STANDARDS DEVELOPMENTS FOR CONDITION-BASED MAINTENANCE SYSTEMS

Michael Thurston and Mitchell Lebold
Applied Research Laboratory
Penn State University
P.O. Box 30
State College, PA 16804-0030

Abstract: An effort is underway to develop an Open System Architecture for Condition-Based Maintenance. The architecture development has focused on the definition of a distributed software architecture for CBM. The distributed software model was selected due to the recent emergence of enabling software technologies and the benefits of the approach. In particular, the availability of network connectivity provides a ready hardware backbone over which the software system may be distributed. The requirements for a general CBM architecture are defined, and the framework of the distributed architecture is provided.

Keywords: Condition-based Maintenance; Condition Monitor; Health Assessment; MIMOSA; Open System Architecture; Prognostics

Introduction: Condition Based Maintenance (CBM) is becoming more wide-spread within US industry and the military. The factors that have driven an increase in the use of CBM include the need for: reduced maintenance and logistics costs, improved equipment availability, and protection against failure of mission critical equipment.

A complete CBM system comprises a number of functional capabilities: sensing and data acquisition, signal processing, condition and health assessment, prognostics, and decision aiding. In addition, a Human System Interface (HSI) is required to provide user access to the system. The implementation of a CBM system usually requires the integration of a variety of hardware and software components. Across the range of military and industrial application of CBM, there is a broad spectrum of system level requirements including: communication and integration with legacy systems, protection of proprietary data and algorithms, a need for upgradeable systems, and limits implementation time. Most purchasers of CBM systems or equipment are also very sensitive to the costs.

Standardization of specifications within the community of CBM users will, ideally, drive the CBM supplier base to produce interchangeable hardware and software components. A non-proprietary standard that is widely adopted will result in a free market for CBM components. The potential benefits of a robust non-proprietary standard include: improved ease of upgrading for system components, a broader supplier community resulting in more technology choices, more rapid technology development, and reduced prices. This paper will describe an effort underway to develop an open system standard for CBM systems.

Open Systems and Standards: Openness is a general concept that denotes free and unconstrained sharing of information. In its broadest interpretation, the term "open systems" applies to a systems design approach that facilitates the integration and interchangeability of components from a variety of sources. For a particular system integration task, an open systems approach requires a set of public component interface standards and may also require a separate set of public specifications for the functional behavior of the components. The underlying standards of an open system may result from the activities of a standards organization, an industry consortium team, or may be the result of market domination by particular product (or product architecture). Standards produced by recognized standards organizations are called de jure standards. De facto standards are those that arise from the market-place, including those generated by industrial consortia. Regardless of the development history of the standards that support an open system, it is required that the standards are published, and publicly available at a minimal cost. An example of an open de jure standard is the IEEE 802.3 which defines medium access protocols and physical media specifications for LAN Ethernet connections. An example of a proprietary de facto standard is the Windows OS (operating system). Examples of open de facto standards are the UNIX OS and HTTP.

An open system standard that receives wide-spread market acceptance can have great benefits to consumers and also to suppliers of conformant products. The emergence of the IBM PC architecture as a market leading de facto standard stimulated the market for both PC hardware and software suppliers. It also led to price reductions due to market competition in the face of rapid technology advances. The emergence of a dominant proprietary standard for PC operating systems and software resulted in benefits to consumers in terms of application interoperability, but arguably at the cost of increased prices and reduced performance compared to the possibilities that an open system software standard might have offered. In addition to commercial issues, the interchangeability of components enabled by an open system architecture yields several technical benefits: system capability can be readily extended by adding additional components, and system performance can be readily enhanced by adding components with improved or upgraded capabilities.

CBM Architectures: A complete architecture for CBM systems should cover the range of functions from data collection through the recommendation of specific maintenance actions. The key functions that facilitate CBM include:

- sensing and data acquisition
- signal processing and feature extraction
- production of alarms or alerts
- failure or fault diagnosis and health assessment
- prognostics: projection of health profiles to future health or estimation of RUL (remaining useful life)
- decision aiding: maintenance recommendations, or evaluation of asset readiness for a particular operational scenario
- management and control of data flows or test sequences

- management of historical data storage and historical data access
- system configuration management
- human system interface

Typically, CBM system integrators will utilize a variety of COTS (commercial off the shelf) hardware and software products (using a combination of proprietary and open standards). Due to the difficulty in integrating products from multiple vendors, the integrator is often limited in the system capabilities that can be readily deployed. For some applications a system developer will engineer an application specific system solutions. When user requirements drive custom solutions, a significant part of the overall systems engineering effort is the definition and specification of system interfaces. The use of open interface standards would significantly reduce the time required to develop and integrate specialized system components. CBM system developers and suppliers must make decisions about how the functional capabilities are distributed, or clustered within the system. Due to integration difficulties in the current environment, suppliers are encouraged to design and build components which integrate a number of CBM functions. Furthermore, proprietary interfaces are often used to lock customers into a single source solution, especially for software components. An ideal Open System Architecture for CBM should support both granular approaches (individual components implement individual functions) and integrated approaches (individual components integrate a number of CBM functions).

A given CBM architecture may limit the flexibility and performance of system implementations if it does not take into account data flow requirements. To support the full range of CBM data flow requirements, the architecture should support both time-based and event-based data reporting and processing. Time-based data reporting can be further categorized as periodic or aperiodic. An event-based approach supports data reporting and processing based upon the occurrence of events (limit exceedences, state changes, etc...). A specific requirement that may be imposed on a CBM system involves the timeliness of data reporting. Timeliness requirements may be defined broadly as time-critical or non time-critical. The non time-critical category applies to data messages or processing for which delays have no significant impact on the usefulness of the data or processing result. The time-critical category implies a limited temporal validity to the data or processing result requiring deterministic and short time delays [1]. Two different messaging approaches may also be employed within a system, synchronous and asynchronous. In the synchronous model, the message sender waits for a confirmation response from the message receiver before proceeding to its next task. In the asynchronous model, the message sender does not wait for a response from the receiver and closes the communication. The asynchronous model is generally more applicable to time-critical communications.

A variety of communication models exist for communication between a network of components, they include: multicast, broadcast, and client-server. In the multicast model, the information supplier publishes his information to the network, addressed to a known list of recipients; an asynchronous approach. In the broadcast model, the information supplier publishes his information to all network listeners and the listeners must decide if they are interested in the content of the message. Finally, the c-s (client-server) model pairs a client (who initiates communication) with a server (who is designed

to respond to certain requests). The server implements interfaces that may be used by a client to request a service. A client can only request services available at the server's public interfaces. Data passing may be implemented by means of a single synchronous message, or through a pair of asynchronous messages.

Current PC, Networking, and Internet technologies provide a readily available, cost effective, easily implemented communications backbone for CBM systems. These computer networks are built over a combination of open and proprietary standards. Software technologies are rapidly developing to support distributed software architectures over the Internet and across LANs. There is a large potential payback associated with market acceptance of an open standard for distributed CBM system architectures. With the ready availability of network connectivity, the largest need is in the area of standards for a distributed software component architecture; that subject will be the focus of the remainder of this paper.

One model for distributed computing is Web-based computing. The Web-based model utilizes HTTP servers which function primarily as document servers. The most common medium of information transport on the Web is the HTML page; HTML is a format for describing the content and appearance of a document. An alternate format for information transport over the Web is becoming increasingly popular; XML (eXtensible Mark-up Language). In contrast to HTML, XML is focused on describing information content and information relationships. XML is readily parsed into data elements that application programs can understand and serves as an ideal means of data and information transport over the web. A simple model for data access over the web is a smart sensor which periodically posts new data in XML format to a Web page. Information consumers may access that updated data directly from the Web-page. HTTP servers also provide remote access to application programs by means of the Common Gateway Interface (CGI). In this model, the interface between the remote client and the Web server is by means of HTML pages or XML; the webserver utilizes the CGI to communicate with the application program. The web-based distributed computing model requires that each data server have HTTP server software. With the development of compact and embedded HTTP server software it remains a feasible approach.

With the growth of distributed computing, a class of software solutions are evolving which enable tighter coupling of distributed applications and hide some of the inherent complexities of distributed software solutions. The general term for these software solutions is middleware. Fundamentally, middleware allows application programs to communicate with remote application programs as if the two programs were located on the same computer. Current middleware technologies include: CORBA, Microsoft's DCOM, SUN's Java-RMI, and Web-based RPC (Remote Procedure Call). A brief discussion of the middleware options will be given here, those with a more detailed interest are referred to [2]. CORBA (Component Object Request Broker Architecture) is an open middleware standard developed and maintained by the Object Management Group (OMG) [3]. A number of companies have CORBA based product offerings for a variety of hardware and OS platforms. DCOM (Distributed Component Object Model) is the extension of Microsoft's object technology to distributed software objects[4]; DCOM is built into the Windows 2000 operating system and Windows NT 4.0. A number of software companies have ported DCOM to other OS platforms, however DCOM is just

one component of the complete solution for distributed computing which is provided by Windows 2000. The SUN solution for distributed computing uses JAVA RMI (remote method invocation) for managing calls to distributed Java objects [5]. JAVA RMI operates over IIOP (the Internet Inter-Orb Protocol)[6], the CORBA protocol for communication between distributed software components. This allows JAVA RMI based solutions some level of interoperability with CORBA based solutions.

A middleware technology using XML based RPC over HTTP is emerging as a possible solution for distributed software systems on the World Wide Web. XML based RPC utilizes XML syntax for the execution request and for returning the results of the remote program. The leading standard for XML based RPC is SOAP [7] (the Simple Object Access Protocol) being developed by Microsoft and DevelopMentor. The XML based RPC approach provides more transparent access to distributed applications than does the use of CGI scripts. The "full service" middlewares provide better application integration at the cost of higher upfront development costs, and should provide better system performance. However, there is a simplicity associated with the XML based approaches that is attractive.

Looking ahead, it is likely that the market will support a Web-based middleware and one or more full service middlewares. It is very difficult however, to project which (if any) of the current technologies will dominate, or when some new technology will come along and make the current technologies obsolete. A prudent approach would be to develop a core architecture standard which is technology independent, and extend the core architecture with implementation specifications for several of the current implementation technologies.

OSA/CBM: An industry led team has been partially funded by the Navy through a DUST (Dual Use Science and Technology) program to develop and demonstrate an Open System Architecture for Condition Based Maintenance. The team participants cover a wide range of industrial, commercial, and military applications of CBM technology: Boeing, Caterpillar, Rockwell Automation, Rockwell Science Center, Newport News, and Oceana Sensor Technologies. Other team contributors include the Applied Research Laboratory at Penn State, and MIMOSA (Machinery Information Management Open System Alliance). The focus of the team is the development and demonstration of a software architecture that facilitates interoperability of CBM software modules. This section will give an overview of the architecture being developed.

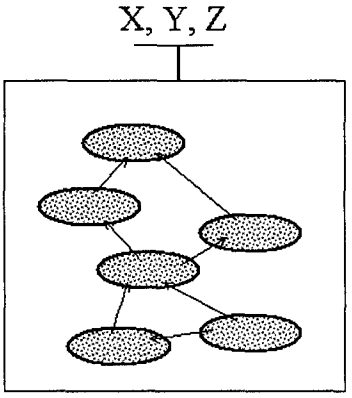
MIMOSA is a not-for-profit trade association founded in 1994 and incorporated in December of 1996. Their general purpose is the development and publication of open conventions for information exchange between plant and machinery maintenance information systems. The core of the MIMOSA development activity is the MIMOSA CRIS (Common Relational Information Schema). The second version of the CRIS (CRIS V2.1) was released in May 2000 and is publicly available at the MIMOSA website [8]. The CRIS defines a relational database schema for machinery maintenance information. The schema provides broad coverage of the types of data that need to be managed within the CBM domain:

- A description of the configuration of the system/equipment being monitored

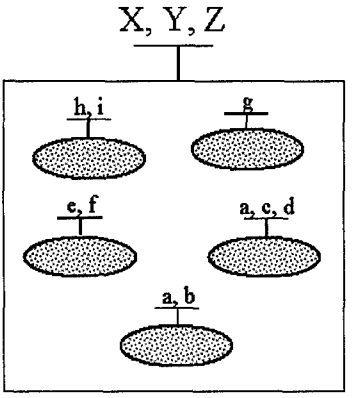
- A list of specific assets being tracked, and their detailed characteristics
- A description of equipment functions, failure modes, and failure mode effects
- A record of logged operational events
- A description of the monitoring/measurement system (sensors, data acquisition, measurement locations, etc.) and the characteristics of monitoring components (calibration history, model number, serial number, etc.)
- A record of sensor data (and its characteristics) whether acquired on-line, manually logged, or manually acquired using hand held roving instrumentation.
- A means of describing signal processing algorithms and the resulting output data
- A record of alarm limits and triggered alarms
- A means of describing diagnoses of evolving equipment faults and projections of equipment health trends.
- A record of recommended actions and the basis of those recommendations
- A record of work requests from initiation through completion.

Based upon the CRIS, MIMOSA has also defined several system interface standards. A set of import/export file formats has been defined and released, followed by a set of SQL client/server interfaces. Currently under development is a complete set of XML interfaces for CRIS V2.1 defined by means of DTDs (XML Document Type Definition). These will be released after beta testing is completed.

The OSA/CBM development approach was formulated based on the assumption that the large body of work that constitutes the MIMOSA open standards would be used as a basis for development. The CRIS represents a static view of the data produced by a CBM system. The MIMOSA interface standards define open data exchange conventions for sharing of static information between CBM systems (openness at the intra-system level). The goal of the OSA/CBM project is the development of an architecture (and data exchange conventions) that enables interoperability of CBM components (openness at the inter-system level). Figure 1 illustrates the distinction being made. Figure 1a illustrates a CBM solution composed of components with proprietary interfaces, but open at the intra-system level. The proprietary system solution is enclosed in a MIMOSA compliant wrapper which exposes a set of public MIMOSA compliant server interfaces (X, Y, Z). The interface set (X, Y, Z) allows external clients open access to the information generated within the proprietary system solution. Figure 1b illustrates a CBM system open at the inter-system and intra-system levels. In this case the individual components all expose public server interfaces (a, b, c, ... , i). These component interfaces offer access to the data and services supplied by the component, and provide for open information flow between components during system operation. In addition, components may be readily replaced by components with improved capability as long as they follow the same public interface standards. If the interface standard is an open standard that has market acceptance, there should be available COTS components which may be readily integrated for the purpose of expanding or upgrading the system capability. If the open component interfaces are based on the same information model as the open system level interfaces, the mapping between the two sets of interfaces will be significantly simplified.



a) MIMOSA compliant CBM system with proprietary component interfaces



b) MIMOSA compliant CBM system with open component interfaces

Figure 1: Granularity of an Open System Solution

After evaluating a variety of architectural options, a decision was made to develop an object oriented architecture based upon a client-server approach to distributed computing. The initial architecture development direction was focused towards the loosely coupled Web-based approach and XML messages over HTTP. There were significant concerns that this approach would limit the usefulness of the resulting standard based on performance limitations. However, tying the architecture to a specific full-service middleware brought up concerns about being tied to proprietary middleware technology, or selecting a technology which might achieve limited market acceptance. The approach that was ultimately adopted was to develop a technology-neutral abstract design specification, which could then be mapped to implementations utilizing any of the implementation technologies discussed earlier. The decision was also made to utilize an object oriented design methodology as a core strategy. The components of the architecture will be discussed in more detail later.

In order to standardize an architecture for CBM components, the first step as suggested by Figure 1b is to assign the CBM system functions defined earlier to a set of standard software components. The software architecture has been described in terms of functional layers. Starting with sensing and data acquisition and progressing towards decision support, the general functions of the layers are given below. A complete description of the inputs and outputs required for a given layer is beyond the scope of this paper.

Layer 1 – Sensor Module: The sensor module has been generalized to represent the software module which provides system access to digitized sensor or transducer data. The sensor module may represent a specialized data acquisition module that has analog feeds from legacy sensors, or it may collect and consolidate sensor signals from a data bus. Alternately, it might represent the software interface to a smart sensor (e.g. IEEE

1451 compliant sensor). The sensor module is a server of calibrated digitized sensor data records.

Layer 2 – Signal Processing: The signal processing module acquires input data from sensor modules or from other signal processing modules and performs single and multi-channel signal transformations and CBM feature extraction. The outputs of the signal processing layer include: digitally filtered sensor data, frequency spectra, virtual sensor signals, and CBM features.

Layer 3 – Condition Monitor: The condition monitor acquires input data from sensor modules, signal processing modules, and from other condition monitors. The primary function of the condition monitor is to compare CBM features against expected values or operational limits and output enumerated condition indicators (e.g. level low, level normal, level high, etc). The condition monitor also generates alerts based on defined operational limits. When appropriate data is available, the condition monitor may generate assessments of operational context (current operational state or operational environment). Context assessments are treated, and output, as condition indicators. The condition monitor may schedule the reporting of the sensor, signal processing, or other condition monitors based on condition or context indicators, in this role it acts as a test coordinator. The condition monitor also archives data from the Signal Processing and Sensor Modules which may be required for downstream processing.

Layer 4 – Health Assessment: The health assessment layer acquires input data from condition monitors or from other health assessment modules. The primary function of the health assessment layer is to determine if the health of a monitored system, subsystem, or piece of equipment is degraded. If the health is degraded, the health assessment layer may generate a diagnostic record which proposes one or more possible fault conditions with an associated confidence. The health assessment module should take into account trends in the health history, operational status and loading, and the maintenance history. The health assessment module should maintain its own archive of required historical data.

Layer 5 – Prognostics: Depending on the modeling approach that is used for prognostics, the prognostic layer may need to acquire data from any of the lower layers within the architecture. The primary function of the prognostic layer is to project the current health state of equipment into the future taking into account estimates of future usage profiles. The prognostics layer may report health status at a future time, or may estimate the remaining useful life (RUL) of an asset given its projected usage profile. Assessments of future health or RUL may have an associated diagnosis of the projected fault condition. The prognostic module should maintain its own archive of required historical data.

Layer 6 – Decision Support: The decision support module acquires data primarily from the health assessment and prognostics layers. The primary function of the decision support module is to provide recommended actions and alternatives and the implications of each recommended action. Recommendations include maintenance action schedules, modifying the operational configuration of equipment in order to accomplish mission objectives, or modifying mission profiles to allow mission completion. The decision support module needs to take into account operational history (including usage and

maintenance), current and future mission profiles, high level unit objectives, and resource constraints.

Layer 7 – Presentation: The presentation layer may access data from any of the other layers within the architecture. Typically high level status (health assessments, prognostic assessments, or decision support recommendations) and alerts would be displayed, with the ability to drill down when anomalies are reported. In many cases the presentation layer will have multiple layers of access depending on the information needs of the user. It may also be implemented as an integrated user interface which takes into account information needs of the users other than CBM.

After defining the layers in the architecture, a decision was made to focus the standard on the specification of interfaces for layers 1 through 5. Although a general set of interfaces may be conceptually described for the decision support layer, the structure of the information that it serves is tied to the specific requirements of the targeted application to a degree that standardization of the interface is not feasible. Since the presentation layer acts only as a client in the c-s communication model there are no server interfaces that are required to be defined.

The components of the OSA/CBM architecture are shown in Figure 2. The primary inputs to the architecture definition are the functional description of the layers (as discussed above) and the MIMOSA CRIS, along with the general requirements described in the section on CBM Architecture. An object oriented data model has been defined (using Unified Modeling Language – UML - syntax) based upon a mapping of the MIMOSA relational schema to the OSA/CBM layers. For a given layer of the

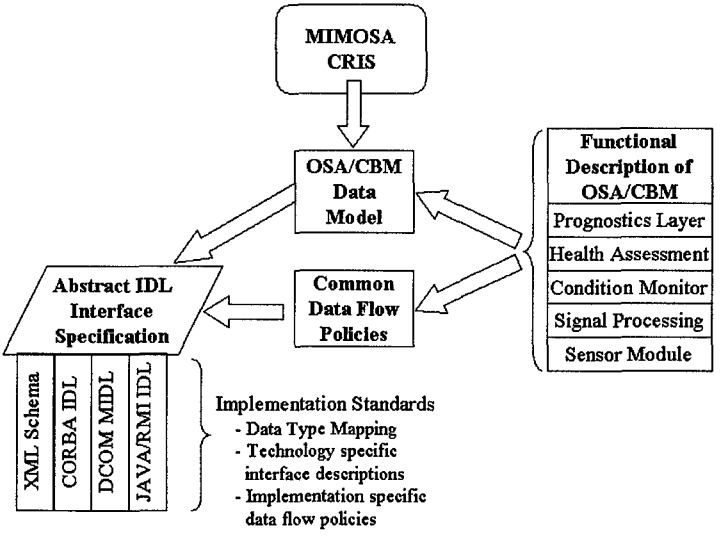


Figure 2: Outline of the OSA/CBM Architecture

architecture, the data model does not describe all of the object classes that would be required for a software implementation. The focus is on describing the structure of the information that might be of interest to clients of that layer. In fact, in the same way that the MIMOSA interface standard does not impose a structure on the components that comprise a MIMOSA compliant system, OSA/CBM does not impose any requirements on the internal structure of compliant software modules. The architectural constraints are applied to the structure of the public interface and to the behavior of the modules. This approach allows complete encapsulation of proprietary algorithms and software design approaches within the software module.

The common data flow policies were defined to standardize the approach to time-based and event-based messaging. In general, the information consumer (at the higher layer of the architecture) will initiate requests for information (a pull rather than a push messaging model). Requests for static information, information which is expected to be available upon request, (e.g. configuration information) will use synchronous messaging protocols. Requests for dynamic information (new sensor data, or updated processing results) will use asynchronous messaging protocols. The asynchronous messaging model requires that the supplier implement a "request()" interface and the consumer a corresponding "notify()" interface.

This messaging model defined above supports time-based or event-based pull of information but does not support event-based push from the information supplier. Event-based push may be implemented by the information supplier acting as a client and utilizing the information consumer's "notify()" interface to initiate communication. Implementation of this messaging model also requires that the supplier can associate a set of information consumers to a particular triggering event.

At the time of implementation, further design decisions may be necessary with respect to data passing. An information supplier may pass actual data elements, or alternately may pass remote references which the information consumer may use to "callback" to get the actual data. There are pros and cons to each approach, but a complete discussion is beyond the current scope. The further detail of the messaging approach will be called out as part of the implementation specific data flow policies.

For CORBA, DCOM and JAVA-RMI implementations, the software interfaces are formally described using their own specific Interface Definition Languages (IDL). In order to commonize the various implementations to the greatest extent possible, the architecture utilizes a common abstract IDL specification (the abstract IDL will be specified using the CORBA IDL syntax). The abstract IDL describes a set of common data structures and interface configurations. The OSA/CBM standard will include mappings to several of the implementation technologies listed in figure 2 (XML and DCOM implementation specifications are currently being developed). Although the abstract specifications provide a strong base of software commonality, software interoperability would not be possible without the detailed implementation specifications. Software applications implemented using different technologies will not be interoperable. In order to have interoperability between applications built in CORBA and DCOM, for instance, a software interface (typically called a bridge) would be required to implement a a mapping. The OSA/CBM architecture still provides strong benefits in this case. The efforts involved in designing and building the bridge will be significantly reduced due to

the common core design requirements and the existence of the associated implementation standards. This view of the architecture also illustrates its extensibility to new or evolving middleware technologies; the core architecture may be mapped to an implementation in the new technology. Since the core software architecture has not changed, existing applications should be able to be readily ported to the new middleware.

Current Status: The architecture that has been discussed is being applied in demonstrations across a variety of different CBM applications as a part of the DUST program. The demonstrations will cover several of the technologies which were discussed, and associated implementation standards will be developed. Lessons learned during the implementation process will be used to update and improve the core architecture. The architecture is also being evaluated for transition by both ARMY and NAVY programs. The ARMY is evaluating the use of the MIMOSA and OSA/CBM standards as a part of their maintenance architecture. Other NAVY research programs in the area of CBM are being directed to consider the OSA/CBM architecture in their system and component designs [9].

Acknowledgment: This work was supported by the Office of Naval Research through the *OSA-CBM Boeing DUST* (Grant Number N00014-00-1-0155). The content of the information does not necessarily reflect the position or policy of the Government, and no official endorsement should be inferred.

References:

[1] Thomesse, J.P., and Leon Chavez, M. "Main Paradigms as a Basis for Current Fieldbus Concepts," *Fieldbus Technology, Systems Integration, Networking and Engineering*. Dietrich, D., Neumann, P., and Schweinzer, H., eds. Vienna: Springer-Verlag, 1999.

[2] Serain, D. *Middleware*. London: Springer-Verlag, 1999.

[3] Object Management Group (OMG) Website, http://www.omg.org/

[4] Microsoft. "COM: Delivering on the Promises of Component Technology." http://www.microsoft.com/com/default.asp

[5] Sun Microsystems. "Introduction to the J2EE Architecture." http://developer.java.sun.com/developer/technicalArticles/J2EE/Intro/index.html.

[6] Sun Microsystems. "Remote Method Invocation over IIOP." http://java.sun.com/products/rmi-iiop.

[7] W3C Note 08 May 2000, "Simple Object Access Protocol (SOAP) 1.1." http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[8] MIMOSA Website, http://www.mimosa.org/

[9] Roemer, M. J., et. al, "Prognostic Enhancements to Naval Condition-Based Maintenance Systems," *Improving Productivity Through Applications of Condition Monitoring*, 55th Meeting of the Society for Machinery Failure Prevention Technology, April, 2001.