Software Architecture of Manufacturing Execution Systems

Heiko MEYER
University of Applied Science Munich
Department of Business Administration and Engineering
Lothstraße 34, 80335 Munich, Germany

and

Heiko MEYER
Gefasoft AG
Research and Development
Dessauerstraße 15, 80992 Munich, Germany

ABSTRACT

The globalization of the economy and the associated factors of increasing effectiveness in production, shortening innovation cycles, safeguarding high quality, etc. are continually augmenting the pressure on the production business. It has been possible to compensate somewhat for this pressure in recent years by relocating production to low-cost countries. However, in the medium term, the demands of workers in countries that are now still low cost will increase, and production costs will rise as a result, so the need for action will arise. Tools will be needed to increase efficiency in existing production processes. It also must be considered that production in high-cost countries definitely has its advantages, so these countries are becoming more and more feasible as production locations and will remain so in the long term. The degree of automation is already extremely high in these countries, so modifying production processes will not increase efficiency significantly.

Keywords: Database Management System, Enterprise Resource Planning, Manufacturing Execution System, OPC UA, SOAP, Web service.

1. INTRODUCTION

Additional new challenges for production-oriented information technology (IT) systems arise from norms and guidelines such as quality assurance standards and regulations in the food and pharmaceutical industries. While these demands were relevant mainly for security-oriented systems decades ago, now transparency and traceability are playing an increasingly important role in other sectors as well.

In order to achieve effective value creation in production, equipment is needed that can meet these new demands 100 percent. Existing enterprise resource planning (ERP) systems established on the market are largely administrative and accounting systems. The new systems needed must include functions for planning, logging, and control that not only act but also react in real time. For these systems, the concept of a *manufacturing execution system* (MES) has arisen. Since MES is a multifaceted area, each sector interprets the concept from its own standpoint.

2. FUNDAMENTAL VARIANTS

If you analyze the architecture of the different systems, which are temporarily on the market, you will find two architectural variants with fundamentally different approaches:

- *Application-centered systems.* Here, the application controls the booking function in the database and the business logic of the system. The database serves only as a performance-saving medium.
- *Database-centered systems.* With this approach, the database is not only a data memory but also the pivot of the entire system. A large part of the bookings and also parts of the business logic are handled through the database.

Application-centered approaches offer advantages for development by the use of high-level languages. Updates are also simpler because the data structures are less complex. The main disadvantage is that errors in the application logic endanger the consistency of the data - in case of doubt, extensive repairs to the database by the manufacturer become necessary. In addition, core themes of a database management system (DBMS), such as cache consistency and transaction management, must be integrated through the increased use of multithreading; these have matured for years in DBMS with regard to freedom from errors, scaling, and performance. In systems with large data volumes, performance disadvantages also can arise because optimizations for the use of special DBMS strengths are not possible or need to be implemented several times and therefore are left out.

In database-centered systems, all data-related operations also occur within the DBMS in its native programming language in the form of what are known as *stored procedures.* A disadvantage is that platform dependency is no longer provided at this point. Software development also becomes more difficult because generally a high-level language offers more options than the programming language of the DBMS. This disadvantage is amplified by various current initiatives to make high-level languages (e.g., Java or .Net) directly useful for the programming of stored procedures. Updates are more elaborate because tests for integrity are an elementary part of the update process. The great advantage of the database-centered approach is that transaction management and the safeguarding of data

consistency are transferred by declaration to the DBMS. The existing abilities of the DBMS, such as the support of cluster solutions or distributed systems, can be used without large adjustments to the application. The *data model* is a part of the application; that is, access becomes simpler for external systems (e.g., reporting) and is also possible without the contribution of the manufacturer in cases of doubt. A considerably better performance can be made possible by focusing on the DBMS with its specific optimizing options. This is particularly important for a manufacturing execution system (MES), which is confronted with a considerably higher transaction volume than an enterprise resource planning (ERP) system.

The following conclusion can be drawn from this consideration: Application-centered approaches are easily at an advantage in systems with relatively low transaction volumes as well as in the early development stadium. For large systems with high requirements for availability, data integrity, and performance, on the other hand, the database-centered approaches are at an advantage. The considerations in this chapter therefore are based largely on this approach.

3. OVERVIEW OF CENTRAL COMPONENTS

The software architecture suggested in Figure 1 for an MES is also present in many systems currently on the market in this form or a similar form.

Figure 1 shows a server-based system, the core of which forms a relational *database.* In order to be able to handle larger data sets, the database can be divided into two parts—an *online database* and an *archive database*. This database also takes on the basic functions of data processing, such as the booking in of complex data into various table structures. Laborious processing functions such as the calculation of key figures are handled in an external module. In this module (e.g., in the form of an application server), the *business logic* and *administrative tasks* (e.g., authentication of the user and data care jobs) of the MES are mapped out. Independent *interface adapters* and/or *software services* are available as interfaces to neighboring information technology (IT) systems. A server component is also needed for the *user interface* and the *reporting module* embedded in it. In the case of a Web solution, this is a Web or application server.

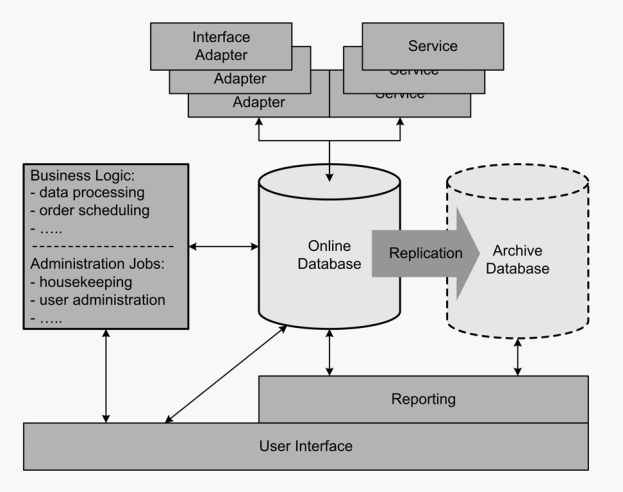

Figure 1: Central software components as an overview.

4. APPLICATION SERVER

An *application server* is a component that provides a framework and various services for the execution of applications. Here, the concept *server* does not necessarily denote an independent hardware system. An application server provides special services such as authentication or access to directory services and databases via defined interfaces to the applications as a run-time environment.

The concept *application server* has developed to become one of the most used concepts in IT. Other concepts that are used in this context are *middleware* and *three-tier architecture.* Software applications with a three-level architecture generally are classified in the presentation, business logic, and data management layers. Applications for the mapping out of the business logic are also referred to as *application servers* in today's usage. Because of application in this middle layer, it is also referred to as *middleware.*

5. PLATFORM INDEPENDENCE

The necessary outlay for the creation of true platform independence may seem high but must be taken into account with regard to the long system running times of an MES (generally more than 10 years). Changes in the IT landscape of the company may not lead to the end of the MES implemented; much more, platform independence should help to minimize costs for running and care.

At this point, the concept *platform* refers especially to the fundamental *computer architecture* (e.g. processors) *and the operating system used.* Naturally, the database also represents a platform, but this is not taken into account because there are only a small number of systems on the market that support different databases. Despite this, true platform independence is only a given if the user also can select the database system freely.

On the subject of platform independence, two generally different viewpoints exist on the market:

- Users, generally in small and medium-sized companies, who are not set on any particular platforms and therefore follow the recommendation of the supplier
- Users, often in large companies with independent IT departments, who have decided on internal conditions for platforms and also only permit systems that comply with these conditions

If a supplier wishes to serve both groups (so the entire potential market), this supplier's system absolutely must be *platform-independent.* In order to analyze this requirement more closely, the individual modules of the system are examined separately:

- *Database.* The database is the central core of the system. Therefore, the highest requirements with regard to scalability, availability, and performance are made of this module. Many suppliers support only one database system because it is hardly possible to carry out optimization for different databases. The database also requires regular software maintenance, for which

the relevant skilled workers are needed. *Conclusion:* If the database used is platform-independent (e.g., Oracle), an implementation on the platform preferred by the customer can succeed; if not, it is necessary to use a platform recommended by the supplier.

- *Business logic/administrative tasks.* Many of these data-processing tasks can be completed directly in the database using stored procedures. This method is distinguished by high processing speeds. The disadvantages are that more complex tasks can be mapped out only in Structured Query Language (SQL) with difficulty and that change management is cumbersome. If you wish to avoid these disadvantages, you should use a Java- or C++-based approach. Both codes can be ported to different platforms with acceptable outlay (and so are platform independent to a limited extent), provided that you do not use any special libraries. From the viewpoint of user friendliness, the encapsulation into individual "functions" is desirable. For this purpose, the application of an *application server* or a *script engine* that contains individual jobs in the form of Java or JavaScript programs is possible.
- *Interface adapter.* Here, in turn, what we mentioned under "Business logic/administrative tasks" above applies to platform independence. It is possible that the handling of ports also can occur in the same software module. However, adaptiveness with regard to change is even more important for interfaces—no other topic is so often changed and requires as much time for implementation and testing. The use of scripts (at best, independent modules per interface) that can be changed simply and throughout the lifetime of the entire system is an adequate approach here. However, the possible existing infrastructure also must be taken into account, for example, if control systems of the production department can only be linked usefully via existing object linking and embedding (OLE) for process control (OPC) servers. A platform linked solution (in this case, Windows) must be used here. OPC technology or Active-X-Controls are only available in Windows.
- *Software services.* The use of Web services has become established for a service-oriented architecture. The server-based implementation can occur platform independently in the scope of a Web application service.
- *User interface/reporting.* A Web-based approach is often also platform-independent. However, the many advantages of a "true" Web solution (using HTML and JavaScript exclusively) are still offset by the disadvantage of a sometimes low level of user friendliness.

6. SCALABILITY

As is generally known, the only constant is change. If we follow this maxim, which applies somewhat for modern production, scalability is another important requirement of the system architecture of the MES in addition to platform independence. On the one hand, the system must be adjusted as precisely as possible to the requirements of the customer, and on the other, changes in the production structure, that is, both changes in the scope of functions and in the quantity structure, must be easy to map out in the MES.

Changes in the scope of functions generally arise from the introduction of new products or from new ideas on organization and the changed activities linked with them. Examples here are the introduction of a worker information system or the switch to group work with a changed wage system. Many system suppliers cover such functional expansions with independent software modules, which are offered as add-ons for the basic software.

Scaling as per the quantity structure is more difficult and also affects the software architecture. The following key data of the system should be taken into consideration with regard to scaling:

- Number of machines and workplaces
- Number of articles produced
- Number of measurement values taken/sample rate for the measurement values taken
- Number and frequency of reports created (e.g., disruption reports from production controls)
- Number of simultaneous users (i.e., client stations on the network)
- Number and calculation cycle of key performance indicators (KPIs) and quality data
- Archiving period for KPIs, quality data, measurement values, and reports
- Type and frequency of evaluations of data sets
- Number of interfaces and frequency of data exchange

These points affect the database in particular, whereby the *transaction volume* (effects on the CPU load and the interfaces of the system) and *data volume* (effects on the required memory capacity) aspects are taken into account. This means that both the processing power of the database system (usually in the form of additional processors) and the memory capacity (usually in the form of additional hard drives) should be suitable for flexible scaling.

Based on the suggested architecture, the second bottleneck arises with growing quantity structures in the application server. Here, too, scaling can occur through adjustment of the processing power (i.e., additional processors), but the individual processes (applications) of the system can be allocated to several processing systems. The option to allocate the processes to several systems is a true advantage for system running and maintenance as well. With the example of the suggested architecture, all components represented could be run on a common server in the simplest case. This architecture would lower the costs for a small system with small data quantities. At the other end of the scale, however, each of the software components represented could work in a separate server system. In order to be able to realize both extremes represented, the architecture must be appropriately flexible and as platform-independent as possible.

7. FLEXIBLE ADJUSTMENT VERSUS SUITABILITY FOR UPDATES

The only constant is change. Does this saying sound familiar? Correct—that was the introduction to the preceding section of this chapter. But this statement is just as valid for this section. Changes to real production also require changes in the MES. These changes should be implemented as quickly as possible with low financial and organizational outlay. Despite this, the MES should be a standard product, that is, suitable for updates, stable, and ensured for the future. Thus it applies that the competing requirements of high flexibility and stability must be united in one system. This can be achieved only with extensive and complex *parameterization options* and simultaneous *suitability for updates.* Suitability for updates refers primarily to all core functions of the system, which must remain the same independently of the specific application. But what does *suitable for parameterization* mean? Is this just a number of switches, system parameters, or user profiles— or are other mechanisms also required? This question cannot be answered globally but must be answered specifically for individual modules and functions of the system:

- *Interfaces.* Various methods and technologies, such as OPC, telegram exchange via TCP/IP, remote function calls (RFC), message queues (e.g., MQSeries or Com+), Web services, and database interfaces based on views, have become popular for the technical development of software. Here, a flexible system should support various technologies, and the partners must agree on one of those technologies. The data exchanged are, however, hardly normed and are subject to frequent changes. Therefore, flexible tools such as programmable scripts are needed for configuration of the interface content. In order to guarantee suitability for updates for scripts as well, it must be ensured that already existing tools remain viable in the case of an update to the framework (e.g., scripting engine).
- *Main functions.* The main functions of the MES, such as resource management, fine planning, and Machine Data Acquisition (MDA), should be available as *modules* of the overall system. This means that it is easy to carry out functional scaling. In the case of an update, individual modules also can be brought to a new status.
- *Partial functions.* Within these main functions are partial functions that can be activated or deactivated independently of their application. Here, there is the option of configuration on the basis of *system parameters.* These parameters must be saved in an update-secure manner to avoid unpleasant surprises after software updates.
- *Project-specific data processing.* Data processing, such as the calculation of KPIs, is also strongly influenced by customer requirements. The use of scripts that are easy to change for the system supplier or that even can be adjusted by the customer is a tried and tested method. Here, too, the condition applies that scripts must be suitable for update. In the case of a software update, then, only the framework for running the scripts is changed to a new software status and not the scripts themselves.
- *User interfaces for standard functions.* It should at least be possible to adjust these interfaces in their look and feel to suit the needs of the user. For example, it should be possible to adjust a given corporate identity with predefined colors and a company logo globally. Saving the settings, such as selection of columns for a table view, and the default saving of the table should occur user-specifically.
- *Customer-specific user interfaces.* In some projects, not all needs of the customer can be covered with standardized interfaces. The software concept should allow for the project-specific creation of interfaces for exceptions. These interfaces created especially for a customer also must function after an update to the entire system.
- *Visualization via black diagrams/flowcharts.* Although standardized visualization with a generic approach (e.g., mapping out all machines/stations with the most important order-processing data) saves projection outlay, it is often too inflexible. A freely "parameterizable" visualization solution with the option to provide process visualization increases both the flexibility of the MES and its acceptance by users.
- *Reporting.* Meaningful and optically pleasant reports are the calling cards of the MES for company management, and this is why a highly flexible reporting system is needed. A set of standard reports should be present in the basic interface of the system. However, it must be possible for trained users to change these reports and for the reports to be used as templates for the creation of the user's own reports.

Software that is adaptable and also suitable for updates is absolutely realizable with modern IT tools. However, this comes at the cost of performance because of the necessary complexity. A simple system with few parameterization and setting options seems to be less expensive at first glance. The limitations can be seen only in the course of implementation or when the system is in use, when the first extensions are needed. Then the seemingly cheaper system also can develop into a money pit.

8. SERVICE-ORIENTED ARCHITECTURE

The basic idea of service-oriented architecture (SOA) aims to organize business processes into individual services. The client calls up a service for a defined task (i.e., order to the service), this order is then processed through the server, and the result (i.e., response from the server) is sent back to the client. The structure of the service (i.e., data structure for order and response) is managed in a common repository. A unique address (i.e., the server that provides the service) exists for every order to which an order can be sent.

The best-established technological SOA approach takes the form of Web services. The World Wide Web Consortium (W3C) has carried out an extensive standardization of Web services and data exchange using the Simple Object Access Protocol (SOAP, a protocol for data exchange via HTTP and

TCP/IP) and therefore makes it possible to apply the technology in heterogeneous environments (Fig. 2).

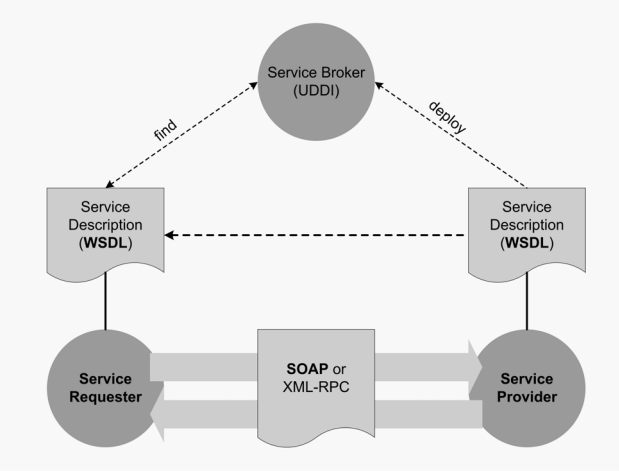

Figure 2: Concept of Web services with SOAP.

The "Service Broker" depicted in the figure is required for companywide or globally used services to inform an arbitrary client which server provides which services. These metadata, which describe a Web service, are exchanged with the aid of the Universal Description, Discovery, and Integration (UDDI) protocol. If these Web services are used only "locally," which is generally the case within an MES, or in connection with neighboring systems, the service broker can be omitted. The Web Service Description Language (WSDL) descriptions of the services are known to the clients in this case. The functions and the interface of a Web service are precisely defined in a WSDL file. With this information, the client can use the service provided. The data exchange itself is carried out via SOAP (see above; usually via HTTP, but other protocols are also possible) or via RPCs.

Using this architecture, it is possible to arrive at a situation where a *process* and the related data are mapped out only *once in the company's IT system,* and the software functions are still made available to all users in their specific context. Thus the desired integration of applications and production data is achieved.

For example, internal interfaces such as the connection of MDA (Machine Data Acquisition)/PDA (Production Data Acquisition) terminals to a server can be implemented in a simple, flexible manner with Web services. Another example is the handover of attendance records for workers to the MES from a staff timekeeping record. Using these data, the MES can plan the resources that are actually present or carry out a plausibility test for order responses related to workers. By querying the data using an ID, this also can be rendered anonymous in the MES. In this example, the staff timekeeping system is the server, and the MES is the client for the Web services.

9. CONCLUSION

At the beginning, it was described two general approaches for the architecture of an MES whereby a suggestion, namely, the database-centered approach, is subsequently examined in more detail. Here, the central components of the system were explained. Essential characteristics of a modern MES, such as platform independence and scalability, were explained. The basis of innovative communication mechanisms, such as the OPC UA, is a service-oriented architecture (SOA). This approach is also valid for the architecture and communication mechanisms of MES.

10. REFERENCES

[1] International Society of Automation (ISA): http://www.isa.org.

[2] Manufacturing Enterprise Systems Association (MESA): http://www.mesa.org.

[3] Meyer, H. et al.: Manufacturing Execution Systems (MES) : Optimal Design, Planning, and Deployment. 1. Edition, McGraw-Hill, New York: 2009.

[4] Schaefer, M. et al.: MES – Anforderungen, Architektur und Design mit Java, Spring & Co. 1. Auflage, entwickler.press, Frankfurt am Main: 2009.

[5] Thiel, K.; Meyer, H.; Fuchs, F.: MES – Grundlage der Produktion von morgen – Effektive Wertschöpfung durch die Einführung von Manufacturing Execution Systems. 1. Auflage, Oldenbourg-Industrieverlag, München: 2008.

[6] Ricken, M.; Vogel-Heuser, B.: Engineering von Manufacturing Execution Systems. SPS/IPC/Drives Kongress, Nürnberg: 2009.

[7] Scholten, B.: MES Guide for Executives: Why and How to Select, Implement, and Maintain a Manufacturing Execution System. ISA, Durham: 2009.