An Open Architecture Model of System Development

Sushil Birla
University of Michigan

Harry Egdorf
Los Alamos National Laboratory

Richard E. Igou
Y12 and Oak Ridge National Laboratory

John L. Michaloski
National Institute of Standards and Technology

Douglas J. Sweeney **George Weinert**
Lawrence Livermore National Laboratory

David Uchida **C. Jerry Yen**
General Motors

Abstract

Application of Open Architecture (OA) technology is a fledgling practice within the controller industry. OA solutions offer great potential and flexibility in the design and implementation of controllers. Yet, as an evolving technology, OA controller development offers unique challenges compared with traditional project development. This paper presents an OA Life Cycle developed by the TEAM/ICLP API workgroup to assist its specification efforts. A high-level view of the Life Cycle will study the differences between a traditional and an OA Life Cycle model. A detailed Life Cycle model will present a practical treatment to the OA development activities and present the type of information required. The detailed OA Life Cycle model will study the development process from the perspective of the end user, the system integrator, and the component developer.

1 BACKGROUND

Open architecture (OA) technology offers great potential for improving tomorrow's controllers [OMA94, Bab96]. With an open architecture, one can use multi-vendor, plug-compatible components to build a controller. The desire for open-architecture controller components is high, but vendors are slow to respond. One reason for the delay in industry action is that no clear open-architecture solution has evolved. In an effort to promote open architecture control solutions, the Technologies Enabling Agile Manufacturing (TEAM) program for Intelligent Closed Loop Processing (ICLP) sponsors an Application Programming Interface (API) workgroup. The goal of the TEAM API workgroup is to specify standard APIs for a set of controller components. Such an API specification is a daunting task, especially since previous industry attempts have not been fruitful. Currently, other American and international efforts are also attempting to define standard Open Architecture API [OSA96, OAS96, OSE96]. The TEAM/ICLP API workgroup has established ties with many of these efforts.

The TEAM/ICLP API workgroup specifies API's using the following Object Oriented methodology concepts:

- class definitions that contain the data and methods and offer encapsulation (hide implementation from specification (API)),
- notion of inheritance to use base and derived classes to add specialization,
- foundation classes and aggregating components to construct "building blocks."

The foundation and aggregation classes form part of OA solution hierarchy:

1. **Foundation Classes**
2. **Components** built from classes
3. **Integration Architecture**
4. **Applications** or controllers built from components

A good design analogy would be hardware component technology:

1. Foundation: hardware chips, RAM/ROM, CPUS, logic.
2. Components: Memory bank constructed from memory and logic chips.
3. Integration Architecture: A hardware board (with data and memory addressing, control logic and ground wiring, as well as CPU/FPU/Clock, Memory, Control Logic, Bus Logic) is an integration architecture.
4. Applications: computer board, memory board, device controller.

Deciphering component requirements can be confusing because of a great deal of overlap between layers within the OA solution hierarchy. Adding the traditional life cycle concepts increases the confusion. This paper presents a Life Cycle influenced by Open Architecture issues to help understand the role of OA component technology and its relationship to the overall development process.

This paper is organized as follows. The next section presents a high-level overview of the controller Life Cycle process with related commentary concerning the effects of OA. The third section gives a practical treatment to describe the detailed steps within OA Life Cycle from the perspective of the major contributors – end user, system integrator, and component developer. The final section charts the progress of the TEAM/ICLP API workgroup at standardizing elements within the OA Life Cycle.

2 HIGH LEVEL OVERVIEW

The high-level open architecture development activities can be characterized by the sequence of steps shown in Figure 1. The application development applies the traditional methods of analysis and design. However, the Figure pointedly illustrates the concept of deriving two OA implementations based on different design and implementation specifications. The actual Commercial Off-the-Shelf (COTS)[1] component set is a subset of the total available component sets. Overall, Figure 1 describes the following development stages.

Domain Model analyzes the set of all solutions (i.e., domain) for an application and then bounds the problem to the essential characteristics.

Requirement Specification specifies system requirements that include functionality, capabilities, performance, interface requirements or inputs and outputs, user interface, design requirements, and development standards.

System Design Specification describes system elements that include integration architecture, interfaces between components, resources available, resource dependencies.

Detailed Design describes component elements that include functional operations, data definitions, input output behavior, performance, control flow, logic.

Implementation is the process of translating the design into hardware and software components that would be followed by testing, verification, and validation.

[1]End-user developed components will be considered inclusive within COTS.

A Life Cycle models the period of time that starts when a system is conceived and ends when a system is no longer available for use. The stages of development activity can be modeled by any number of Life Cycle models. Figure 1 most closely resembles the traditional "waterfall" life-cycle with its top-down approach to system development. One can easily apply more sophisticated life cycle procedures. The "spiral" life-cycle methodology proposes a more risk-oriented development process in terms of a diverging spiral, which iterates through essentially the same sequence of steps, but in increasing degree of elaboration [Boe88]. The Legacy- and Reuse-Based life cycle offers guidance to incorporating legacy technology and augments top-down synthesis with bottom-up domain re-engineering to archive software reusability [AP95].

Figure 1 outlines "canonical" life cycle stages – but this is not the primary concern of this paper. Instead, this paper focuses on the quantifier information required as the OA development progresses from a *component-orientation* to a *systems-orientation*. Such quantifier information includes the assignment of component default parameters, definition of the necessary system resources, and resolution of the interdependencies of components. Timing with the OA Life Cycle for satisfying an information quantifier is a gray area, subject to numerous interpretations because one could assign application parameters at several points along the development cycle. For example, assuming a practical perspective, one could assign a parametric quantifier – such as number of axes – at either the "make," "compile," "link," or ".ini file" phase. With multiple vendors, explicit instructions are necessary at one or more phases of parametric quantification if one desires conformance.

Figure 1 also incorporates two additional Life Cycle notions that are important to the OA development process - the **Implementation Environment and Tool Specification** and **COTS components**. The **Implementation Environment and Tool Specification** has a profound impact on the OA Life Cycle marketplace. The choice of computing platforms and control equipment, as specified by the marketplace, and user demand dictates available OA COTS components. To exercise the benefits of OA development, a sufficient set of COTS components for the selected **Implementation Environment and Tool Specification**. By direct implication, the market share will also effect the viability of particular **Implementation Environment and Tool Specification** must exist.

Finally **COTS Component** technology itself must offer special features. Component *flexibility* provides default or minimal functionality where the user has not selected any. Component *scalability* allows convenient methods of experimentation and quick reconfiguration of components. It also allows for the capture of results in order to analyze the ex-

Figure 1: High Level OA Life Cycle with COTS Components

perimentation. Component *modularity* allows controls users and system integrators to purchase and replace components of the controller without adversely affecting the rest of the controller. Component *extensibility* allows advanced users and third parties to incrementally add functionality to a component without replacing it completely. Component *portability* allows components to run across platforms.

3 DETAILED OVERVIEW

A unique aspect to the OA Life Cycle is the differing contributor's perspective within the development process. To achieve OA success, *component developers* must build their components to a standard specification and integration architecture. From an *end user* perspective, the application requirements dictate the selection of the tools and resources such as hardware, control devices, and computing platforms. Included within this perspective is the *platform* which supplies system low-level services (e.g., file-management, job control.) The *system integrator* picks software components that match the application behavior and functional requirements. The system integrator configures the components to match the application specification, connects the selected components using a integration architecture and verifies the system operation. The system integrator also checks compliance of components to validate the user-specification of performance and timing requirements.

Figure 2 shows the OA Life Cycle that categorizes the major contributors into three categories and partitions the OA Life Cycle activities by contributor. The three contributor categories are:

- **component developer**
- **system integrator**
- **end user**

Acceding to the desires of TEAM/ICLP community, Figure 2 uses the term *module* instead of the term "component" to place an extra emphasis on the goal of modularity and "plug-and-play." After this the term "module" and "component" will be used interchangeably.

3.1 Component Developers' Tasks

Control vendors provide component products and support for hardware or software components. For control vendors to conform to an open architecture specification, they need to conform to the following specifications:

System Service Specification requires a mechanism similar to the NGC profile [SOS94] to describe the system service specification defining such areas as platform capability, I/O control devices, and support

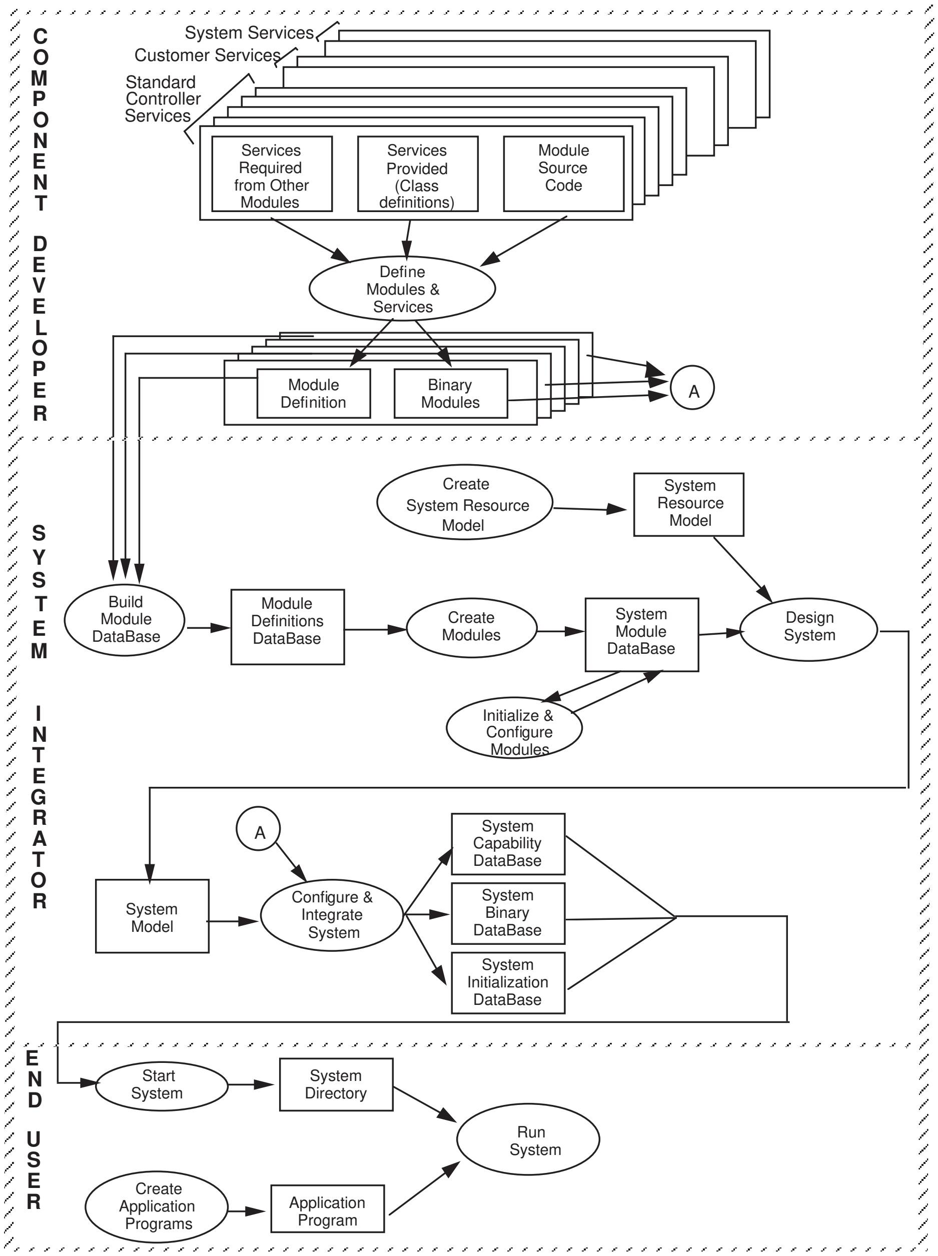

Figure 2: Detailed OA Life Cycle by Participant Perspective

software. The system services describe the platform and infrastructure support (such as communication mechanisms) and the available resources (disks, extra memory, etc.) Computer boards in turn have a device profile that includes CPU type and the CPU performance characteristics. Included within the profile is the operating system support for the CPU. Sensor-effector devices, such as controller cards or drives, subscribe to a general electro-mechanical classification and then provide a more detailed capability feature profile.

Module Class Specification relies on established analysis and design principles [WG94] to derive its specification. In general, the following items would be required:

- class definitions of the module
- services provided by the module

- persistent data in the module that needs to be initialized
- creation information
- services used by other modules
- resources consumed and services required by the module (including memory space, operating system functions, etc.)
- system services supported by the module
- other system services used by the module
- response time benchmarks (e.g., published latency)
- items requiring configuration (interrupt lines, addressing of buffers, etc.)

Customer Specifications describes two aspects – 1) the computing and control resources and 2) the degree of behavioral functionality. The customer provides "initial conditions" such as hardware, control devices, and computing platforms that in general constitute the application resources. The customer also specifies the maximal functional capability and performance expected of a module. The module class definitions and module binaries would then be expected to operate under these computing, control, and behavioral constraints.

3.2 Control System Integrators' Tasks

The System Integrator builds the controller from component parts as specified by the end user based on market demand. Many times the system integrator and the component builder are the same. However, to derive the major long-term benefits of standard open-architectures, it is necessary that the component builder and the system integrator be clearly separated within the Life Cycle.

The control component builder provides binaries (as some form of an object library) from which the system integrator selects components based upon the design criteria (controller performance: cost, accuracy, speed, reliability, tolerances, available parts, etc.). The main responsibilities of the control system integrator are as follows.

Build Module Database to collate module definitions of all selected modules to be included in the system.

Create Modules to select the subset and the number of instances of all modules participating in this controller and assign a name to each instance.

Initialize and Configure Modules to give the initial values to persistent data for each module selected in the system.

Create System Resource Model to collate the control hardware, as well as any software platforms selected.

Design System to produce as output the system model which describes resource and performance assignments. In producing this model, one verifies the module resource consumption against system resource definition; allocates modules to resources; defines interaction of modules via interprocess communication; assigns timing information; creates a prototype implementation and test suite. This process is iterated as necessary.

Configure and Integrate System to execute a sequence of "make file"-like operations - a "compile" operation and "link-like" operation to generate: 1) `System Capability Database` contains processor and memory capabilities for load balancing; node, network and module connection information, 2) `System Binary Database`, and 3) `System Initialization Database` contain start-up information for each module in the system for initialization. It also includes initial values of initialization files of each module and persistent data.

3.3 End Users' Tasks

The end user is responsible for creating application programs to run on the controller. The end user can be expected to handle the start-up and shutdown operations. The end user can test and debug application programs on the controller. Different classes of users can be expected. Some end users can be tasked with program generation, some with maintenance and others with operation. A runtime system configuration registry (i.e., local and global ".ini" files) would be expected to handle the general start-up and shutdown sequencing and specific system customization. The end users' tasks, as illustrated in Figure 2, can be summarized as: start system, create application programs, and run system. Other end users' tasks are beyond the scope of this discussion.

Start System assumes that a functional controller satisfies the initial application requirements. The functional controller is assumed to be integrated into the actual (as opposed to simulated) machine and equipment it will control. The controller is also at a stage where it can be modified, purchased, tested by end users, as well as enabled for integration of sensors, video, CAD packages, data base logging or other COTS component functionality. One need within an OA controller is for a `System Directory` which acts as a global name registry and contains unique identifiers for all names in the controller.

Create Programs expects different classes of end users to produce different types of application programs. The operator may only run part programs with minimal programming capability. An end user may develop and run part programs or develop an integrated program suite. A system administrator may develop diagnostic and maintenance programs.

Run System assumes that the controller has been tested and debugged, and user programs for both discrete and motion applications can be executed. Desired operational sequences are specified within the application programs.

4 DISCUSSION

This paper demonstrated the Life Cycle considerations in an open architecture systems model. A high-level OA Life Cycle view that describes the major life-cycle phases required to build a controller was given. The paper included a detailed OA Life Cycle that focused on the contributor activities and the parametric quantifiers associated with OA controller development.

In a perfect world, one could use an automated tool to build an application controller from COTS components. One would computerize the application requirements that would trigger decision rules to automatically select from a set of COTS components to be assembled into the application controller. Unfortunately, such automated development is not yet a reality.

As a step along the path toward automated OA development, the TEAM/ICLP API is developing specifications, guidelines, and implementation examples for OA controller technology. Foremost, the TEAM/ICLP API workgroup has concentrated on defining class definitions for standard controller modules. Because of the level of effort, explicit quantification of all OA Life Cycle phases has not yet been attempted. The effort has produced Application Programming Interfaces for controller components [TEA96a, TEA96b]. Without other OA Life Cycle factors, the defined API's provide "plug-only" compatibility. However, "plug-only" compatibility is a step in the right direction along the path to automated OA development. The OA Life Cycle steps leave considerable flexibility at the early stages of development (during class and component API selection) so that additional system requirements and performance constraints can help narrow the choices during the progression through the later stages of the OA lifecycle. In conclusion, as Open Architecture technology continues to evolve, the OA Life Cycle will serve as a valuable model to understand the OA development process.

References

[AP95] J.D. Ahrens and Noah S. Prywes. Transition to a Legacy- and Reuse- Based Software Lifecycle. *Computer*, pages 27–36, October 1995.

[Bab96] Michael Babb. PCs: The Foundation of Open Architecture Control Systems. *Control Engineering*, pages 75–78, January 1996.

[Boe88] B.W. Boehm. A Spiral Model of Software Development and Enhancement. *Computer*, pages 61–72, May 1988.

[OAS96] OASYS. U.S. Army and Air Force Sponsored Open Architecture Initiative. Web Address: http://www.interaccess.com/controls/www_oasys/, 1996.

[OMA94] Chrysler, Ford Motor Co. , and General Motors. *Requirements of Open, Modular, Architecture Controllers for Applications in the Automotive Industry*, December 1994. White Paper – Version 1.1.

[OSA96] OSACA. European Open Architecture Effort. Web Address: http://www.isw.uni-stuttgart.de/projekte/osaca/english/osaca.htm, 1996.

[OSE96] OSEC. Japanese Open Architecture Effort. Web Address:http://www.mli.co.jp/OSE, 1996.

[SOS94] National Center for Manufacturing Sciences. *Next Generation Controller (NGC) Specification for an Open System Architecture Standard (SOSAS)*, August 1994. Revision 2.5.

[TEA96a] TEAM. *Technologies Enabling Agile Manufacturing (TEAM) Intelligent Closed Loop Processing (ICLP) Open Architecture Specification: Part 1: General Information*, January 1996. Web Address:http://isd.cme.nist.gov/info/teamapi.

[TEA96b] TEAM. *Technologies Enabling Agile Manufacturing (TEAM) Intelligent Closed Loop Processing (ICLP) Open Architecture Specification: Part 3: Class Definitions*, January 1996.

[WG94] I. White and M. Goldberg. *Using Booche Method - A Rationale Approach.* Benjamin/Cummings Publishing, Redwood City, CA, 1994.