Proceedings of the 14th ASME 2012 International Mechanical Engineering Congress & Exposition
IMECE2012
November 9-15, 2012, Houston, Texas, USA

IMECE2012-87941

TOWARD BETTER INTEGRATION OF VEHICLE ASSEMBLY PRODUCTION SYSTEMS

Jorge Arinez
General Motors
Warren, MI, USA

Jerry Yen
Mitsubishi Electric Automation
Chicago, IL, USA

John Michaloski, Frederick Proctor, William Rippey
National Institute of Standards and Technology
Gaithersburg, MD, USA

ABSTRACT

In today's manufacturing world, system integration often necessitates composing systems of technology that are not designed to interoperate with each other. This inherent incompatibility results in redundant, non–value added work that is required for information to be properly transferred and processed in order for the total system to function properly. As a result, current approaches to systems integration tend to be complicated, costly, time–consuming, and error–prone. In the automotive industry, this integration predicament is found most dramatically in vehicle assembly systems, which are built from a collection of different, incompatible, and multi–vendor "silo" subsystems. This paper investigates the problems associated with integration of vehicle assembly systems and proposes a standard information and communication model to address the integration problems due to incompatible data models. Benefits to the standard information and communication model, including better integration, improvements to the efficiency of the existing vehicle assembly operations, and additional capabilities to increase productivity, is discussed.

NOMENCLATURE

API	Application Programming Interface
CNC	Computer Numerical Control
COM	Microsoft Component Object Model
DA	Data Access
DCOM	Distributed Component Object Model
COTS	Commercial–Off–The–Shelf
ERP	Enterprise Resource Planning
FTP	File Transport Protocol
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
IO	Input/Output
IP	Internet Protocol
IT	Information Technology
MAP	Manufacturing Automation Protocol
MMS	Manufacturing Message Specification
MES	Manufacturing Execution Systems
OLE	Object Linking and Embedding
OPC	OLE for Process Control
PC	Personal Computer
PLC	Programmable Logic Controller
REST	REpresentational State Transfer
SCADA	Supervisory Control and Data Acquisition
SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
WS	Web Services
XML	eXtensible Markup Language
XSD	XML Schema Definition

BACKGROUND

Assembly efficiency and capability is a competitive discriminator in every product manufacturing sector [1]. In theory, advances in robotics, sensors, controls, effectors, and material handling could support the easy deployment and routine production of customized products in a single assembly production line. However, the problems associated with integration, configuration, and programming of assembly systems has hampered the cost–effective use and production efficiency of assembly systems. The problems with assembly systems can be attributed to incompatibilities between assembly subsystems that are not designed to interoperate and communicate with each other. This inherent incompatibility results in redundant, non–value added work that is required for information to be properly transferred and processed in order for the total system to function properly. As a result, current approaches to systems integration tend to be complicated, costly, time–consuming, and error–prone.

To alleviate this situation, manufacturers are targeting the use of standard platforms and communication protocols. The primary goal of standardization is to enable interoperability of diverse systems. Because systems integration takes place vertically and horizontally, so must standardization. Figure 1 shows a typical factory floor relying on communication networks to integrate the manufacturing technology and the enterprise. Horizontal integration requires standards that allow physical equipment, such as motors, sensors, and controllers; and also software, such as process plans and part programming languages; to communicate efficiently. Vertical integration covers the breath of the enterprise that includes factory equipment, plant–level control, and management used in design, process planning, quality control, accounting, forecasting, and other resource planning activities.

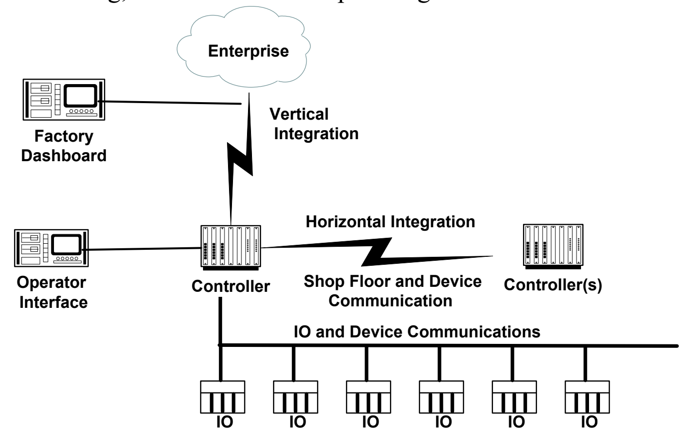

Figure 1 Overview of integration architecture

The available communication standards cover a wide range of capability and cost ranging from high–speed networked I/O subsystems standards to distributed communications standards for integrating all machines on the shop floor into the wider enterprise. Numerous competing communication standards are available to interface controllers to device, shop floor, and IO. Further, application domain communication standards compete with other domain communication standards as well as communications standards from the PC industry. In fact, many PC communications standards have been adapted to the factory floor to provide built–in redundancy and durability, so that all the devices remain connected despite the harsh conditions often found in a factory environment.

For vehicle assembly, horizontal integration and intercommunication of constituent controllers, such as PLC, robot, welder, and conveyor, is the primary technology contributor to the integration impasse, mainly due to the lack of a universal communication standard and common information model in this domain. The focus of this paper is on examining the issues related to smart assembly system integration, specifically intersystem communication, presenting an enhanced integration architecture necessary, and reviewing some potential integration technologies to this difficult problem.

In the next section, the difficulties related to vehicle assembly system integration will be analyzed, including a discussion on data programming and fault handling. We then give an overview of the benefits of an enhanced integration framework that emphasizes a common information model. In particular, the application of the enhanced integration framework to maintenance and troubleshooting is highlighted. Included is a summary of some relevant manufacturing programming and integration technology, and categorizes each technology based on its constituent integration components. We then discuss the issues related to achieving an enhanced integration framework.

PROBLEM ANALYSIS

In this paper, vehicle body assembly will be used to illustrate the various integration problems in vehicle assembly. In body assembly, parts arrive from the stamping facility and are welded together to form the body of the vehicle. There are approximately 4000 welds in the body performed by robots. Robots in the body shop also apply adhesives and sealants at various stages in the assembly process. Generally, the robots have multiple end–effectors (e.g., a welding gun and a gripper to position parts). The grippers and fixtures are relatively adaptable because the components for different vehicles are not identical, and different vehicle types, plus their variants, can be built on the same line at the same time. Because of the complicated nature of vehicle assembly, simulations are often popular in understanding the complexity [2, 3].

Figure 2 shows the typical manufacturing architecture for body welding. There is a master PLC controller which supervises and coordinates the individual pieces of equipment in the manufacturing cell. The master PLC controls the arrival of parts on the inbound conveyor and the subsequent robot transfer to the first welding station. There are individual controllers for the robots and welders. In addition, logic is required for fixture/tooling stations and HMIs (i.e., maintenance/operator display panels) inside the cell. Thus, for each of these controllers, stations, and HMIs, specific ladder logic program routines are stored inside the PLC controller. In addition, each robot and welding controller will have its own local control programs that control the device itself.

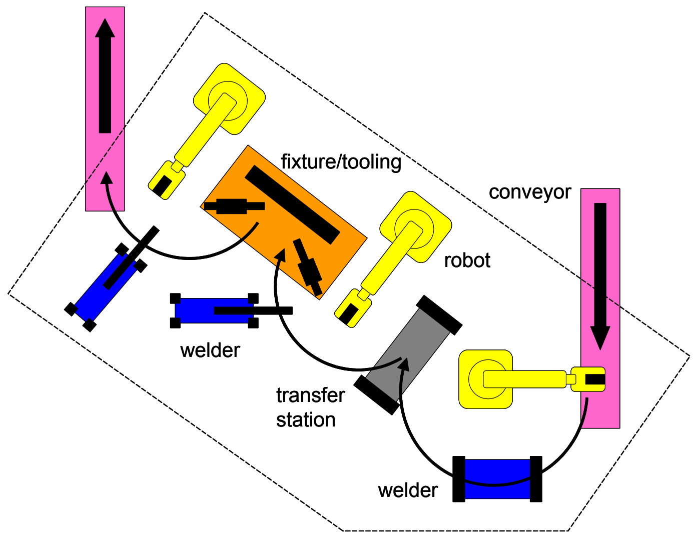

Figure 2 Manufacturing cell for body welding

In manufacturing assembly, there is an emphasis on interworking among controllers, often combining data from several applications to arrive at new information. Examples of inter–transfer of non–real time information between devices and systems in assembly systems include:

- Alarm and event information from plant floor devices/systems to a Production Monitoring system for reporting, analysis, and/or storage;
- Production data (e.g., production counts, equipment status, cycle time information) to plant floor systems for reporting, analysis, and/or storage;
- Configuration and status information of a device that is being transferred from a device (e.g., a robot or weld controller) to another (e.g., a cell controller PLC);
- Maintenance information (e.g., fluid level low, scheduled maintenance required) and requests (e.g., temperature is trending out of control) from an intelligent device/system to a central system for preventative and/or scheduled maintenance actions.

In today's integration landscape, each of the constituent controllers acts as a silo with limited intercommunication abilities, and no common capabilities to assimilate other controllers' information models. Figure 3 illustrates the issues related to the existing integration of plant floor devices and IT applications in a manufacturing environment. The figure shows application silos with the separation of data among the systems and different applications having their own data models. These issues contribute to the complexity of data configuration and lack of common information being transferred.

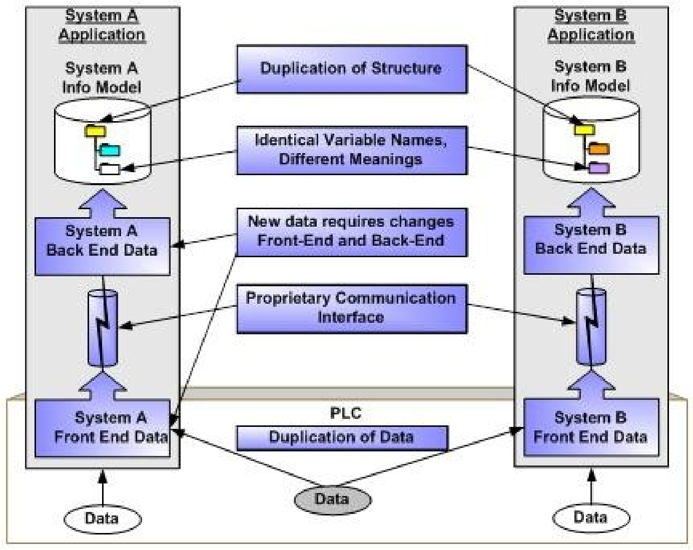

Figure 3 Current plant assembly integration

For the body assembly cell, Figure 4 gives a more detailed description on the current implementation of clients and servers, and in this example, communication is based on OPC technology. OPC is an industry standard with original OPC specifications based on the Microsoft COM/DCOM model [4]. In the OPC architecture, control devices are called OPC Servers. Applications, called OPC Clients, can connect to OPC Servers provided by one or more vendors. OPC offers a number of ways in which OPC clients and servers can communicate, including Data Access, Event and Alarm Management, and Historical Data Access.

The OPC Client can be implemented in and accessed by a process/tooling HMI, upstream by MES, and/or other enterprise applications. The Rockwell OPC Server acts as an Ethernet communication gateway to the underlying PLC. In our terminology, a communication gateway provides one or more application clients access to data of one or more devices. The weld controller and robot controller provide bit–level information to the PLC, and it then transfers the fault/status information, with additional information (such as fault messages associated with the fault codes received) when feasible to the MES/SCADA systems for processing.

Each device may have its own console HMI that can be used to program (or configure) the device during deployment, and display production information and device status during production. It can also be used to help troubleshooting system or device problems. Finally, a historical data base collects real–time shop floor OPC data and is used to help determine the root cause of faults.

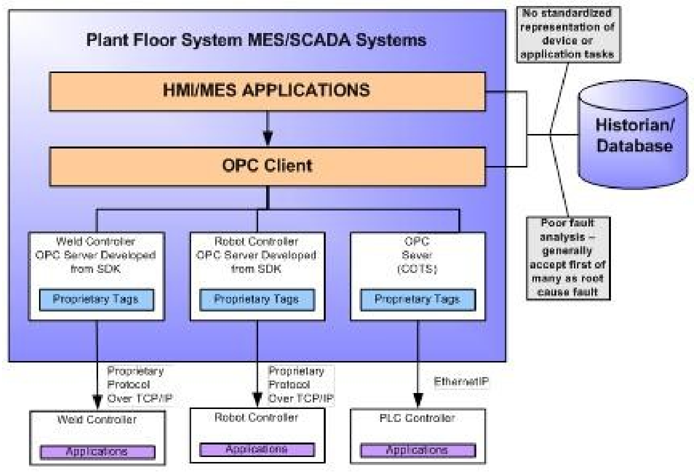
Plant Floor System MES/SCADA Systems
HMI/MES APPLICATIONS
OPC Client
Weld Controller OPC Server Developed from SDK
Robot Controller OPC Server Developed from SDK
OPC Sever (COTS)
Proprietary Tags
Proprietary Tags
Proprietary Tags
Proprietary Protocol Over TCP/IP
Proprietary Protocol Over TCP/IP
EthernetIP
Weld Controller
Robot Controller
PLC Controller
Applications
Applications
Applications
Historian/ Database
No standardized representation of device or application tasks
Poor fault analysis – generally accept first of many as root cause fault

Figure 4 Example of current body assembly integration architecture

From this current vehicle assembly integration architecture, several costly and complicated problems exist, but we will discuss issues related to device data management and fault analysis.

Data Programming

There are two data handling problems in the current vehicle assembly integration architecture: 1) exposing system level data / information, and 2) incompatible data models. The first integration problem is the need to develop a custom application using a Software Development Kit (SDK) provided by the device supplier to gain access to internal system level data inside the device and then expose the data to external systems using OPC DA technologies. This may entail the development of an OPC DA server or application specific programs if the supplier of the device does not provide such OPC DA server commercially. Typically, vehicle assembly workcell implementations do not require the use of SDKs. Instead, procurement of OPC DA servers from the suppliers as commercial–off–the–shelf (COTS) products is the standard procedure.

The OPC DA servers and clients reside in servers of IT systems that are used to gather plant floor data for MES applications. Sometimes, a device SDK must be used to program at the “device” level to access data that is not readily available. For special cases, the equipment supplier offers a SDK with a set of APIs for programming applications to get to the system level data. A device SDK, if used, is used during the system development stage and not in production. One example is the use of SDK to develop an OPC DA server for Fanuc robots. In this case, the Fanuc SDK has the APIs that allow the access to Fanuc internal data through a proprietary protocol. A user then develops the OPC DA server, which calls the APIs to access system level data and makes the data available to the external systems through the OPC DA Server. Application data of a device can be programmed using the programming environment provided by the device such as PLC ladder logic. The data can also be transferred to the upper level systems using the same OPC DA server.

Fault Handling

The incompatible data model is the second integration problem. OPC DA servers are capable of transferring data; however, there is no standard naming and definition of these data points among device suppliers. A complete and standard information model supported by each device and a standard communication mechanism would help overcome this problem.

Fault handling is an explicit example of this issue. For example, opening a safety gate while a robot is running causes a safety fault in the PLC which will cause an immediate stop of the robot, which in turn may cause many additional device faults to occur. Determining the root cause of the fault by unwinding the proliferation of alarms is a programming time sink, as each vendor has their own set of alarms/faults, and to further compound the problem, similarly named faults can mean entirely different things.

Deciphering a fault sequence can be so difficult that instead of even trying to do so, the first fault is often used as the root cause of the fault. This first fault may work most of the time, but when it doesn't, analyzing a fault without a coherent trace could result in erroneous and unpredictable conclusions. The standardization of a vehicle assembly alarm/fault information model is seen as an enabler of fault analysis programming and potentially offering cost savings in reduction of equipment maintenance and increase of production uptime. This standardization would be broken down into device fault models, network communication fault model, and application fault model.

Fault handling integration for welding equipment is another problem. Although welding equipment may be on the Ethernet, it generally only supports FTP for the upload/download of programs and for the backup/archival of welding data to files stored for offline analysis. Real–time access to current welding device data is usually accomplished through device–level networks, such as DeviceNet and EtherNet/IP with limited access to information. Thus, the propagation and logging of error sequences is difficult and incomplete. For example, let us assume there is a fault in the welding controller. Then, the vehicle assembly cell must have the welding cell set an I/O bit on the robot controller to signal a fault, which in turn sets a bit on the PLC informing it of the fault. There is no real–time information collection that correlates the state of the weld, robot, and PLC devices when the fault occurred, as each device is running somewhat independently with occasional synchronization. The key reason why the simple bit interface is chosen instead of propagating error codes is to reduce the coupling with vendor implementations since each vendor chooses to implement error codes differently and may even change from version to version. Further, as this simple bit interface propagates fault bits to upper–level systems, correlation of information from different subsystems is lost.

Again, multiple faults across multiple devices and controllers are difficult to analyze because there is no common model and no correlation of fault messages or types. For example, referring to Figure 4, when a weld controller fault occurs, the details of the fault are contained only inside the weld controller. The robot controller detects the weld controller fault but does not have the details since it only receives a fault bit. However, the robot controller knows its position (for example, the number of weld spot when the weld fault occurs). The PLC receives the weld fault (which translated to a robot process fault) that is passed through by the robot controller but does not have additional details. A proper root cause analysis would require the simultaneous time correlation of faults across multiple devices, which is not easily done from the historical database today.

Configuration of New Vehicle Launch in an Existing Plant

To understand the integration problems, we will look at a common, but potentially involved, "Configuration Use Case" scenario within a vehicle assembly workcell. The Use–Case should offer an end–to–end perspective of the steps involved in configuring (programming, debugging, testing, etc.) a vehicle assembly cell/station.

Our Configuration Use Case covers the circumstances when a new vehicle model is launched in a plant that is already producing other vehicles. We assume a pre–existing production line, and break the Configuration Use Case down into three different configuration scenarios:

1) Some existing equipment that will not require change in order to run the new model,

2) Some equipment that will require modifications to logic, and

3) Completely new equipment that will have been purchased for the production of the new vehicle and added to the production line(s).

In the latter two use cases, there will be configuration required of the end devices (PLCs, robots, weld controllers, etc.) and also on the plant–floor MES system/SCADA system for them to be able to read data from the devices. From a high level perspective, the Configuration Use Case is defined by the engineering steps taken to map, link, and associate a data tag/PLC point into a MES/SCADA system so that the data can be viewed/analyzed/reported on a real–time or historical basis. Figure 5 and Figure 6 are examples showing the ladder logic programs and data structures that may be required to run a production cell as illustrated in Figure 2. Each device has a local set of tags defined in addition to the global tags for the master ladder logic program. Configuration at the controller level involves validating that the tags and the corresponding logic are generating correct values. Also, once a device configuration has taken place at the controller level, additional corresponding configuration must be done at the higher system level of MES applications.

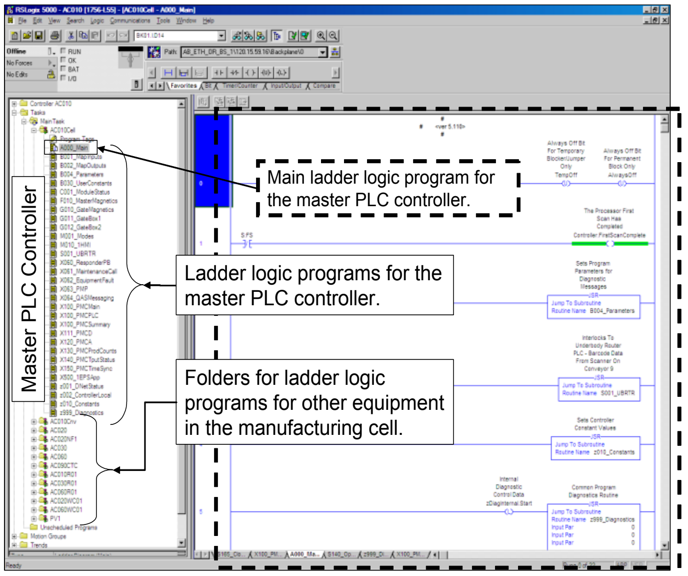
Master PLC Controller
Main ladder logic program for the master PLC controller.
Ladder logic programs for the master PLC controller.
Folders for ladder logic programs for other equipment in the manufacturing cell.

Figure 5 Configuration view of the master PLC controller for the manufacturing cell.

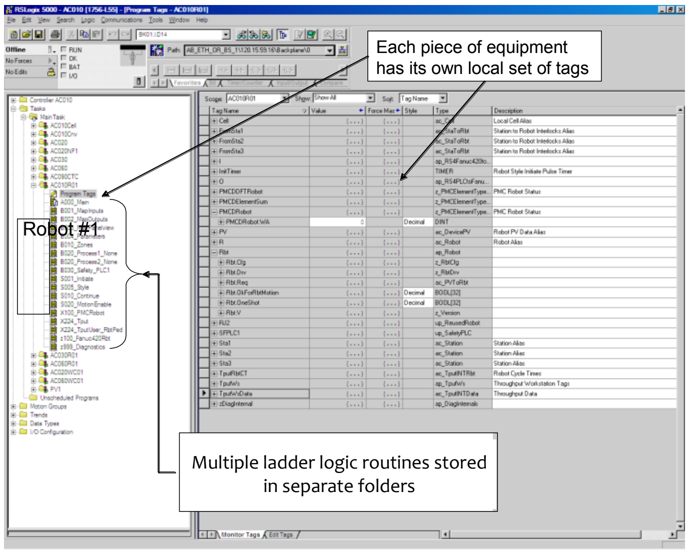
Each piece of equipment has its own local set of tags
Robot #1
Multiple ladder logic routines stored in separate folders

Figure 6 Configuration view of a robot inside the manufacturing cell.

In some cases, the steps to configure a new data point from a plant floor device to the MES or SCADA application are straightforward. A control engineer simply uses the programming editor and looks up all devices on the network and then proceeds to reference the data tags on the device in his logic program. Typically, setting up the MES/SCADA application consists of associating device–level addresses to MES/SCADA projects or programs that will be running so as to perform the monitoring of a manufacturing workcell. This step includes a validation step to ensure that the device is correctly reporting the data to the OPC Server and the corresponding MES/SCADA project logic that makes use of the OPC Server is accurately reporting the data. Furthermore, the data must also be validated from the MES/SCADA program as well; not only must the device be reporting data, the data must be mapped into the MES/SCADA applications correctly as well. Additionally, OPC DA initially can support only primitive data type and not complex data types such as record data. However when an OPC DA complex data specification became available, there were very few implementations by suppliers.

So far we have been simply discussing the data; such as whether a certain bit is on or off, or some other data type is being read and sent correctly. Another important higher level type of data concerns alarms and fault messages. Alarms and fault messages also require configuration and logic verification to ensure that the correct program logic conditions are triggering the correct alarms and faults to the MES/SCADA applications.

The key point of this analysis is to highlight the redundant, non–value added work that is required for the information and data to be properly transferred and processed in order for the total system to function properly. Examples of the non–value added, potentially error–prone work are:

- Configuration of alarm information on the plant floor and also the upper level systems;
- Configuration of data on the plant floor and also the upper level systems;
- Configuration of communication links;
- Translating the information hierarchy from the plant floor to hierarchy in the MES system because of inconsistent information from each device.

ANALYSIS

Improved vehicle assembly integration needs standard device information models as well as a standard communication mechanism. Figure 7 illustrates an Enhanced Integration Architecture embodying a unified state of the plant floor devices and upper layer system integration with the key of having a common data model. Standard information models can improve the process by allowing better programming support for diagnostic status, maintenance, and fault recovery. Standard data and standard communication will also allow for easier configuration programming of the vehicle assembly process to allow leaner manufacturing to reduce waste, create a more robust process, and increase flexibility.

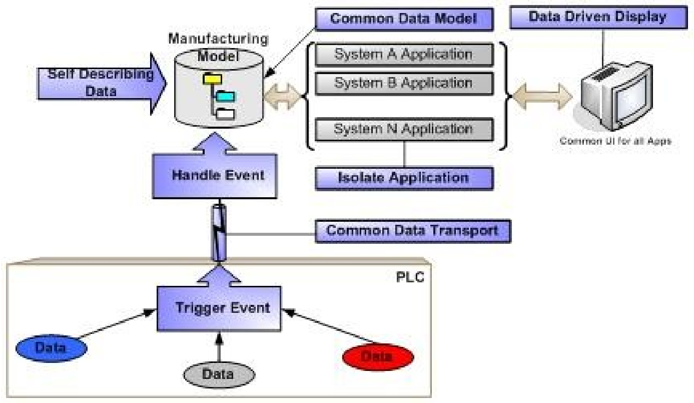

Figure 7 Plant enhanced integration architecture

Using a common unified communication protocol provides the following benefits to end users:

- Reducing the complexity in overall system configuration and debugging during vehicle launch process;
- Reducing the complexity and time to accommodate vehicle assemble process changes;
- Enhancing the capability of the systems to perform maintenance functions that are difficult to accomplish currently, by having standard information models and common ways of making the data/information available without additional development work;
- Gathering more meaningful fault messages for diagnostic and process improvements.

The Enhanced Integration Architecture utilizes common communication protocols and mechanisms, such as XML and Web–based services, and a common information/data model. It is not the goal of the Enhanced Integration Architecture in this paper to define a common information model for all the assembly system applications. Instead, the focus of the paper is on the device–specific operations, independent of the device application and programs.

For example, a robot is an articulated mechanical device that operates an end–effector tool (e.g., welding tip, sprayer, or gripper) under program control. Robots can perform a variety of tasks including welding, drilling, and painting of automobile body parts. Intelligent robots have multiple sensors, effectors, and controls, but we will only consider the robot and its process diagnostic data requirements.

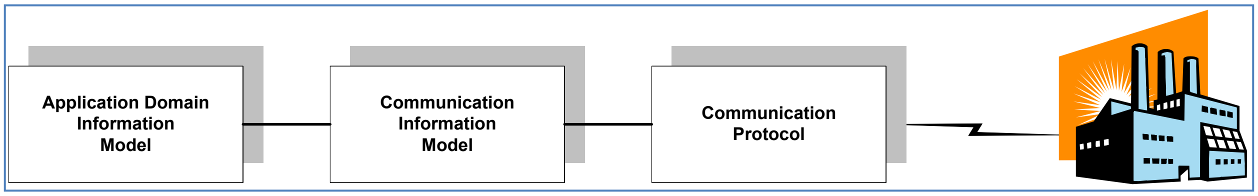
Application Domain Information Model
Communication Information Model
Communication Protocol

Figure 8 Integration architecture information components

Figure 9 shows the relevant activity diagram within the scope of the robot information modeling for our discussion. The focus of common data will be on the status of the robot controller and the related auxiliary equipment. From this, the robot common information model contains device specific information (e.g., welding tip, sprayer, gripper) that is independent of the application such as welding, spray painting, or material handling. In this scenario, we want to determine what can go wrong with the robot system, not with the robot application. Thus, if a robot programmer wrote a bad application program, this is not the information of primary concern. The implication for the common information model is to reflect the importance of the maintenance status of machines, not the application status.

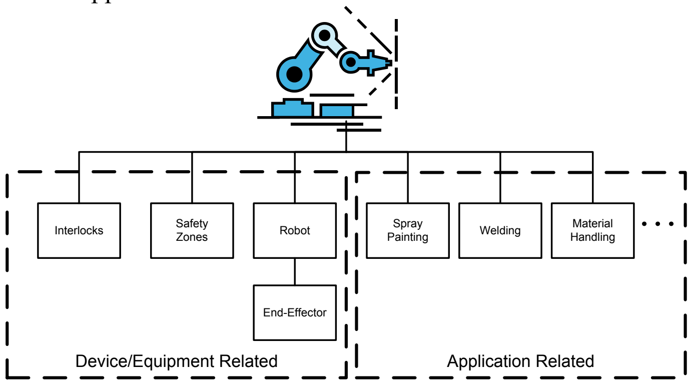

Figure 9 Robot information modeling diagram

Integration Architecture

This section presents a classification scheme in order to better understand manufacturing integration based on communication and programming standards. Then, a comparative analysis of the several high visibility factory integration or programming standards is performed based on their communication and information modeling technology. The distinction between standards for factory floor connectivity versus factory communication protocol is considered important, in that many communication standards may or may not provide an information model describing the device beyond the communication functionality. For example, SOAP is a web communication protocol that is a generic communication and information modeling standard, upon which specialized application information models are built to allow process–to–process communication and connectivity [5]. Another distinction is emphasis on communication from controller to controller, or controller to application, as opposed to a field bus implementation of communication to network devices.

Figure 8 shows the classification scheme for factory integration architecture that will be developed to highlight some of the important distinctions in assembly integration standards. The classification divides the integration model into: the Application Domain Information Model, the Communication Information Model, the Communication Protocol, and the underlying wire protocol.

The Application Domain Information Model refers to the non–communication aspects of the standard, that is, the command, status, and management of a device based on an information model. The Application Domain Information Model describes a set of communication devices in the standard, and can also contain information that spells out device capabilities, configuration, and other meta–data information about the device. Clearly, a standard information model can also decompose devices into subdevices or components to reflect an object–oriented device taxonomy.

The Communication Information Model is separated out as elements because there are a many variations in which devices can communicate, including read only, write only, read/write, synchronous polling, asynchronous event–driven using subscribe/listener software pattern, master/slave, broadcast/multicast/unicast, connection/connectionless, etc. The communication–centric information model can also contain an electronic data sheet that spells out the configuration, topology, and other meta–data information about the network implementation and related communication parameterization.

The Communication Protocol corresponds to the Open Systems Interconnection (OSI) Application layer 7, and covers process–to–process communications, generally across an Internet Protocol (IP) network, using protocols such as HTTP or SOAP. Application layer methods use the underlying transport layer protocols to establish host–to–host connections.

Over the years, numerous standards have been promoted to solve the factory floor integration problem. One legacy factory floor integration standard still in use today is the Manufacturing Message Specification (MMS), which is a messaging system for exchanging data between control applications and networked devices [6, 7]. MMS is a complex standard, with requirements for supporting several OSI Reference Model layers of communication functionality that does easily address the common information model.

Standard	Application Domain	Communication Information Model	Communication Protocol
MMS	Generic	Virtual Manufacturing Device	Messages over TCP
OPC/COM	None	Microsoft IDL DA, Historian,	COM/DCOM
OPC UA	None	Data handling plus OPC Specific Modeling Representation	Modified TCP or SOAP/XML proxy technology with polling, event, subscription, etc.
MTConnect	machine tool, assets, tools, sensors, robots, third party XSD schemas	XML streams	http REST Client/Server model
Robot Operating System (ROS)	Robotics	Programming libraries, URDF(Unified Robot Description Format), Message Ontology	ROS IPC using TCP/IP
PLCopen	Generic Industrial	IEC 61131	MMS, and OPC UA
AutomationML	Generic Industrial	XML	OPC
SEMI CAMX	Electronic Factory	XML	SOAP over HTTP

Table 1 Integration Comparison

OPC is a leading industry standard for the exchange of factory data over a software bus. In addition to the OPC DA discussed earlier, the OPC foundation has developed the OPC Unified Architecture (UA) to support Web Services and Service Orientated Architectures (SOA) [8]. At the communication protocol level, OPC UA leverages existing W3C standards, such as XML, SOAP, and the WS initiatives, and have added an optimized transport layer to improve OPC UA communication performance. OPC UA defines the communication information model, but does not contain any application information model, and the OPC Foundation states that "the Unified Architecture is designed specifically to allow object and information models defined by others." Because OPC does not itself define any application models, information model contributions tend to be domain specific, for example, packaging [9], power industry [10], or plant maintenance [11].

MTConnect is an open and extensible protocol designed for the exchange of data between shop floor equipment and software applications used for monitoring and data analysis [12-14]. MTConnect is based on XML and HTTP and uses a Web "REST" protocol to communicate. MTConnect defines an XSD information model for a variety of devices, including machine tools and its constituent components, sensors, tools, and also allows third party XSD schema data to be communicated. Of note, the OPC Foundation and the MTConnect Institute have a cooperation agreement to ensure interoperability and consistency between the two standards.

PLCs are controllers that provide sequencing control, motion control, process control, distributed control systems, and/or networking for industrial automation systems. There is a trend in industrial automation for PLC–based robotic controls, which is an attractive alternative since the robot has to interface to many auxiliary devices and end–effectors as well as to the PLC. However, because each manufacturer's approach to PLC I/O addressing, memory organization, and instruction sets is different, it makes PLC and vehicle assembly integration somewhat better, but not trouble free.

PLCopen is a standard programming effort for PLCs with its core activities focused around the industrial control programming standard IEC 61131-3 [15, 16]. PLCopen offers communication protocol support for MMS as well as for OPC UA [17, 18]. PLCopen XML is also part of AutomationML (Automation Markup Language), which is an XML–based open standard for the storage and exchange of plant engineering information [19, 20]. AutomationML implements topology with CAEX (IEC 62424) [21]. AutomationML implements geometry with COLLADA [22] to represent graphical attributes and 3D information. AutomationML implements kinematics with COLLADA to represent connections and dependencies among objects to support motion planning. Programming logic in AutomationML is implemented with PLCopen XML [23].

Robot Operating System (ROS) is a free and open source system based on collaboration between industry and academia . Robotics itself is a challenging systems integration problem and ROS offers many tools to manage the complexity in building and understanding systems. A ROS system is composed from a set of reusable libraries that are designed to work independently and use a message-based system for communication. Although ROS is a powerful platform, most of its commercial applications at this time are with mobile robots, and not industrial robots.

Computer Aided Manufacturing using XML (CAMX) is a standards integration technology developed for electronics manufacturing equipment communication, specifically printed circuit board assembly [24]. The set of standards include models for board fabrication, bare board product electrical testing data, assembly manufacturing, and assembly test and inspection. Although the application is a different domain, eh set of standards provide an insight into the breadth of vehicle integration common information models that are potentially required.

Table 1 presents a comparison of the different integration capabilities covered. Although each integration strategy has its strengths, no technology has proven to be the dominant player in solving the vehicle integration problem, and as such commercial proprietary solutions featuring silo integration architectures will continue to persist.

DISCUSSION

In general, difficulties arise for companies and vendors to adopt standards that are excessively broad and complex, making it costly to develop the required hardware and software. Further, the limited adoption of a standard yields marginal benefits. However, the lack of integration standards is severely hampering the vehicle assembly integration, production, and maintenance. This paper looked at using an Enhanced Integration Architecture with a provision for a common information model. A common information model would allow application independent device health data. Thus, in addition to data collection, formatting, and cleansing, the model would also provide the capability to focus on improving performance and behavior of the plant-floor system.

We reviewed several industrial standards or open source solutions relevant to vehicle assembly, and presented a categorization of their information model capability. Each of these technologies provides attractive partial solutions, but at this time no single technology has a meaningful impact in the vehicle assembly market.

It is not the goal of this paper to pick winners or losers, but instead attempt to better elucidate the universe of potential integration solutions, in order to better understand a potential migration path to the described enhanced integration architecture. As such, it remains a challenge to integrate equipment from numerous suppliers, each with their own silo application view of the factory. In order to achieve success in factory integration for manufacturers, it will require some cooperative activities of end-users, vendors, and academia.

DISCLAIMER

Commercial equipment and software, many of which are either registered or trademarked, are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology or General Motors, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

REFERENCES

[1] National Institute of Standards and Technology, 2012, *"Smart Assembly: Industry Needs and Technical Challenge"*, smartassembly.wikispaces.com.

[2] D. Kibira and C. R. McLean, 2007, *"Generic simulation of automotive assembly for interoperability testing,"* in Proceedings of the 2007 Winter Simulation Conference, Washington, DC, United states, pp. 1035-1043.

[3] J. Wang, J. Yu, S. Li, G. Xiao, Q. Chang, and S. Biller, 2008, *"A Data Enabled Operation-Based Simulation for Automotive Assembly,"* ICSC 2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing, pp. 726-731.

[4] OPC Foundation, 1998, *"OPC Data Access 2.0 Specification."*

[5] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, 2003, *"SOAP Version 1.2 Part 2: Adjuncts,"* W3C.

[6] International Organization for Standardization, 1990,*"ISO/IEC 9506-1, Industrial Automation Systems - Manufacturing Message Specification - Part 1: Service Definition."*

[7] International Organization for Standardization, 1990,*"ISO/IEC 9506-1, Industrial Automation Systems – Manufacturing Message Specification - Part 2: Protocol Specification."*

[8] International Electrotechnical Commission, 2008, *"IEC 62541-1: OPC Unified Architecture Specification - Part 1: Overview and Concepts."*

[9] Microsoft Corporation, *COM: Component Object Model Technologies.*

[10] R. D. S., Fensel; A., Fensel, 2011, *"OPC UA goes semantics: Integrated communications in smart grids,"* in 2011 IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-4.

[11] Machinery Information Management Open System Alliance (MIMOSA), www.mimosa.org.

[12] MTConnect Institute, *"MTConnect Standard Part 1: Protocol and Overview"*, http://mtconnect.org.

[13] MTConnect Institute, *"MTConnect Standard Part 2: Components and Data Items"*, http://mtconnect.org.

[14] MTConnect Institute, *"MTConnect Standard Part 3: Streams, Samples, and Events"*, http://mtconnect.org.

[15] E. Van der Wal, 1999, *"Introduction into IEC 1131-3 and PLCopen,"* in IEE Colloquia on the Application of IEC 61131 in Industrial Control: Improve Your Bottom-Line Through High Value Industrial Control Systems.

[16] E. Van der Wal, 2000, *"IEC 61131-3 software: changing the world of industial automation - the statue, the structuring tools, the activities and the libraries,"* in IEE Colloquia on the Application of IEC 61131 in Industrial Control: Improve Your Bottom-Line Through High Value Industrial Control Systems.

[17] I. Miyazawa, M. Murakami, T. Matsukuma, K. Fukushima, Y. Maruyama, M. Matsumoto, J. Kawamoto, and E. Yamashita, 2011, *"OPC UA information model, data exchange, safety and security for IEC 61131,"* in 2011 Proceedings of SICE Annual Conference (SICE), pp. 1556-1559.

[18] PLCopen and OPC Foundation, 2010, *"PLCopen and OPC Foundation: OPC UA Information Model for IEC 61131-3. Release 1.00."*

[19] R. Drath, A. Lüder, J. Peschke, and L. Hundt, 2008, *"AutomationML - the glue for seamless automation engineering,"* Emerging Technologies and Factory Automation, pp. 616-623.

[20] M. Schleipen and R. Drath, 2009, *"Three-view-concept for modeling process or manufacturing plants with AutomationML,"* Emerging Technologies and Factory Automation, pp. 1-4.

[21] International Electrotechnical Commission, 2008, *IEC 62424, Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*

[22] R. B. Arnaud, Mark C., 2006, *COLLADA: Sailing the Gulf of 3D Digital Content Creation*: A K Peters/CRC Press.

[23] M. Marcos, E. Estevez, F. Perez, and E. Der Wal, 2009,*"XML exchange of control programs,"* Industrial Electronics Magazine, IEEE, vol. 3, pp. 32-35.

[24] S. Cousins, 2010,*"Welcome to ROS topics,"* IEEE Robotics and Automation Magazine, vol. 17, pp. 13-14.

[25] S. Cousins, B. Gerkey, K. Conley, and Willow Garage, 2010,*"Sharing software with ROS,"* IEEE Robotics and Automation Magazine, vol. 17, pp. 12-14.