Expression and Application of Smart Manufacturing Profiles

The Smart Manufacturing (SM) Platform is a combination of strategic technologies useful for organized collection and transmission of manufacturing data, which work together to enable open access to a variety of applications, helping the industry identify constraints, predict challenges, and optimize decision-making and operational orchestration. At a high level, there are three main components to the SM Platform: the Edge, the Core Services, and the Integrated Applications – although the architecture will support many different deployment and scale options, as a general rule the demarcation between components is as follows:

- The SM Edge resides close to the machine or system that is emitting data, and is used to collect that data in-context and transmit it to the Core Services.
- The Core Services reside in a hyperscale Cloud platform and receive the transmitted data, organize and store it, and make it available to constituent applications and services.
- The Integrated Applications are consumers of the data, but also augmenters and modifiers, that can work together through Workflow orchestrations in a data-centric fashion.

All of these components use or benefit from, the Smart Manufacturing Profile – a key enabler to democratizing data access in manufacturing. The SM Profile provides a “decoder ring” for the source equipment or system, a mechanism for establishing and maintaining the intent and context of the data as it flows through the system, and an interface definition language for programmatic access when the data is at rest in an information system.

The SM Profile as the Abstract Class

At its most primitive level, the SM Profile provides an open-standard language for Class definition, following Object Oriented Programming conventions. This definition is articulated in the Abstract, and as a derivation – all SM Profiles derive from other SM Profiles, save for the absolute base Class (for this discussion, called the AbstractManufacturingObject). Class derivation allows for increasing levels of specificity as a Model matures or is extended by other constituents. In this manner, a broad industry exercise to model Extrusion machines, for example, can be nearly universally applicable – although minimally valuable. Follow-up development that extends that base Extrusion class for specific types of Extruders or extrusion applications can assume the pre-existing functionality and focus on creating new, case-specific value.

Figure 1: An Abstract Profile Definition (Mock-Up)
As an Abstract Class definition, the primitive SM Profile contains no information about a particular instance, or even how to create or hydrate one; rather it is the data definition (called Data Points from here on) only.

**Data Binding the SM Profile through Instantiation**

The first practical extension to the Abstract class then is the Data Acquisition extension. Although this creates a derived Class, through application of inheritance principles, this is a functional extension, not a Model extension. Multiple Data Acquisition extensions can be declared, and become part of a Compound Class that is assembled by the Interpreter on-the-fly. The Data Acquisition extension provides a Data Source Definition, which indicate the real-world binding instructions that Interpreter uses to bind the Abstract Data Point to a physical Data Source. The extension does not limit the Data Source definition to a particular protocol – nor does it guarantee that a given Interpreter is capable of communicating over that protocol. It simply provides the communication instructions that should be utilized if an instance and Interpreter pair were available in the real world.

The pairing of an instance of a real world machine/system communicating on a given protocol, that matches an existing Abstract Class definition (SM Profile) and an Interpreter capable of communicating on that protocol results in a programmatic Instantiation of the SM Profile Object in the SM Platform. In an ideal state, this match can be automatically detected with no human intervention. However, since no existing specification provides for universally unique machine or system identification, the near future implementation will require a human to specify a match – specifically, a human will select the Profile that fits what they know about their machine or system and its communication capabilities.

Regardless of how the pairing is identified, the typical deployment scenario will have this operation performed on the “Edge.” The SM Platform specifies a component that is responsible for data acquisition and publication to the core SM Platform services, with appropriate support for secure communication and transmission, fault tolerance, buffering and fail-over. For reasons of cyber security CESMII does not recommend connecting devices, machines or systems directly to the Internet – nor would most manufacturing networks support such a connection. The SM Edge functions as a data diode, ensuring data can be securely transmitted to the SM Platform, but that no third party may transfer data back to the source machine or system.

As the component usually responsible for data acquisition and instantiation, the SM Edge is the first SM Profile Interpreter. It must include a mechanism for retrieving Profiles from the SM Platform (either through automatic detection or user interaction), an ability to assemble the Compound Class of all applicable Profile extensions on-the-fly, and the ability to instantiate an Object, apply at least the Data Acquisition extension instructions, and execute the requisite communication messaging to the source. Because some communication protocols and device storage mechanisms are necessarily primitive, the Interpreter must be able to apply simple combination, concatenation or
conversion on-the-fly. Other extensions to the Profile may include ingestion or data rate rules, data type definitions, access control defaults, information characterization, or basic streaming analytics, to be determined in the future. The Edge Interpreter must ignore Profile instructions it does not understand – a principle that should be applied for every Interpreter.

Profiles as an Open Interface Definition Language

The next Profile interpreter is the SM Platform itself – specifically, the Core Services. An important capability of the Core Services is the Context Graph; this is the Relationship Model between object instances. Each object instance delivered by the SM Edge is relatively basic, by virtue of the unapplied Profile extensions that the Edge does not have the ability to Interpret. The Context Graph will augment the Instance by reading in the Profile and morphing the instance to include the members that were ignored on the Edge. This may include rendering or analysis rules, code extensions, integrations with member application APIs, or Workflow extensions. This newly enriched object is stored in the Context Graph, and related to other objects; it is the SM Platform that contains hierarchical relationships between objects (eg: ISA 95, or an Enterprise view). Even constituent parts of a system (eg: this machine is made up of 5 devices, and both the machine and the devices have Profiles) will be presented as a flat list by the Edge component. It’s the responsibility of the Context Graph to find inferred relationships, as well as apply human-curated relationships.

This newly enriched Object stored in the Context Graph have their member values refreshed via the stream from the Edge component(s), but execution or calculation of the enhanced member values will be performed at the Platform level. It’s important to note that while the Profile definition or extensions may encapsulate, or indicate, code to be executed, the Profile itself is not an executable file – it’s an interpreted file.

Finally, the SM Platform retains the Profile Compound Class (the assembly of all Profile extensions) as a link from each instance. This important feature of the Platform provides an Interface Definition for each Object, so that external code can determine the Type (and sub-Types) for any requested Object. Any API call to the SM Platform for data will have a parallel call that returns the Interface Definition in the form of the assembled Profile.

![Figure 3: Profiles in use for Interface Definitions (Mock-Up)](image)

This allows Integrated Applications, the third Profile Interpreter, to match any Object to whatever capabilities it knows about – regardless of the polymorphic nature of the Platform’s data model. As an example, a dashboarding application may know how to render a status display for any Extruder that derives from the afore-mentioned industry modeling exercise. When this application requests a payload of objects from the SM Platform, the resulting Instances will likely have been extended (morphed) into a new Compound Class which has changed the structure and data of the Instance far beyond what the Interpreting application has any knowledge of. However, since the Interpreting application is also provided with the assembled Profile, as an Interface Definition, it can still walk the interface to find the subset of data it knows how to render. This interface reflection can be utilized by any integrating application or tool, including Workflow, Data Science and AI algorithms, or other member capabilities.
Polymorphism through Graph Relationships

Polymorphism isn’t restricted to the core SM Platform or SM Edge Interpreters, however. Applications that Interpret the Instance objects may also extend them on-the-fly as well – provided they follow the patterns established by CESMII. In this fashion, an Application that finds Objects it understands may also pass back to the SM Platform an extension Profile, and corresponding data. Rather than directly morphing the live object instance, this licensed transaction causes a new Object to be created in the SM Platform Context graph, of the Type specified by the extension Profile, and a new graph relationship link to be established between the original object instance, and the new extension object instance – a foreign key linking external data to the data stored in the Platform. This “loose coupling” allows any CESMII member to contribute their capabilities to the Platform, without needing to inject them into the data ingestion end of stack. Other applications can follow links in the Context Graph, read the Interface definition, and match or use capabilities of otherwise unintegrated applications, in a data-centric approach to a horizontally scaled architecture.

Conclusions

At every level of the stack, the SM Profile enables the capabilities of the Platform, and other Smart Manufacturing applications. The extensible nature of the SM Profile allows different capabilities to be expressed by personas most knowledgeable about a particular need, eg: the machine vendor may provide the Abstract class definition, but the System Integrator may provide the Data Source definitions. The various Interpreters of the Profile perform the instructions that are appropriate for their domain, and morph or extend the Instances on-the-fly, to make the platform data centric, and almost infinitely adaptable in an open fashion.