

Aspect Object Technology

The Practical Application of Aspect Object Technology in a Distributed Control System

Condensed from multiple ABB authors

The Problem

A central problem in plant operations as well as asset lifecycle management is that you need a way to keep together, manage, and have access to information about all different aspects of a great number of plant and process entities. These entities, or real world objects, are of many different kinds. They can be physical process objects, like a valve or a motor, or more complex entities, like a reactor. They can be immaterial, like recipes, manufacturing orders, and customer accounts. Other examples are products, raw material, production batches, etc.

In a system that integrates automation, information, and collaborative business processes across the enterprise, each of these real world objects needs to be described from several different perspectives. Each perspective defines a piece of information, and a set of functions to create, access, and manipulate this information. We call this an *aspect* of the object.

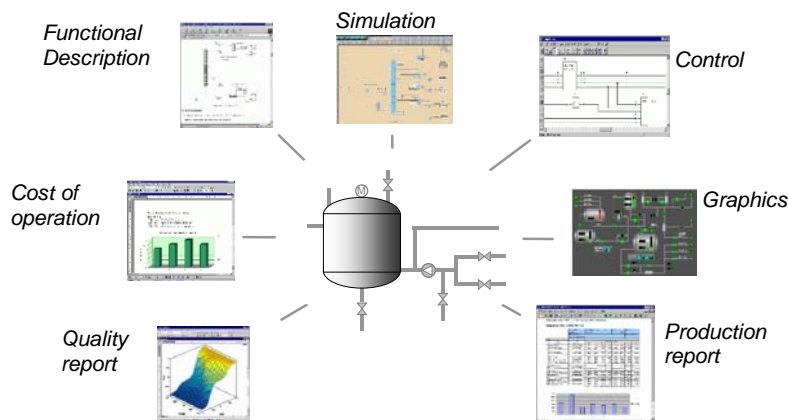


Figure 1: Different aspects of an object

It is for many reasons necessary to be able to implement these aspects using many different applications, existing and new ones, from ABB, third parties and customers. It is desirable to be able to do this without changing the way these applications work internally. The applications must be able to cooperate to provide an integrated view and functionality of the object.

The Solution: Aspect Objects

Aspect Objects provide a solution to this problem. In this concept, rather than creating one single object or data model in the system to represent the real world object, each aspect is modeled separately. An Aspect Object is a container that includes these independent models.

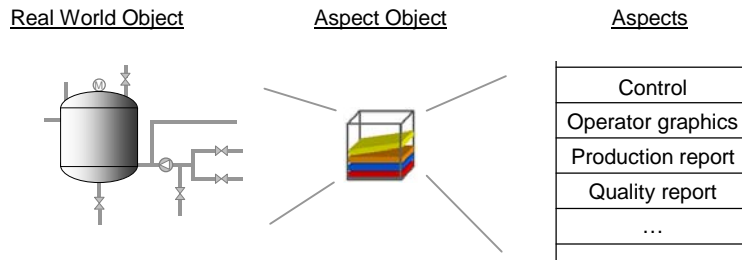


Figure 2: An Aspect Object is a container of aspects

Different kinds of Aspect Objects have different sets of aspects. Figure 3 below shows some examples: a device level object (e.g. a valve or a motor), a unit level object (e.g. a reaction vessel), and a manufacturing order object.

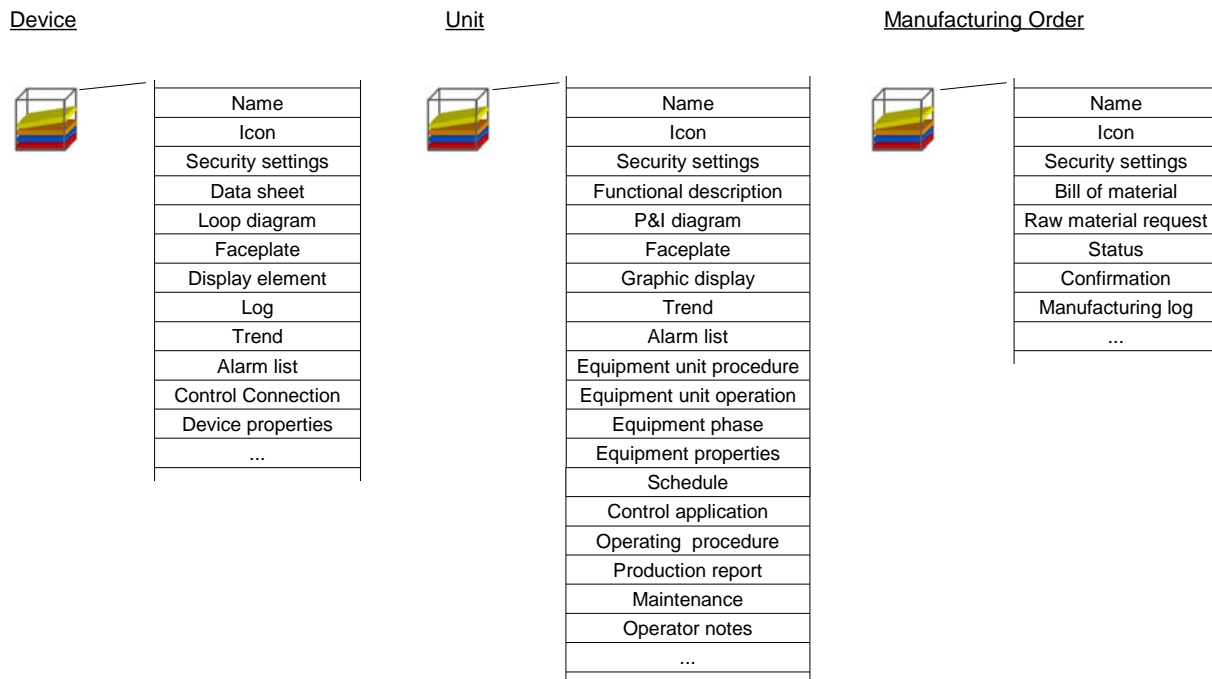


Figure 3: Aspect examples

Aspect Object Architecture

The Aspect Object Architecture is made up of an information model, basic system functions, tools and documentation. This architecture forms the core of ABB's System 800xA extended automation distributed control system (the core of which was previously called the *Aspect Integrator Platform*). This information architecture seamlessly links equipment and applications in real time.

There are several important requirements that the Aspect Object architecture is designed to meet:

- Make it possible to build a system that provides functionality for process automation, asset optimization and collaborative business processes, yet is easily understood and efficient to use
- Be optimized for the performance, predictability, reliability and availability that is required for high performance, real-time process control
- Provide security mechanisms that allow all operations to be access controlled and logged to comply with regulatory requirements
- Scale competitively from very small to very large
- Provide strong support for building reusable application solutions
- Allow software and equipment of different origin and with different internal implementation technologies to be integrated and work together as one consistent and integrated system
- Allow new functionality to be incrementally added to and integrated with the system without changing or recompiling existing software
- Efficiently support development by independent groups in a distributed organization.

Aspect Systems

Aspects are implemented by software systems known as *aspect systems*, each of which stores, manages and presents its information in its own optimal way. The environment in which aspect systems are integrated is called the *Aspect Framework*. This framework provides mechanisms by which the aspects systems can cooperate and share data, to provide an integrated view and functionality of the object, and one time data entry.

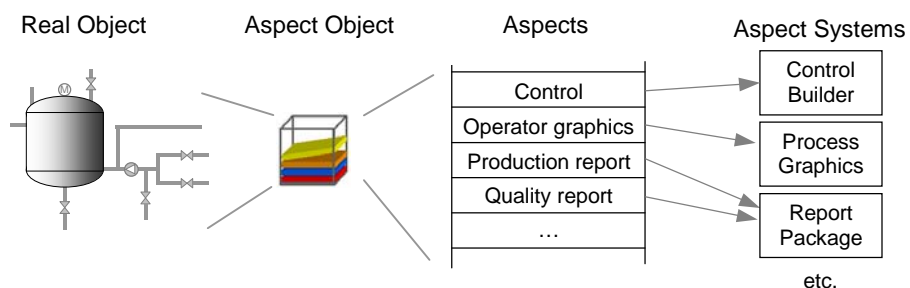


Figure 4: Different aspects are implemented by different aspect systems

Each aspect system is responsible for storing and maintaining its own data. However, in many cases data must be shared among a group of aspect systems. Unless handled in a correct way, this may cause data consistency problems. The Aspect Object architecture offers two ways to avoid that:

- A shared data item is stored in only one copy by one aspect system, and other aspect systems access it from there, through framework defined interfaces.
- Several aspect systems hold their own copies of a shared data item. When an aspect system updates its copy, it must inform the framework, which in turn informs other aspects systems to update their copies.

An aspect system defines one or several *aspect types*, each representing the implementation of a certain aspect. Of each aspect type, one or more *aspect categories* can be defined as different specializations. For example, the aspect system Graphics implements the aspect types Graphic Display, Faceplate, and Display Element, where the aspect type Graphic Display includes the categories Overview, Group, and Object Display. See Figure 5:

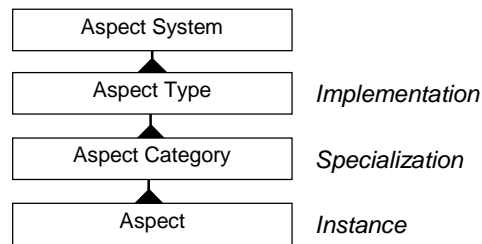


Figure 5: Relations between aspects and aspect systems

In the Aspect Object architecture, aspect systems cooperate with other aspect systems without knowing which they are, or even how many they are. To make this possible, they do not interact directly with each other, but only with the Aspect Framework.

The framework includes an *Aspect Directory*, where all Aspect Objects and their aspects are registered, and also all aspect systems and the operations they support. To perform an operation on an Aspect Object, an application (e.g. an aspect system) invokes a framework interface for that operation. Using information in the Aspect Directory, the framework then invokes the corresponding interfaces of all aspect systems that are concerned by that particular operation for that particular object. Thus, to copy and paste an object, for example, all aspect systems that implement aspects that are defined for that object are involved and perform their part of the operation.

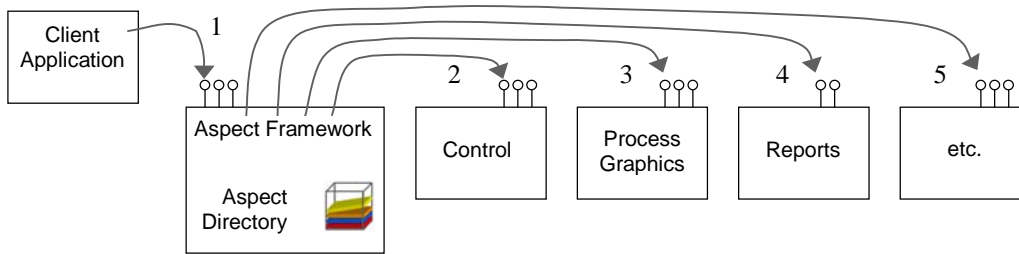


Figure 6: The Aspect Framework provides application independence

The result is a system of integrated but independent software systems. It is open, so that new software systems that were not even anticipated from start can be added without changing or recompiling the ones that are already in place. Users work with concepts and entities (objects) that are familiar to them, i.e., valves, reactors, products, manufacturing orders, customers, etc., rather than with the implementation objects that realize the individual aspects.

Using Object Structures and Plant Modeling

A very natural way to represent relations between different entities is to organize them in a *structure*. Depending on from which perspective we look at it, the same entity fits naturally in several different structures. For example, a certain piece of process equipment has a certain position in a functional structure depending on the functional breakdown of the plant. It is also physically placed somewhere, and thus it has a place in a location structure. The same piece of equipment may currently be allocated to a certain production batch, so it belongs in the batch structure. Because it is used to produce a certain product, it fits in a product structure.

Thus, there is a very obvious need to organize Aspect Objects in structures. It must be possible for one Aspect Object to be represented in several structures at the same time. For certain applications it must be possible to dynamically move an object between different positions in the same or a different structure, for example representing products moving through an assembly line, or production orders being allocated to different process equipment.

The concept of structures is central in the Aspect Object architecture. A number of structures are used to represent different types of information related to the system. All Aspect Object structures are hierarchical, i.e., the structures are defined by parent-child relations between Aspect Objects. Standards dealing with structural relationships between entities, such as IEC 61346 and S88, are supported.

The relation to a certain structure is represented as an aspect. By adding several structure aspects to an Aspect Object, the object can be placed in several structures, or even in several positions within the same structure. By dynamically adding, deleting, and changing structure aspects, the object can be inserted in, deleted from, or moved to different positions in various structures.

By means of Aspect Objects, and by arranging Objects in various structures, it is thus possible to effectively model many and various types of plants, equipment, products, processes and procedures.

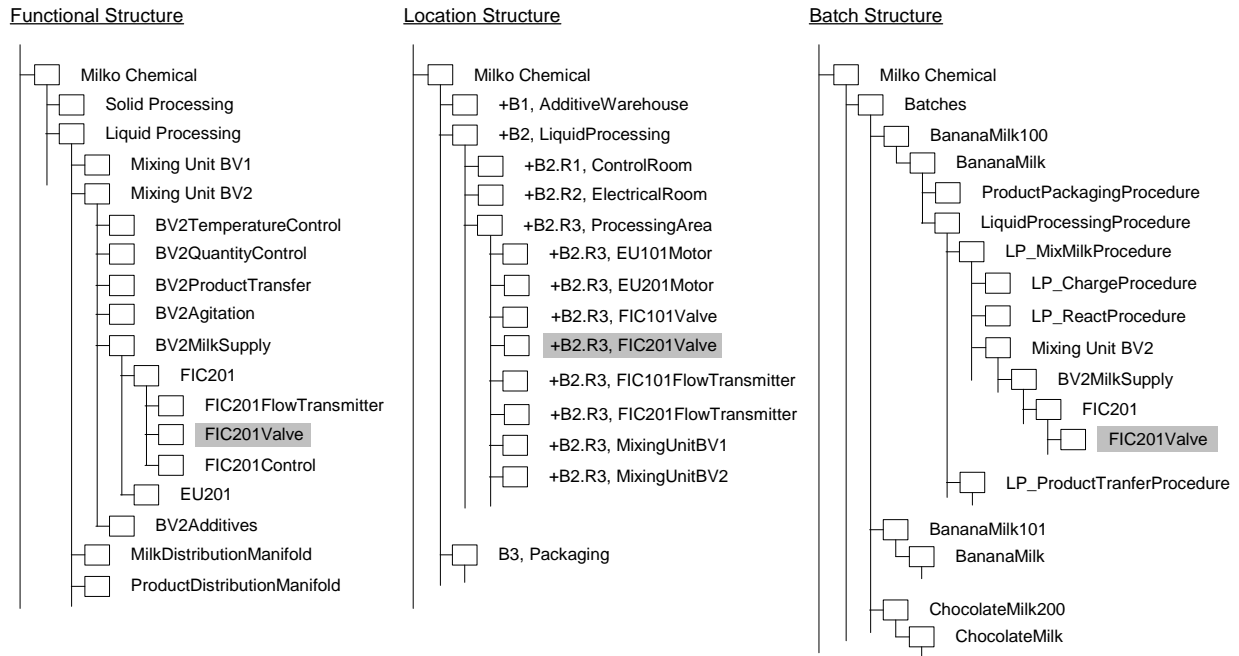


Figure 7: Multiple object structures

The *Plant Explorer* is used for navigation in and between object structures. It is the default tool to create and manage Aspect Objects and to browse for information. It is based on the Windows Explorer metaphor, but instead of folders and files it deals with Aspect Objects and aspects.

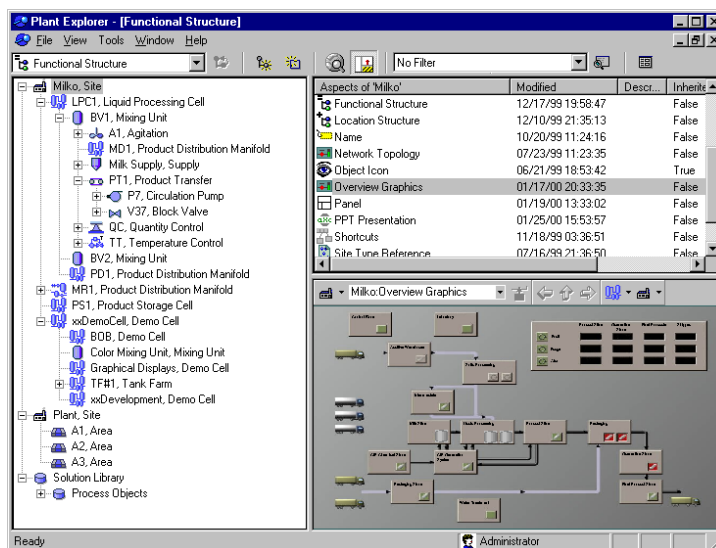


Figure 8: The System 800xA Plant Explorer

An Aspect Object can inherit aspects from its parent or a higher-level ancestor in each structure where it is placed. The child object inherits those aspects of the ancestor object that are marked “to-be-inherited”. This *structure inheritance* is different from traditional forms of inheritance in object oriented systems, in that it is not defined in terms of object class hierarchies, but in terms of structural relationships between object instances of unrelated classes. For example, a control valve may be part of a flow control loop for a mixing unit. The valve is placed as a child to the flow control loop in the functional structure, and thus inherits the security settings that apply to the mixing unit.

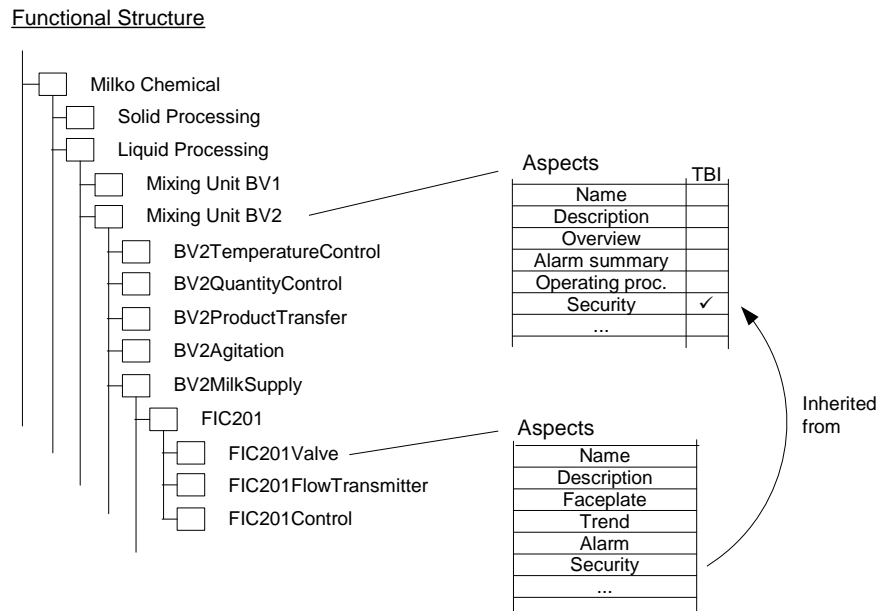


Figure 9: Structure inheritance

Structure inheritance is dynamic. When an Aspect Object is inserted in a particular structure, it inherits aspects from its ancestors. When it is deleted from a structure, it loses the aspects that were inherited through that structure. When it is again inserted in a different position in the same structure, or in a different structure, it inherits from its new ancestors.

Aspect Object Types

An *Aspect Object type* defines certain characteristics that are common to several Aspect Object instances, such as a basic set of common aspects. This makes it possible to create and efficiently re-use standardized solutions to frequently recurring problems. For example, rather than building an Aspect Object from scratch for every valve in a plant, you can define a set of valve types, and then create all valve objects as instances of these types.

When an *instance* of an object type is created, the aspects that are defined in the object type are instantiated and associated with it. You can add aspects to a specific instance, or replace inherited

aspects with instance specific aspects of the same type, but it is not possible to delete aspects that were inherited from the object type.

An object type has rules associated with it. These are either aspect rules that control what aspects can be associated with an instance of that type, or child rules that control what objects can be placed as children under an instance of that type, in a particular structure.

Object types can be created as *specializations* of other types. A specialized object type inherits aspects and other characteristics from the type it is derived from, the *super type*. For example, from a generic valve type that has a certain set of aspects, you can create specializations for block valves, control valves, etc., adding aspects and other characteristics that are specific to those types.

A *simple* object type describes one object; each time it is instantiated, precisely one object is created.

A *composite* object type describes a set of objects organized in a structure, with a parent object and one or several child objects. The children in a composite object type are called *formal instances*, because they inherit from object types defined elsewhere in the Object Type Structure, but they are not actual instances. Only when a composite object is instantiated are actual instances created for these child objects. This is illustrated in 10.

Object types are placed in the *object type structure*. They can be packaged, delivered, and installed as object type *libraries*.

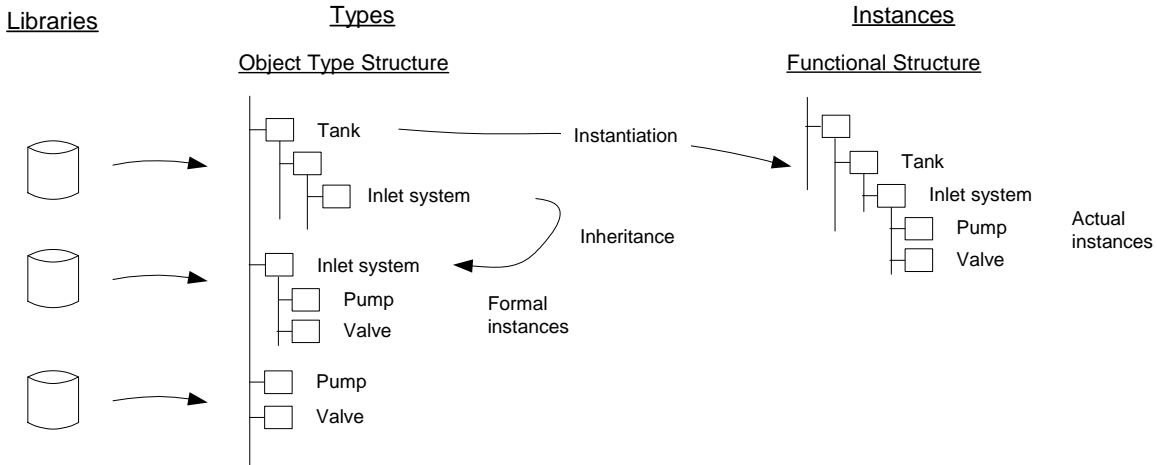


Figure 10: Aspect Object types

Aspect Object Technology in System 800xA

An important feature of System 800xA is that information and functions are centered on Aspect Objects. To participate in Aspect Object operations, an application must present itself as an *aspect system* (or possibly as several aspect systems). In essence this means that the application provides *aspect system objects* which support certain framework-defined interfaces, through which the application can initiate and participate in common operations on objects and aspects.

Object architectures allow for all of the similar pumps in a process plant, for example, to be treated as “instances” of a more generic pump “object.” These pump objects in turn can in turn be aggregated into larger objects such as distillation columns, and so on. This underlying object architecture is what makes it possible to implement the common engineering, information and visualization environment that effectively abstracts implementation details from the configuration and day-to-day management of seamlessly integrated production systems.

In a practical sense, this means that the basic and extended functions of System 800xA are “plugged in” to the information (aspect object) architecture which is the core of the system. See Figure 11 below:

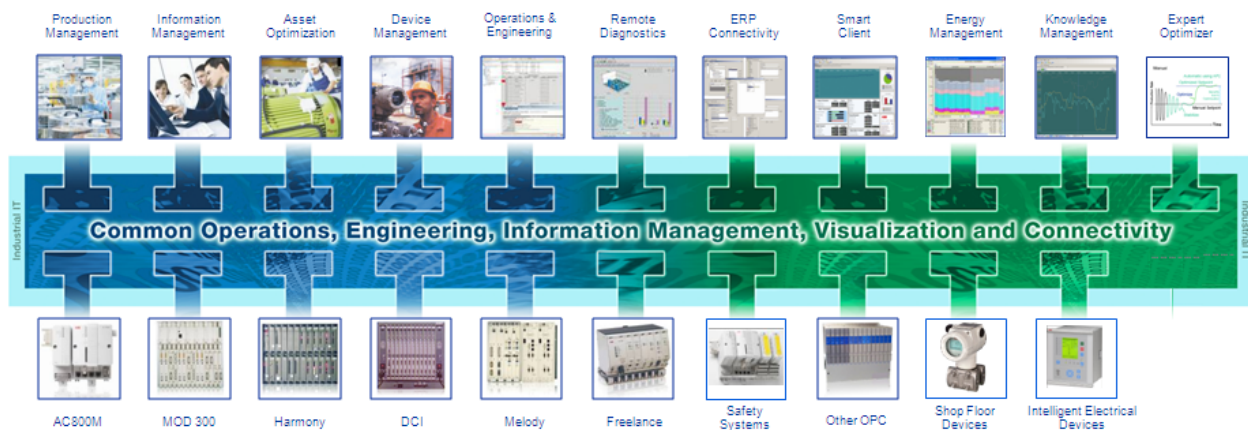


Figure 11: System 800xA common information platform

Summary

System 800xA delivers the exact information and collaboration environment necessary for the formation and execution of sound business decisions. ABB’s integration architecture, based on Aspect Object technology, relates all of your plant data, the Aspects, to the specific plant assets, the Objects. This architecture enables one click navigation, efficient engineering and presentation of the right information in the right context to the right user.

The integration architecture of System 800xA allows seamless integration of applications and supports 3rd party systems such as computer based maintenance management and video systems. As shown in

the diagram in Figure 11, applications developed with or for System 800xA can be “plugged into” the common engineering, information and visualization environment.

This same architecture enables us to deliver solutions such as integrated process control and safety and integrated process and power automation solutions. Both these and many other System 800xA solutions feature common visualization across plant areas and deliver savings in engineering as well facilitating the right operational and business decisions and actions to maximize productivity.