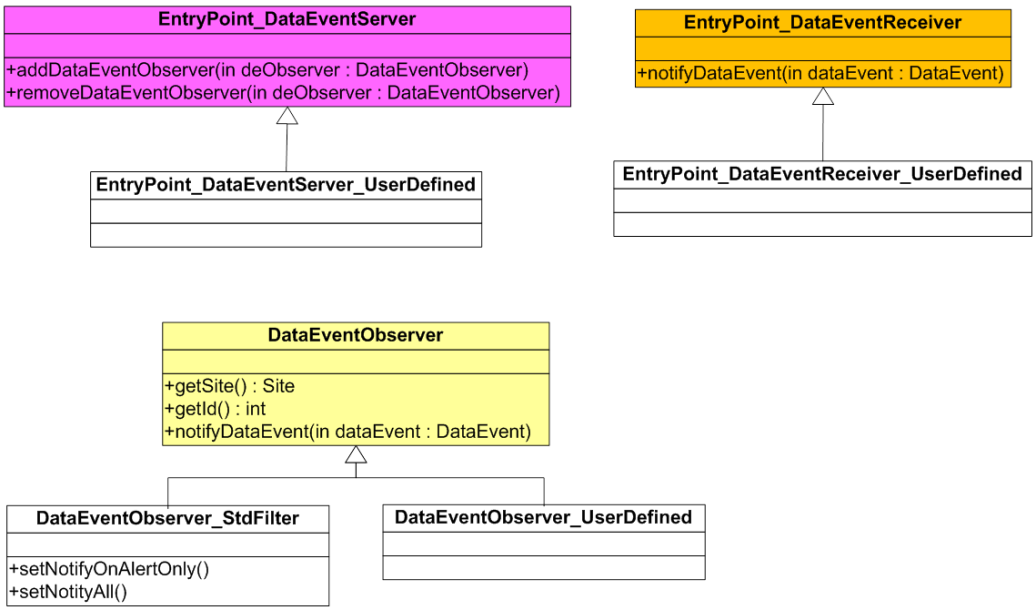

Fig. 3. OSA-CBM Subscription Interface.

3.1.2. Message Queues

The ReceivedData and ProcessedData message queues provide asynchronous communication between OSA-CBM modules. A function call to update data returns immediately after the data event has been inserted in the received data message queue. The processed data message queue acts as a data buffer stored before being sent to the subscribed OSA-CBM modules. In our framework, queues are used to solve the call blocking performance issue. Hence, the CPU can be fully utilized for the CM related data process.

3.1.3. Function

A Function is self-contained wrapper software for the data processing algorithm so that it can be used within an OSA-CBM module. Functions communicate only with the OSA-CBM module's internal components, reacting to Blackboard data events and publishing results to the Blackboard. All Functions are implemented by extending/inheriting from one of the base Functions, i.e. TransducerFunction and DataProcessingFunction in Fig. 7. Most if not all Functions, specialized behaviors are implemented by overriding the pre-defined methods of the inherited Function, e.g. NiDAQmxFunction, FilterFunction, DcOffsetFunction, TsaFunction, KurtosisFunction, ResidualFunction, FftFunction, Ena4Function and BayesFunction in Fig. 7. In this way, Functions bring functionality to an OSA-CBM module; they together are the essential computing engine of each OSA-CBM module.

3.1.4. Blackboard

The Blackboard provides the Functions with the ability to specify and interact with data event of specific interest to that Function, see also [24]. It acts as an OSA-CBM module's shared data space. The model of communication between the Blackboard and Functions is data-driven via publish-subscribe mechanism. The Blackboard tracks all changes in the data event and distributes the updates to all interested/subscribed Functions (see Fig. 10 *top-right*). Fig. 4 illustrates a linear representation of the OSA-CBM module and Function activity as the module process a single data event. In terms of OSA-CBM application development, Function or algorithm developers would only have to focus on the specialized Functions that are labeled Function #.. in the Fig. 4.

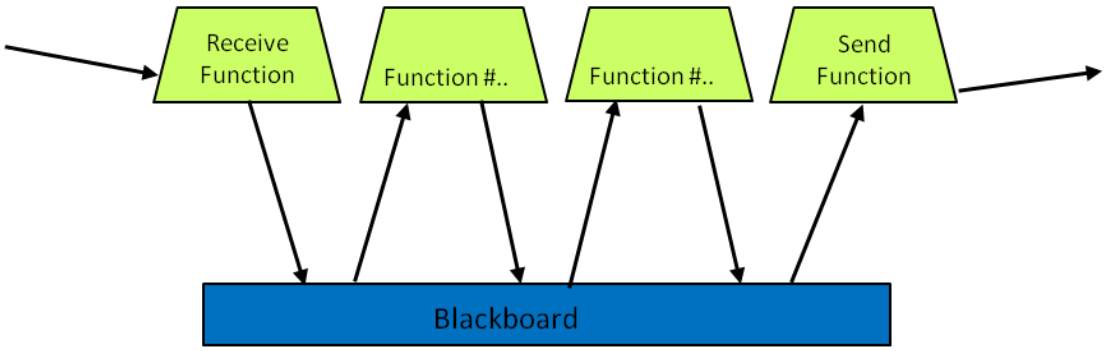

Fig. 4. Function Data Processing Flow.

These few generic software components (i.e. Entry Points, Message Queues, Blackboard and Functions) act as building blocks of an OSA-CBM module. The components are dynamically loaded and connected together at application start time using a given configuration information. The component interaction is through abstract interfaces, for example publish-subscribe API calls are used to pass data between Blackboard and Functions. A data processing flow or a complex algorithm is built using a combination of simple specialized Functions. This flexibility allows the developer to easily reuse pre-developed Functions in OSA-CBM module construction or customize a CM application to meet a required CM functionality.

3.2. Java and RMI

Embedded OSA-CBM applications are complex; different parts of CM data process are distributed across different processing units to perform the data processing in a cooperative way. However, the embedded space is rapidly changing in two ways: 1) faster processors that consume less power and 2) low cost memory. This means a greater choice of development platforms and code from broader software community can be reused. Hence, with the available CPU and memory resources, operating systems (OSs) and Java technology would make the implementation process of OSA-CBM component framework much more productive. The operating system and drivers are used to handle low-level device and network communication tasks. Java virtual machine (JVM) is a key concept in Java technology (see Fig. 5). Source code is compiled to Java bytecode, which is verified, interpreted or Just-In-Time (JIT) compiled for the native architecture. The Java APIs and JVM together make up the Java Runtime Environment (JRE). Therefore, OSA-CBM software written in Java is compiled once and can run across different Java-enabled embedded platforms (CPU: x86, PowerPC, ARM and OS: Linux, XP Embedded). The task of porting OSA-CBM code to

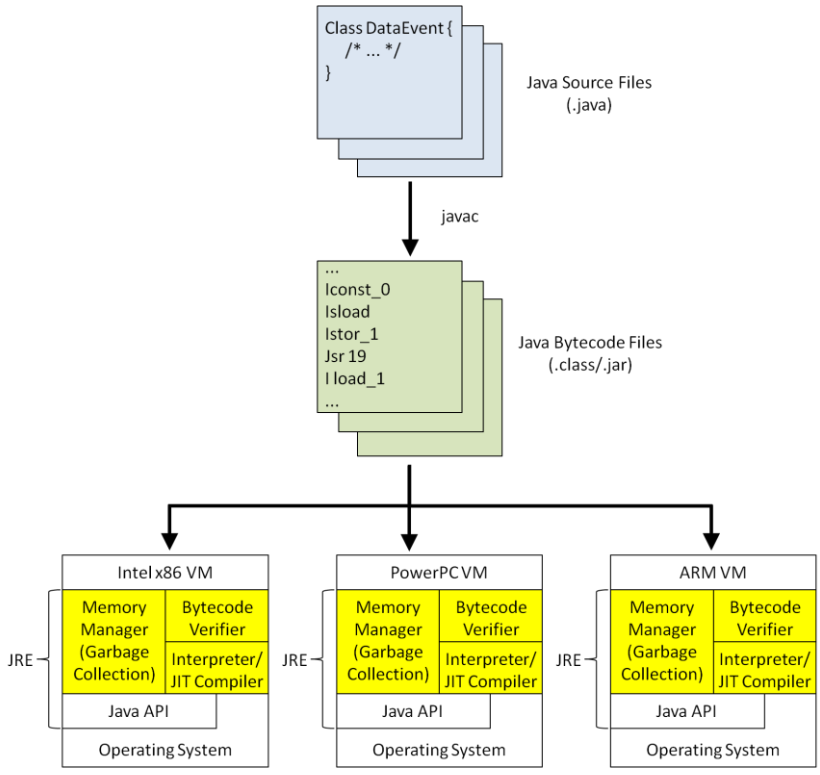

Fig. 5. Overview of Java Virtual Machine Architecture.

different embedded devices is simplified. Moreover, it is more productive to code in Java than to code in C++. The task of implementing the CM algorithms is also reduced by using many well-developed open numerical or scientific libraries (without a need for code re-compilation). Productivity (i.e. time and ease of development) offered by Java outweighs performance offered by C++ in our case, where activities are mainly on research and proof-of-concept prototyping.

In order to use or validate the information content of OSA-CBM messages, a mapping of UML data model to Java is required. MIMOSA used the Sparx Systems' Enterprise Architect UML tool for generation of the XML schema from the OSA-CBM data model. In this paper, the Java Architecture for XML Binding (JAXB) tool is then used to convert OSA-CBM XML schema to Java class representation. The generated Java classes (with XML binding) form parts of the framework and are also used in the developed OSA-CBM application for storing module's configuration data and dynamic data events.

Java Remote Method Invocation (RMI) is a distributed middleware technology. It implements the remote object model for Java objects. RMI extends the Java language with the ability to invoke a method on a remote object and to pass Java objects as parameters in the method call.

The communication process in RMI begins with a proxy object (i.e. stub) that implements the same interface as the remote object. When a method is called on the stub, it calls the remote object through the remote JVM, waits for the response and then returns the results. In our component framework (see Fig. 2), the Entry Points, i.e. DataEventServer and DataEventReceiver, are implemented by extending the RMI's Remote class. Fig. 6 shows the data flow of the corresponding subscribing and data updating process for the OSA-CBM subscription interface. The Executive module establishes a connection between the publisher, OSA-CBM module #1, and the subscriber, OSA-CBM module #2, by invoking the *addDataEventObserver(…)* method which its argument, *deObserver*, contains the details of the subscriber. Whenever the data is ready, the publisher invokes the *notifyDataEvent(…)* method to update the subscriber the new data. With every update notification, the *dataEvent* is automatically serialized into a compact binary RMI format, i.e. marshalling/unmarshalling process, and sent to the receiving module. This formatter offers a good performance for sending a data but can only be used by Java applications.

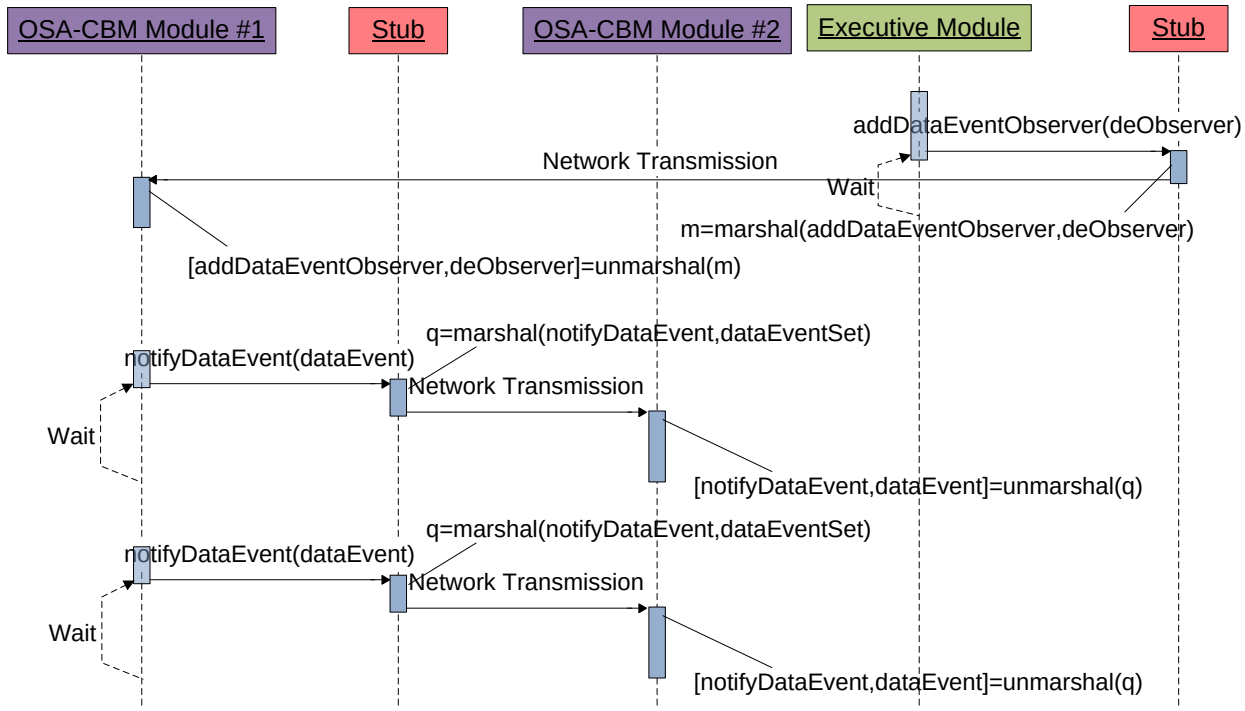

Fig. 6. Subscription and Data Updating Process.

4. Application

4.1. Gearbox Vibration Analysis

A gearbox vibration monitoring is chosen as an example to demonstrate the adequacy and effectiveness of our distributed CM software development framework. The purpose of the CM system is to monitor the vibration signals produced by the gearbox and determine the level of damage (or fault) of its pinion gear. Fig. 9 shows the experimental platform and the related CM data processing chain. In this setup, an accelerometer (Endevco 7251) is mounted on top of the gearbox to measure the vibration in the axial dimension, and an optical sensor is used to provide a one pulse per revolution signal used for measuring the speed of the gear shaft. A data acquisition board (NI 9234) is used to continuously collect 20s batch data for the vibration analysis at the rate of 25 kHz.

The data processing algorithms used in this paper are ranged from data manipulation to state detection. Based on collected data and an extensive analysis, a combination of Kurtosis and NA4* (or ENA4) of the axial axis is sufficient for accurate fault detections in this particular system. Note that, we only outline one possible example of data processing flow in

this paper, an extensive discussion of the gearbox vibration analysis can be found in [25] and [26]. To determine the condition of the gearbox, the raw tacho and acceleration signals are first conditioned by passing through the low-pass filters and the DC offset remover. The random vibrations and external disturbances are further removed by averaging multiple data segments, each segment equals to one revolution of the gear shaft. This technique is known as Time Synchronous Averaging (TSA). In addition, the shaft rotation speed can also be obtained during the TSA computation. The Kurtosis and NA4* features are then extracted from the TSA and residual signals, respectively. The residual is computed by removing the shaft frequency and gear mesh harmonics from the TSA signal. The aim is to have an analytic signal (i.e. the residual signal) which is less dependent on the speed and load of the gearbox. The Bayesian classifier processes the Kurtosis and NA4* input features and determines the gearbox condition based on the statistical properties of the training data.

4.2. Implementation

The development framework described in section 3 is employed in building the OSA-CBM based embedded CM system. The first step of the implementation is to categorize the underlying algorithms are according to the ISO 13374 abstract functionalities shown in Fig. 1, i.e. DA – NI DAQmx, DM – Filter, DC Offset, TSA, Kurtosis, Residual, NA4*, FFT and SD – Bayesian Classifier. In this paper, the NI DAQmx C library and Java Native Interface (JNI) are used for low-level communication with the NI 9234 data acquisition board, and the vibration processing algorithms are implemented based on the Java Apache Commons Mathematics Library. These generic vibration algorithms are then wrapped to form an OSA-CBM algorithms library by extending/inheriting either the TransducerFunction or DataProcessingFunction, and the resulting gearbox vibration analysis specific Functions are shown in Fig. 7. The OSA-CBM compliant data format for the corresponding inputs and outputs of the Functions is summarized in Table 1. Note that the implemented algorithm wrappers (i.e. Functions) are generic and configurable. The configuration parameters are the number of inputs and outputs, expected input data type and algorithm parameters. For example, the BayesFunction can have multiple input features which can be the same or different OSA-CBM data types and also multiple classes which the statistical parameters are the means and co-variances of each individual class.

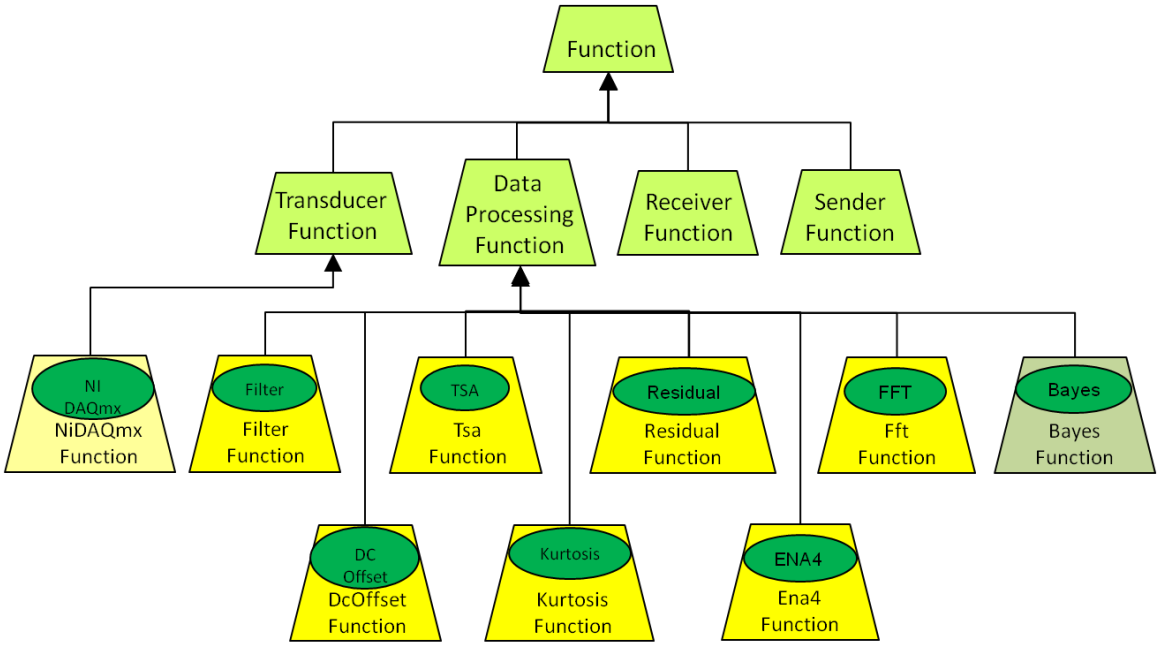

Fig. 7. Base and Gearbox-Related Specialized Functions.

Table 1.
OSA-CBM Data Types for Gearbox Algorithm Wrappers

Function	OSA-CBM Data Type	
	Input	Output
NiDAQmxFunction	NA	DAWaveform[1] , DAWaveform[2], etc.
FilterFunction	DAWaveform or RealWaveform	RealWaveform
DcOffsetFunction	DAWaveform or RealWaveform	RealWaveform
TsaFunction	DAWaveform[1] or RealWaveform[1] and DAWaveform[2] or RealWaveform[2]	DMReal[1] and RealWaveform[2]
KurtosisFunction	DAWaveform or RealWaveform	DMReal
ResidualFunction	RealWaveform	RealWaveform
Ena4Function	RealWaveform	DMReal
FftFunction	DAWaveform or RealWaveform	CmplxFrqSpect
BayesFunction	DAReal[1] , DAReal[2] , etc. or DMReal[1] , DMReal[2] , etc. or SDReal[1] , SDReal[2] , etc.	SDInt

[1] annotates first input or output. Similarly for [2], [3], etc.

Note that legacy or new algorithm implementations are practically expected to be also in different programming languages other than Java. Hence, how algorithms implemented in C/C++, .NET or MATLAB are integrated in our framework is worth discussing. JNI can be used to interface C/C++ algorithm library compiled .dll (or .so in Linux) binary files [27]. NiDAQmxFunction is in fact a JNI example which in this case is an OSA-CBM wrapper for the NI DAQmx C library. In MATLAB, algorithms can be compiled using MATLAB Builder JA to deployable Java classes which can then be used as part of a Java program [28]. Alternatively, MATLAB Coder can automatically generate portable C/C++ code from MATLAB code [29]. The generated C/C++ code can then be interfaced using JNI. .NET version of JNI exists in both proprietary and open source Java libraries. Examples are JNBridgePro and jni4net. These libraries allow a Java program to call .NET libraries (.dll files) in a similar way as JNI for native C/C++ [30]–[31].

Fig. 8 shows the network setup of the distributed CM systems. The objective of this system is to partition the CM data processing chain described in section 4.1 and to distribute the algorithms to form a hierarchical CM system. In this setup, the OSA-CBM module #1 hosts the ReceiveFunction, NiDAQmxFunction, FilterFunctions, DcOffsetFunction, TsaFunction and SendFunction. It runs on the local sensor node which is an Intel Atom D510 single-board computer (Advantech PCM 9362) attached to NI 9234 data acquisition board and has

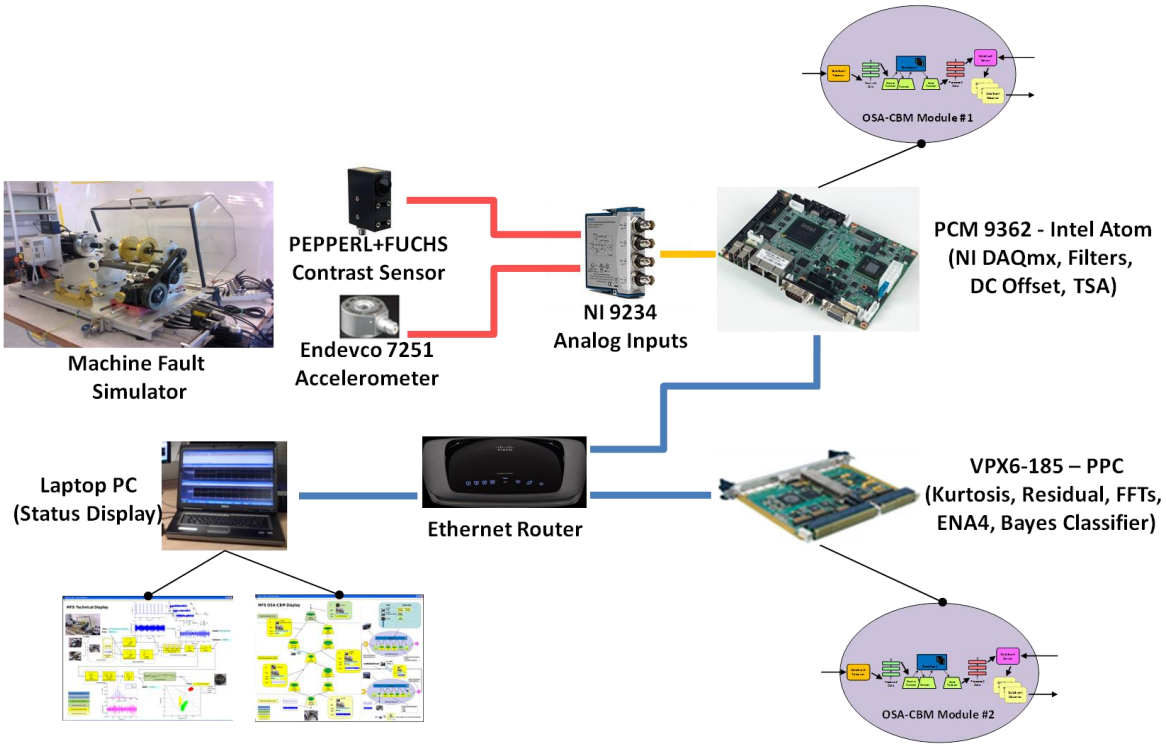

Fig. 8. Distributed Gearbox Condition Monitoring System.

Windows XP Embedded, Java SE 6 and NI DAQmx C library installed. This OSA-CBM module is for acquiring the tacho and vibration signals and to perform low-level data processing from filtering up until TSA. In this phase of data processing, for each 20s batch data, the vibration information is compressed from 500,000 samples of tacho signal + 500,000 samples of acceleration signal to TSA signal of 4,096 double data points.

The OSA-CBM module #2 hosts the ReceiveFunction, KurtosisFunction, ResidualFunction, Ena4Function, FftFunction, BayesFunction and SendFunction. It runs in a Freescale MPC8641D flight-worthy single-board computer (Curtiss Wright VPX6-185) and has a pre-compiled Linux file system from the Embedded Linux Development Kit (ELDK) version 4.2 and OpenJDK 6 installed. The CM data process extracts the Kurtosis and NA4* features from the TSA data generated by the OSA-CBM module #1 and determine the current condition of the gearbox.

Fig. 10 shows the data processing hierarchy and how the OSA-CBM modules are organized. Due to the limited display space, the Message Queues and DataEventObservers are not shown in the display window. At each OSA-CBM module start time, the module specific XML configuration file is loaded and the module is created by means of runtime instantiation and configuration of the pre-compiled Functions. The XML files are formatted according to the OSA-CBM Configuration XML schema. The configuration information contains the module description, a description of algorithms (incl. parameters) used for data processing and lists of individual algorithms' inputs and outputs.

Our current distributed gearbox CM system runs in the soft real-time mode which is more suited to novel CM applications like the advanced vibration analysis. In comparison to the hard real-time case, the time when a data is sent or arrives is determined less accurately in the soft real-time mode, though it is possible to bound the possible times. In this paper, this problem is easily mitigated by using the time stamp (at the point in the system at which data are acquired) and other metadata attributes available in the dynamic OSA-CBM data events. This way the data can be synchronized and appropriately processed by the OSA-CBM modules without the need for hard real-time properties.

Note that the current experimental setup can be extended by having a high-level CM system takes more pre-processed data from other local sensor nodes (e.g. another current sensor + processing node attached to the motor). The high-level module can then fuse all the data available and perform a more throughout system-level condition monitoring, for example, an overall drive-train system condition monitoring.

5. Result

To interact with the OSA-CBM embedded nodes, we create two small MATLAB GUI displays shown in Fig. 9 and 10. The m-files are compiled into deployable Java classes using MATLAB Builder JA (for Java language). The Java display module integrates the deployable Java classes (i.e. compiled MATLAB GUI components) for the graph plotting and text updating functionalities and also implements the standardized OSA-CBM DataEventReceiver interface for interoperability with the OSA-CBM modules.

In this setup, a static executive module (a small Java console program) is used to manage the data subscription or un-subscription between the OSA-CBM module #1, OSA-CBM module #2 and display program. The subscription and data updating mechanisms follow the sequence similar to the process shown in Fig. 6.

Fig. 9 provides the algorithmic view of the distributed gearbox CM system. It is aimed to give a transparent view of the data processing chain and how the data is transformed or computed in each data processing step. The CM system is able to make accurately assessments of the gearbox condition. On the other hand, Fig. 10 illustrates the software view of the CM system. It shows the software component hierarchy and the transformation of raw data into information that is meaningful to and can be used by the maintenance engineer. The raw measurement data are formatted by adding the metadata to form the OSA-CBM data events. As the data events are further processed by the Functions, the metadata (e.g. id) and data type are changed to reflect the transformation of the data in the CM data processing chain. In addition, the display also illustrates how the OSA-CBM modules are organized and the dynamics of the data exchange between the software components.

The CM data process operates at the fixed rate schedule of 20 s timeframe. The mean elapsed time between at the point when the raw tacho and acceleration data are acquired and the condition level of the gearbox is computed is <1 s, which is well less than the timeframe required for the real-time operation. Note that this level of computational power is typical for the embedded single-board computers used in the aerospace vehicles.

The data subscription or un-subscription between the OSA-CBM module #1 and #2 can be established dynamically anytime during the operation, through the *addDataEventObserver(..)* and *removeDataEventObserver(..)* APIs. Once a connection is established, the module #1 pushes the updated data to the module #2, and the module #2 responses to the arriving data events generated by the module #1. On the other hand, once the un-subscription is requested, the module #1 stops sending the updated data to the module #2. The module #2 effectively

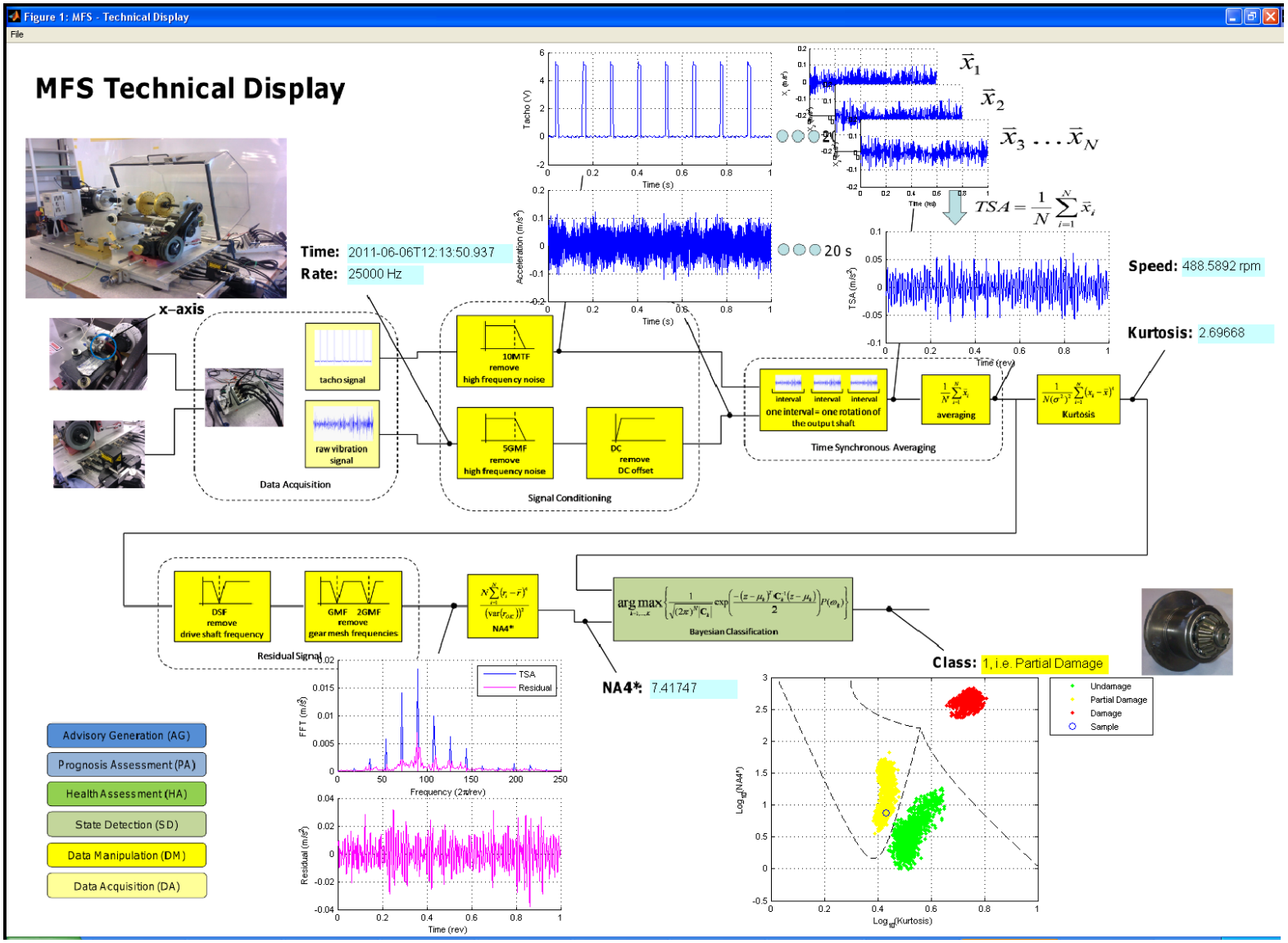

Fig. 9. Screenshot of the Gearbox CM Technical Display.

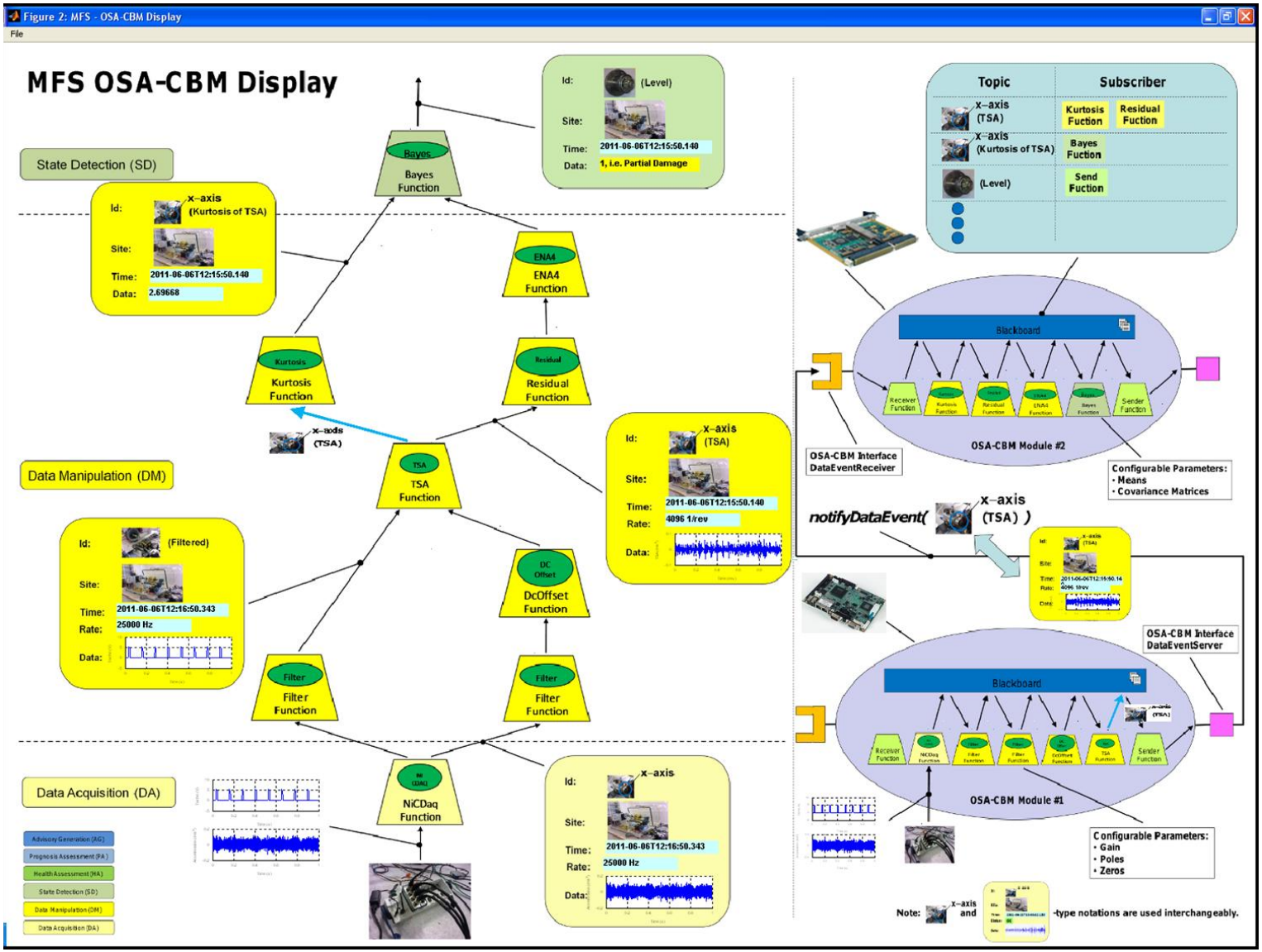

Fig. 10. Screenshot of the Gearbox CM OSA-CBM Display.

becomes idle, waiting for the incoming data events.

6. Qualitative Evaluation

From section 1, it is desirable that an OSA-CBM software development framework allows rapid prototyping and re-configuration of CM data process. At this point, it is important to stress that the OSA-CBM modules shown in Fig. 8 and 10 are not hard-coded. The OSA-CBM modules are instantiated from the given XML configuration files. Different CM applications can be built based on a pre-existing OSA-CBM algorithm library. For example, an embedded hardware could have computation capabilities to allow all the vibration algorithms described in section 4.1 to be packaged into a single OSA-CBM module. Given that OSA-CBM modules developed using the proposed framework share a common software structure. Hence, in order to create a new module, it only needs to write a configuration file that contains the required information of algorithms and their parameters. The software components are dynamically loaded and configured at the application startup time.

Meanwhile, at the system level, the distributed CM data process is dynamically configured using late binding. For example, a new OSA-CBM component can be added to the system to form a new data processing chain without making changes to an existing system. A CM data event is then be (re-)routed by invoking appropriate methods of an OSA-CBM module's EntryPoints (see Fig. 6).

In this framework, the requirements of rapid deployment (or prototyping), data processing reconfiguration (local- and system-level) and interoperability are met through the reuse of common software structure, configurable software components and OSA-CBM data model and interface.

7. Conclusion

A standard like OSA-CBM benefits in easing integration of multiple vendors' CM software components. The interoperability issue is addressed through the common standardized interface

and data model. To the CM application developers, OSA-CBM will save considerable time and effort required to develop an architecture and related data classes. The developer can make uses of the already well designed API and data model.

To enable reuse, data process partitioning, configurable and rapid deployment, a development framework likes the one described in section 3 is required. The proposed component model eases the CM implementation process as the common/generic CM tasks are handled by the framework and the developed OSA-CBM algorithm wrappers can be reused in a new application. Developers can concentrate on the application logic, i.e. CM data processing and resource allocation, rather than worrying about the software issues. With pre-existing OSA-CBM algorithm libraries, the task of creating a new CM application could be simplified to a matter of writing the OSA-CBM module configuration files.

Acknowledgment

This work was funded by Integrated Vehicle Health Management Centre, Cranfield University, UK. The authors would like to thank the IVHM Centre's industrial partners (BAE Systems, Boeing, Meggitt, Rolls-Royce and Thales) for giving numerous feedbacks and suggestions throughout the development of this work and also for helping in many other ways.

References

[1] K. Swearingen and K. Keller, "Health ready systems," in *Proceedings of the IEEE AUTOESTCON*, Baltimore, MD, 2007, pp. 625–631.

[2] F. Ponci and A. A. Deshmukh, "A mobile agent for measurements in distributed power electronic systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 5, 2009, pp. 1657–1669.

[3] C. Chen, D. Brown, C. Sconyers, B. Zhang, G. Vachtsevanos and M. E. Orchard, "An integrated architecture for fault diagnosis and failure prognosis of complex engineering systems," *Expert Systems with Applications*, vol. 39, no. 10, 2012, pp. 9031–9040.

[4] C. Chen, D. Brown, C. Sconyers, G. Vachtsevanos, B. Zhang and M. E. Orchard, "A .NET framework for an integrated fault diagnosis and failure prognosis architecture," in *Proceedings of the IEEE AUTOESTCON*, Orlando, FL, 2010.

[5] K. K. Tan, S. N. Huang, Y. Zhang and T. H. Lee, "Distributed fault detection in industrial system based on sensor wireless network," *Computer Standards & Interfaces*, vol. 31, no. 3, 2009, pp. 573–578.

[6] IEEE, *IEEE 1451.1-1999 Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processer (NCAP) Information Model*. New York, NY: The Institute of Electrical and Electronics Engineers, 1999.

[7] K. B. Lee and R. D. Schneeman, "Distributed measurement and control based on the IEEE 1451 smart transducer interface standards," *IEEE Transactions on Instrumentation and Measurement*, vol. 49, no. 3, Jun. 2000, pp. 621–627.

[8] E. Y. song and K. B. Lee, "STWS: a unified web service for IEEE 1451 smart transducers," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 8, 2008, pp. 1749–1756.

[9] E. A. Batista, L. Gonda, A. C. R. da Silva, S. R. Rossi, M. C. Pereira, A. A. de Carvalho and C. E. Cugnasca, "HW/SW for an intelligent transducer network based on IEEE 1451 standard," *Computer Standards & Interfaces*, vol. 34, no. 1, 2012, pp. 1–13.

[10] N. Kularatna and B. H. Sudantha, "An environmental air pollution monitoring system based on the IEEE 1451 standard for low cost requirements," *IEEE Sensors Journal*, vol. 8, no. 4, Apr. 2008, pp. 415–422.

[11] V. Viegas, J. M. Dias Pereira and P. M. B. Silva Girao, ".NET framework and web services: a profit combination to implement and enhance the IEEE 1451.1 standard," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 6, Dec. 2007, pp. 2739–2747.

[12] IEEE, *IEEE 1232-2010 Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*. New York, NY: The Institute of Electrical and Electronics Engineers, 2010.

[13] J. W. Sheppard, S. G. W, Butcher and P. J. Donnelly, "Demonstrating semantic interoperability of diagnostic reasoners via AI-ESTATE," in *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2010, pp. 1–10.

[14] P. J. Donnelly, L. E. Sturlaugson and J. W. Sheppard, "A standard-based approach to gray-scale health assessment using fuzzy fault trees," in *Proceedings of the IEEE AUTOESTCON*, Anaheim, CA, 2012.

[15] K. Swearingen, W. Majkowski, B. Bruggeman, D. Gilbertson, J. Dunsdon and B. Sykes, "An open system architecture for condition based maintenance overview," in *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2007, pp. 3717–3724.

[16] S. Kunche, C. Chen and M. Pecht, "A review of PHM system's architectural frameworks," in *the 54th Meeting of the Society for Machinery Failure Prevention Technology*, Dayton, OH, 2012.

[17] MIMOSA, *MIMOSA Technology Update*. Tuscaloosa, AL: Machine Information Management Open Systems Alliance, 2007.

[18] A. D. Mathew, L. Zhang, S. Zhang and L. Ma, "A review of the MIMOSA OSA-EAI database for condition monitoring systems", in *Proceedings of the World Congress on Engineering Asset Management*, Gold Cost, 2006.

[19] MIMOSA, *OSA-CBM Primer*. Tuscaloosa, AL: Machine Information Management Open Systems Alliance, 2006.

[20] MIMOSA, *OSA-CBM UML Specification 3.3.1 Release*. Tuscaloosa, AL: Machine Information Management Open Systems Alliance, 2010.

[21] M. Tiemann, "An objective definition of open standards," *Computer Standards & Interfaces*, vol. 28, no. 5, 2006, pp. 495–507.

[22] ISO, *13374-2 Condition Monitoring and Diagnostics of Machines – Data Processing, Communication and Presentation – Part 2: Data Processing.* Geneva: International Organization for Standardization, 2007.

[23] K. Lee and E. Song, "Object-oriented application framework for IEEE 1451.1 standard," in *IMTC 2004 Instrumentation and Measurement Technology Conference*, Como, Italy, 2004, pp. 1182–1187.

[24] A. Helsinger and T. Wright, "Cougaar: a robust configurable multi-agent platform," in *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2005.

[25] M. Lebold, K. McClintic, R. Campbell, C. Byington and E. Song, "Review of vibration analysis methods for gearbox diagnostics and prognostics," in *the 54th Meeting of the Society for Machinery Failure Prevention Technology*, Virginia Beach, VA, 2000, pp. 623–634.

[26] P. Vecer, M. Kreidl and R. Smid, "Condition Indicators for Gearbox Condition Monitoring Systems," *Acta Polytechnica*,vol. 45, no. 65, 2005, pp. 35–43.

[27] S. Liang, *The Java Native Interface: Programmer's Guide and Specification*. Reading, MA: Addison-Wesley, 1999.

[28] MathWorks, *MATLAB Builder JA: User's Guide*. Natick, MA: MathWorks, 2011.

[29] MathWorks, *Code Generation from MATLAB: User's Guide*. Natick, MA: MathWorks, 2011.

[30] JNBridge, *JNBridgePro: Bridge Java and .NET* (website): http://www.jnbridge.com. Accessed September 28, 2012.

[31] jni4net, *jni4net: Bridge Between Java and .NET* (website): http://jni4net.sourceforge.net. Accessed September 28, 2012

Distributed embedded condition monitoring management systems based on OSA-CBM standard

Sreenuch, Tarapong

2013-02-28T00:00:00Z

T. Sreenuch, A. Tsourdos, I.K. Jennions, Distributed embedded condition monitoring management systems based on OSA-CBM standard, Computer Standards & Interfaces, Volume 35, Issue 2, February 2013, Pages 238–246.

http://dx.doi.org/10.1016/j.csi.2012.10.002