

Arbor: An Open System Architecture and Data Model for Integrated Systems Health Management

James Branigan

Band XI International, LLC
Creedmoor, NC

John Cunningham

Band XI International, LLC
West Hartford, CT

Patrick Dempsey, Brett Hackleman, and Paul VanderLei

Band XI International, LLC
Bracey, VA, Scottsdale, AZ, and Grand Rapids, MI

Edited and Reviewed by

Matthew Skalny, Kenneth Fischer, and James Bechtel

US Army TARDEC
Warren, MI

ABSTRACT

Arbor is a code name for an open data model and system architecture for solving an interesting set of problems that must handle large quantities of time variant information. Condition Based Maintenance (CBM) is a problem area that the Arbor system was designed to address. We present the reader with a CBM scenario that is implemented on Arbor. Using this scenario as an example, we demonstrate how an Arbor based system provides: (1) reduced development cycle times, (2) shortened maintenance and redeployment cycle times, (3) lower total costs, (4) higher quality, (5) increased operational efficiency, and (6) tactical advantage. We then describe the relationship between Arbor and standards efforts, including those representative of the CBM space. Finally, we mention specific problem areas where Arbor based systems have been deployed and operated at a high technical readiness level.

INTRODUCTION

Many industries have recognized the importance of cycle time in determining the difference between success and failure. In the domain of Integrated Systems Health Maintenance (ISHM), the critical cycles are the product development and product maintenance cycles. ISHM, as a critical part of a military logistics, needs to improve its cycle time in order to provide cost and tactical advantages. While the cost advantages of a faster cycle time are simple to calculate, the tactical advantage provided by a more agile logistics organization should not be underestimated. In combat, logistics organizations are often the deciding factor in achieving “Moral” victory over insurgent groups [1].

Famous fighter pilot and military strategist, Col. John Boyd, USAF, described the cycle that an individual fighter pilot goes through during battle in his now-famous *Observer-Orient-Decide-Act (OODA) loop* [2]. He later extended this theory to demonstrate the importance of a fast OODA loop in battles throughout the history of mankind. A simplification of his theory states that if you can traverse your OODA loop faster than your adversary, then you can

gain an advantage. Repeated, faster traversals of the OODA loop lead to victory.

Toyota has long understood the importance of continuous improvement, referred to as *Kaizen* [3], and shortening cycle time, which they developed into their own product development and manufacturing process, commonly cited as *Lean Manufacturing* [4]. This system of continuous improvement enabled Toyota to grow to be the largest, highest quality, and most dominant auto manufacturer in the world.

In the field of software development, *Agile* software development practices have supplanted traditional *Waterfall* based development practices [5], with one of the more effective collections of agile software development processes specifically called *Lean Software Development* [6]. One of the fundamental differences between *Agile* and *Waterfall* is the number of times the project team cycles through the delivery loop during a project. *Agile* methods make many trips around, known as iterations, while

Arbor: An Open System Architecture and Data Model for Integrated Systems Health Management, Branigan, et al.

Waterfall processes make one linear pass. *Extreme Agile* teams work in one-week iterations, with the habit of building and deploying working systems on a weekly basis. These practices enforce rigorous quality disciplines, such as test driven development, and enhance stability by always demanding the existence of a current working system. [7]

The goal of Arbor is to provide shorter, faster cycle times for product development and product maintenance in the domain of Condition Based Maintenance (CBM). Arbor encompasses the full system architecture and data model, which is described below in more detail.

ARBOR'S TARGET PROBLEM SPACE

The embedded computing domain includes a large and varied class of problems that on closer examination display very similar characteristics. Applications that successfully solve this class of problems address the following challenges:

- Acquire data from a variety of sensors
- Analyze collected sensor data
- Convert raw data into useful information
- Display information to a user/operator using both quantitative and qualitative representations
- Record user/operator actions and patterns of interaction
- Store the sensor and user data for later use
- Detect the presence of peers
- Communicate information to interested peers
- Acquire and integrate information from peers
- Dynamically and remotely install, uninstall and manage applications and application features
- Communicate information to remote systems

AN ADVANCED CBM SCENARIO USING ARBOR

In-Vehicle CBM applications certainly exhibit the characteristics described above. To help understand the capabilities of Arbor in more depth, we present a CBM use case scenario, as originally prescribed by Jacobson [8]. A collection of *Use Cases* represents the set of functional system objectives that comprise the system's requirements model only. It is important to keep this in mind as the Use Cases are developed. These should be limited to capture only the functional requirements. Too often, teams allow analysis, design, and implementation to creep into the Use Cases. This can cause a team to lose focus and be distracted from the goal of capturing the functional scope and setting boundaries.

The *Use Case* models what users should be able to do with the system. Each Use Case is a complete course of events, seen from a user's perspective. The model requires **Actors**

to represent the roles of the system users and other systems. Therefore, these Actors can model anything that needs to exchange information with the system. Each Use Case can therefore be viewed as a complete course of events, or transactions, between the system and some set of Actors.

For development of effective systems, we must also capture contextual information. We do this by elaborating basic Use Cases by identifying necessary *pre-* and *post-*condition requirements for proper definition of the Use Case. Also, when declaring a course of events that comprise a Use Case, we often make certain *Assumptions* regarding the context, system behavior, actor behavior, or resource availability. Likewise, when defining Use Cases, discussions will uncover certain unresolved issues and potential risks. This too should be captured and documented.

The conceptual level should remain at the appropriate level for interaction with the Actors. Use Cases capture the *what* of the system being specified, and leave the *how* to later analysis and design phases. Also, it is important to first capture the basic courses of the Use Cases; those that assume best case scenarios. Variants of these basic courses, elaborating on what is done when errors or exceptions are encountered, should be captured as alternative courses. Normally, a Use Case has just one basic course, but several alternate courses. Early effort should focus on the basic courses.

Actors

- Service Technician
- Algorithm Developer

Systems

- In-Vehicle Computing System
- Handheld diagnostic scan tool
- Server Data Collection and Analysis System

Pre-Conditions

- An Arbor system is deployed into the subject vehicle
- A server is available to collect diagnostic information from subject vehicles
- The subject vehicle periodically reports vehicle sensor values to the diagnostic server

Post-Conditions

- Vehicle performance data has been collected and stored in a master database
- Vehicle has a new diagnostic/prognostic algorithm and related instrument cluster available to the in-vehicle crew

Scenario

1. The subject vehicle experiences an unexpected failure
2. The service technician manually diagnoses and repairs the problem
3. The service technician records the diagnosis in the handheld scan tool
4. The information concerning the vehicle problem is uploaded to the server from the scan tool.
5. The collected sensor data that preceded the vehicle failure and the service technician's analysis are collected into a failure report.
6. The failure report is made available to a failure analysis engineering and algorithm development team.
7. The team determines the key sensor parameters that are useful in predicting the specific failure and formulate a predictive algorithm.
8. The algorithm developer creates an algorithm that can run on the vehicle to predict the failure and notify the operator of imminent failure before the event occurs, providing quantitative information predicting time to failure or qualitative information indicating relative health of the component or overall vehicle
9. The algorithm is fully back tested and verified for correctness against previously captured and representative diagnostic datasets and test cell systems.
10. The algorithm is packaged for deployment to vehicles.
11. The service technician uses the scan tool to deploy the new algorithm to the subject vehicle
12. The subject vehicle operators will now see a new gauge on the dynamic instrument cluster displaying the output of the algorithm. The new gauge will alert them of pending problems, for the previously unknown failure case.
13. Repeat.

HOW DOES ARBOR HELP IMPROVE CYCLE TIME, THUS REDUCING COSTS, IMPROVING OPERATIONAL EFFICIENCY?

The scenario outlined above allows logisticians and vehicle service personnel to schedule maintenance and distribute replacement parts more efficiently. The objective of improved maintenance of the vehicle fleet is fewer breakdowns while conducting operations. Fewer breakdowns have a multitude of benefits, most importantly

reducing the number of targets of opportunity available for our adversaries to engage our forces.

Step 13: Repeat represents the critical step in the scenario. By continuously reviewing the new failure information from the field and developing algorithms to predict potential failures, a system can quickly mature to the point where nearly all failures will be predicted and impending problems can be remedied before the equipment fails. This continuous cycle is critical to the long-term success of CBM.

In the event that a failure cannot be predicted from the available sensor data, the recorded sensor values and service tech report can be used in test cells to develop new sensors which would be able to detect the pending failure. These sensors can be fielded when ready and the software to support them can be installed dynamically.

In this way, we not only have a cycle for continuous improvement of software, but also a cycle for continuous development and deployment of new diagnostic sensors. The approach effectively extends the longevity of the vehicles and enables fleets to run closer to peak efficiency.

HOW DOES ARBOR HELP IMPROVE CBM APPLICATION DEVELOPMENT CYCLE TIME?

Several key capabilities offered by Arbor improve CBM application development cycle times:

- Remotely deploy algorithms and applications
- Standardized access to the sensor inputs
- Standardized form of the algorithm outputs
- Transmitting sensor data to peer systems (i.e., Lead Vehicles, Scan Tools, Data Servers)

Arbor systems make use of open source provisioning technologies from the Eclipse Foundation [9], that allow for remote management and deployment of full new applications, incremental updates, fixes, or configuration data. This technology makes the critical path item for the CBM application cycle the development and testing of the application, as opposed to the time between major vehicle systems overhauls.

It is critical to standardize the input and output formats in order to ensure that algorithms from various vendors can inter-operate. Arbor provides a programming language neutral format for sensor input and algorithm outputs. These values are accessed through a HyperText Transfer Protocol (HTTP) [10] interface, thus ensuring that algorithms can be developed in any programming language. This approach enables the services to adopt the best of breed from a wide variety of providers and avoid a single vendor integration bottleneck.

Transmitting sensor data to peer systems is also vitally important to CBM applications. By using the HTTP

interface and an open input and output format, we enable peer systems to have access to the data, regardless of which technologies or vendors are employed to develop the peer systems.

ARBOR AND INDUSTRY STANDARDS

Arbor makes heavy use of the HTTP specification from the Internet Engineering Task Force (IETF). Arbor adopts the Representational State Transfer (REST) style of architecture in its application of HTTP, as described by Dr. Roy Fielding in his doctoral thesis [11]. Arbor also makes use of the Resource Description Framework (RDF) specifications [12,13] from the World Wide Web Consortium (W3C), a model often referred to as the Semantic Web. Arbor uses the Web Ontology Language (OWL) [14] to describe its data model.

The approach to data modeling taken in Arbor is similar to the approach taken by ISO-15926, *Industrial automation systems and integration—Integration of life-cycle data for process plants including oil and gas production facilities* [15]. While ISO-15926 is not directly applicable to vehicle health systems, the application of RDF to represent the ISO-15926 data models demonstrates that the approach to data model design employed by Arbor is not a new, untested invention.

Standards based efforts in the CBM space, such as Mimosa [16] and OSA-CBM [17] focused on a subset of the problem characteristics targeted by Arbor. For those with existing implementations based on these standards, it is possible to easily participate in an Arbor based solution. In many cases, Arbor can provide additional benefits, such as remote install, uninstall, and management of existing applications, with no changes to existing implementations. In other cases, a small amount of adapter code will need to be written to provide OSA-CBM compatible eXtensible Markup Language (XML) files from the data stored in an Arbor system.

HOW “REAL” IS ARBOR?

Today, Arbor based systems have been deployed commercially in the mining and construction industries, representing Technology Readiness Level 8/9. Arbor based systems were deployed for use by National Guard Civil Support teams for providing perimeter security and situational awareness and sensor synthesis of Chemical, Biological, Radiological, Nuclear, and Explosive (CBRNE) data at Superbowl XL in Detroit and several NASCAR events in Michigan for perimeter threat detection. Band XI is currently engaged in a Phase III SBIR, applying Arbor to the area of CBM with RDECOM.

SUMMARY

The ability to dynamically deploy new applications and diagnostic algorithms provides Arbor based systems with a competitive advantage over other systems providing similar base level capabilities. New functionality can be developed and deployed on a cycle time dictated by that particular function, as opposed to aligning to the cycle of a major vehicle system overhaul. Patches and bug fixes can be remotely deployed, allowing the systems to benefit from continuous improvement, improving user satisfaction and eliminating common causes of operator error. All of this is achieved through the use of open interfaces and open data models. This approach could facilitate the development of a multi-vendor, multi-programming language ecosystem in the area of CBM. By bringing faster cycle times to the CBM community, Arbor enables cost savings, shortened maintenance cycles, improved operational efficiency as well as tactical advantages to the military customers of these CBM systems.

SPONSORSHIP

The work described in this paper was funded under a Small Business Innovation Research grant and funding from RDECOM/TARDEC under contract W56HZV-07-C-0525. Matt Skalny is the Contracting Officer’s Representative and James Bechtel and Kenneth Fischer are Technical Points of Contact.

REFERENCES

- [1] The U.S. Army/Marine Corps Counterinsurgency Field Manual Counterinsurgency, Department of the Army, Field Manual 3-24, December 2006.
<http://www.fas.org/irp/doddir/army/fm3-24.pdf>
- [2] Boyd, John, Col, USAF. *OODA Loop*.
http://en.wikipedia.org/wiki/OODA_Loop
- [3] Imai, Masaaki, Kaizen: The Key to Japan’s Competitive Success, McGraw Hill, 1986.
- [4] Kennedy, Michael, Product Development for the Lean Enterprise, Oklea Press, 2003.
- [5] Royce, Winston, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, 1970.
<http://www.cs.umd.edu/class/spring2003/cmsc838p/Processes/waterfall.pdf>
- [6] Poppendieck, Mary and Tom Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.
- [7] Martin, Robert C., Agile Software Development, Principles, Patterns, and Practices, Prentice Hall, 2002.

-
- [8] Jacobson, Ivar, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
 - [9] <http://www.eclipse.org>
 - [10] Fielding, R., et al *Hypertext Transfer Protocol HTTP/1.1: Request for Comment 2616*, The Internet Engineering Task Force: Network Working Group, 1999. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
 - [11] Fielding, R., *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation at The University of California at Irvine, 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
 - [12] Klyne, Graham and Jeremy Carroll, Resource Description Framework (RDF): Concepts and Abstract Syntax, World Wide Web Consortium (W3C), 2004.
 - [13] Manola, Frank and Erica Miller, Editors, RDF Primer, World Wide Web Consortium (W3C), 2004.
 - [14] Smith, Michael, Chris Welty, and Deborah McGuinness, OWL: Web Ontology Language Guide, World Wide Web Consortium (W3C), 2004.
 - [15] *ISO-15926 Integration of life-cycle data for process plants including oil and gas production facilities*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=29557
 - [16] *Mimosa* – <http://www.mimosa.org>
 - [17] *Open Systems Architecture for Condition-Based Maintenance (OSA-CBM)*. <http://www.mimosa.org/downloads/39/index.aspx>