Analysis of Data Storage Technologies for the Management of Real-Time Process Manufacturing Data

Overview

This Information Technology white paper describes fundamental issues that are important to the design and implementation of a modern information management system for the manufacturing enterprise. It compares the use of traditional data storage techniques, relational database management systems (RDBMSs), and modern process data historians for the capture, management, and use of real-time process manufacturing data. Appendix A describes how to evaluate the features and performance of a process data historian.

Why Collect Process Data?

Operations personnel, process engineers, laboratory technicians, and management all have a need to examine not only what is taking place in the manufacturing process in real time but also what happened in the past.

These users need access to actual process data, not just summary data, to analyze process operations and make effective decisions.

In addition, many "near real-time" applications such as supervisory control, advanced control, statistical process control, data reconciliation, expert systems, neural networks, batch and recipe management, and other manufacturing execution system (MES) functions require plant historical data. Again, the need is to process and analyze actual data, not just summarized data, over a long period of time. This set of requirements led to the development of the process data historian.

Of course, the historian can also record summary data (for example: what was produced, how much was produced, and how much raw material and energy were consumed in the making of the product) for use in accounting and planning and by other business (transactional) applications. But the true "acid test" is the acquisition and long-term maintenance of the large volumes of actual process data generated by most manufacturing facilities.

Sources of Data

Most real-time manufacturing process data consists of continuous analog data representing parameters such as temperature, flow, pressure, weight, composition, etc., plus discrete or digital data, "event data" expressed as time-stamped text strings, and laboratory data in either analog or text form. Typical data sources include distributed control systems (DCSs), programmable logic controllers (PLCs), remote terminal units (RTUs), laboratory information systems (LIMS), and manual data entry.

Data Acquisition

Two factors are important in the acquisition of the data:

1. The sampling rate, which is the frequency at which the data is scanned and read, usually at least once per minute and often as fast as once per second

2. The resolution of the time stamp attached to each data value, usually to the nearest second (though resolutions of up to one microsecond are available)

More often than not, the communication capabilities of the data source are the limiting factor in the data's acquisition rate. The clock function of the host computer's operating system, together with the design of the particular application, sets the limit on time-stamp resolution.

Data Storage Methods

There are three basic approaches to data storage and management (a discussion of each follows):

- "Traditional" data storage techniques developed in the first process computer applications
- Relational database management systems
- Process data historians using data compression

Traditional Data Storage Techniques

Traditional data storage techniques are those that originated in the early days of process monitoring whereby raw analog data values were stored upon each acquisition scan cycle and placed in flat- file databases. Storage technologies of the 1950s through the early 1970s relied on a mix of either slow-speed, high-volume media like magnetic tape, or high-speed, low- volume media like magnetic core memory or dynamic memory. Acquisition of the process data was limited by the access speed of the equipment and the resolution of the signal conditioning circuitry (e.g., 8-bit AIDs). Thus, the acquisition and storage of large amounts of data was often technically limited as well as cumbersome and expensive to maintain.

Several approaches were developed to deal with this situation, the simplest being to acquire the data less often, meaning a slower sample rate of, say, once every minute or every several minutes. As a result, less total data is acquired and more time is recorded per unit capacity of storage media.

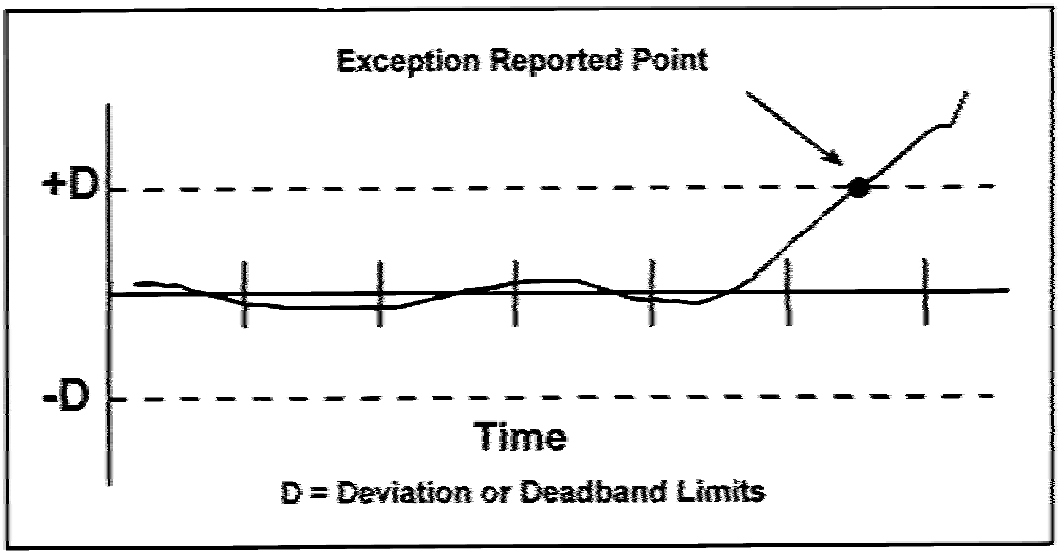

Figure 1: Exception Reporting

But, unfortunately, accuracy also is sacrificed, since not all the data is recorded. Faster data acquisition combined with storage of only the calculated averages also is a way to attempt to represent the actual data accurately yet still avoid storing each piece of raw data.

Both of the above approaches fall short in that they can miss recording key data values or misrepresent significant changes in the data. Upon reviewing the recorded data, the user often cannot tell precisely what occurred because of missing data or because averaged data either exaggerates or mitigates what actually happened. Neither method can reproduce the "true path" or trajectory of the actual raw data well, especially if the data is highly variable, as is often the case in industrial processes.

A third approach is to *filter* or *exception report* the data, that is, to record only those data values that exceed a prespecified data tolerance level or dead band, expressed as a percent of span (range) or as a fixed tolerance.

(For example, ±:2 psig for a 0 to 400-psig-pressure range). This method does a better job of "following" or tracking the actual raw data. However, it also has shortcomings, as will be discussed later.

Relational Database Management Systems (RDBMSs)

Most database packages in use today are based on a relational model. There are many popular suppliers of RDBMS software in the market, including Oracle, Informix, Sybase, IBM, Microsoft, etc., but all share common characteristics. These characteristics form the basis for the following discussion.

The relational model, based on the work of E. F. Codd in the early 1970s, has three major components: structures, operations, and integrity rules. Structures are well-defined objects that define how to store the data of a database. RDBMS systems have both a logical and physical structure. The physical structure is determined by the operating system files that constitute the database.

The logical structure is determined by the table spaces and the database's schema objects, which include the tables, views, indices, clusters, sequences, and stored procedures. Typically, one or more data files on the physical disks of the computer are explicitly created for each table to physically store the table's data.

The logical structure of the RDBMS defines names, rows, and columns for each table. The columns generally have data types such as character, variable length character, number, date and time, long, raw, long raw, row ID, etc. Once the table name and columns have been defined, data (rows) can be inserted into the database. The table's rows can then be queried, deleted, or updated. Note that all rows and columns of data are equally important, which makes it easy to ask queries like "How many apples were sold on Tuesday?" Neither the number of apples sold nor the date/time takes precedence.

Data is retrieved from the tables using Structured Query Language (SQL). SQL statements query the database by searching the table directories for the logical columns and rows that contain the specific data of interest and then returning the results. A procedure is a set of SQL statements grouped together as an executable unit to perform a specific task, for example, to retrieve a set of data with common characteristics. Stored procedures are predefined sets of SQL statements that can be executed together.

Indices are optional structures used to increase the performance of data retrieval. Just as an index in an instruction manual helps one locate specific information faster than if there were no index, a database index provides a faster access path to table data. Indices are logically and physically independent of the data. Procedures and indices can be combined to speed up the access, particularly when the most recent portions of the database are stored in a computer's RAM.

Historians and Data Compression

The development of data-compression techniques that record points at un- equally spaced time intervals stemmed from three factors. First, the need to access ever-increasing amounts of data by various users created the demand. Second, the advances in minicomputer technology provided the processing power to enable collection of the data. Finally, the recognition that there are long periods of operation in which variables are either constant or moving in a predictable path inspired the solution. The first commercial use of the data compression technique was based on what is known as the "boxcar with backward slope" compression method, first published in 1981 by Dupont's John Hale and Harold Sellars and subsequently offered in a commercial data historian by Aspentech (IP 21).

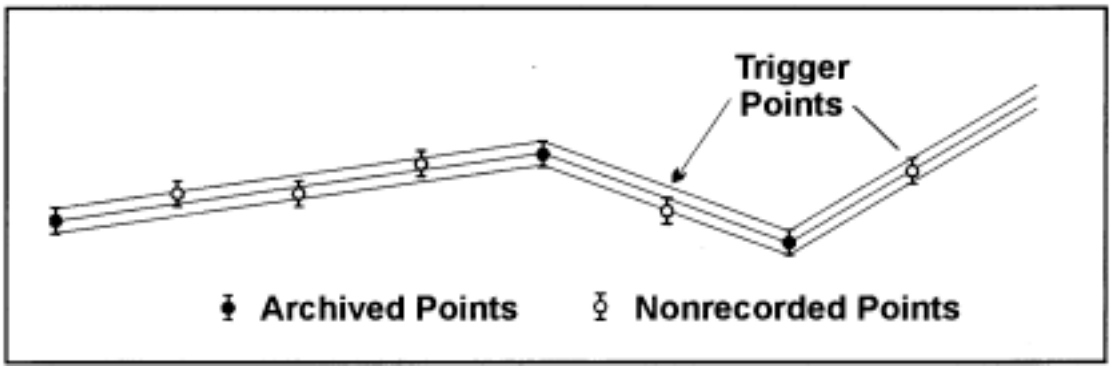

Figure 2 : Dynamic Slope Envelope of Data Compression

In this method, the raw analog data is read and combined with a user- specified tolerance value to produce a rectangular box-like shape or "dynamic slope envelope" around the

actual data, as shown in Figure 2. Mathematically, a set of algorithms is used to calculate the slope, taking into account the raw data value, the tolerance specified, the previously recorded data values, and the time. This "envelope" is projected ahead and used to determine which points are to be archived, resulting in a time- series-based mathematical representation of the actual data. When data is recalled, the algorithms' times and slopes are used to quickly access the compressed data and accurately interpolate it, producing a data trend over the desired time span that represents the original behavior within the accuracy specified by the tolerance.

Data compression is thus a storage methodology that uses the predicted behavior of a process parameter to reduce the actual raw data values to a data subset, which can then be later expanded and interpolated ("decompressed") to produce a representation of the original data accurate to within the data tolerance level specified. A historian based on this technique can store data over long time periods using the minimum storage space.

Comparison of Technologies

When capturing and managing real-time data, four factors are important:

1) The data storage efficiency, or how much disk space is used to store the captured data

2) The precision of the captured data and the ability to accurately recreate the raw input data

3) The speed at which the data can be written to the database

4) The speed of access to the data

The relatively inexpensive hard drives available today make storage efficiency appear less critical. However, the movement toward digital instrumentation and Field bus communications will result in a significant increase in the volume of data available to control and information systems for processing. Hard drives may indeed be inexpensive, but they are not free -and they still require a commitment of precious staff resources for data management and hardware maintenance.

Base Case: Data Compression

As mentioned above, a historian based on data compression can store data over long time periods using the minimum storage space. Accuracy is ensured because the compression algorithm selects only those points for storage that are necessary to reproduce the actual data within the tolerance specified. No further data reduction or file compression techniques are necessary to reduce the storage space. Accurate statistics on the data (averages, standard deviations, accumulated totals, minimums, maximums, etc.)

within a specified time span can be readily generated using proven mathematical techniques for time series data. "Gaps" in the data can be easily identified and highlighted.

Data Compression Metrics

There are two methods of measuring the efficiency of data compression. The first, the compression ratio (CR), is defined as the number of values processed at equal time intervals that result in a single recorded value. The second, the mean time to record (MTTR), is the average length of time between recorded values

For a system processing variables at one-minute intervals, a 30: 1 data compression ratio means that each variable is being recorded once each half-hour on the average. Therefore, the MTTR is one half hour. Note that the Compression Ratio can be increased for a given variable without improving storage efficiency by simply increasing the processing rate, while the MTTR is insensitive to this factor. Hence, the MTTR is the better comparison metric.

Performance of Traditional Storage Techniques versus Data Compression

Data values stored sequentially in one-dimensional flat-file databases have extremely slow access rates for historical values; the older the data is, the longer it takes to go back from the present time and retrieve it. Further- more, once the file is full, it becomes "locked" and cannot be altered with new data or corrected values. The net results are huge media storage requirements and substantial inflexibility in handling the recorded data. Amazingly, many existing human machine interfaces (HMIs) and supervisory control and data acquisition (SCADA) systems still rely on flat-file storage.

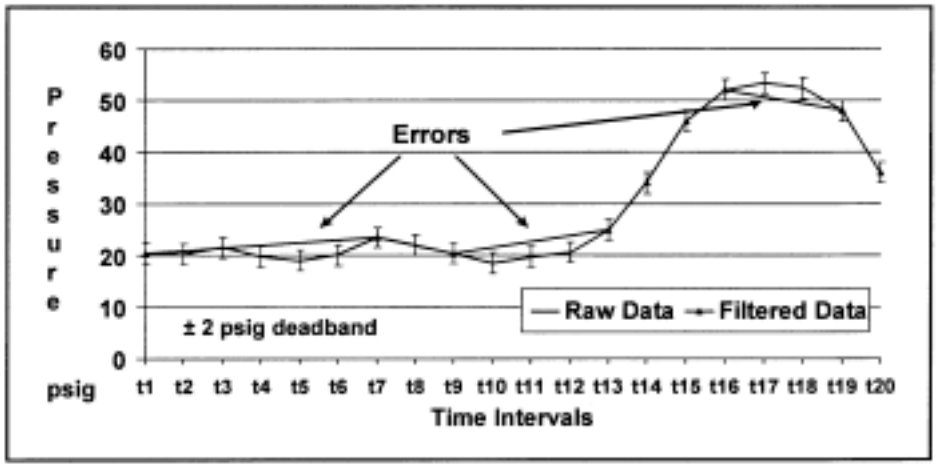

Figure 3- Exception Reporting Errors

How does the addition of data filtering, more commonly known as exception reporting (which is actually a primitive form of data compression), affect the performance for a traditional flat-file approach? Attempting to restore the exception-reported values and represent the actual data means relying on linear or step interpolation between the recorded points. Figure 3 shows that a simple linear interpolation between the recorded points produces errors that often exceed the tolerance level originally specified. A step interpolation will also lead to errors. Now the compression ratio or MTTR varies with the tolerance specified and the variability of the data as well as the method used to record the data. Exception reporting can produce compression ratios of up to 20: 1 for moderately variable process industry data. A ratio of 10: 1 is typical.

Performance of RDBMSs Versus Data Compression

RDBMSs address data storage by creating a relationship between the data and its associated time stamp such that the search for the data is easier to make and faster than sequential searches. RDBMSs do allow for much more complex relationships to be maintained and queried because of the cross-referencing of the tables and rows, combined with indices, procedures, and other advanced SQL querying and data manipulation techniques.

Thus, RDBMSs are highly desirable for analyzing non-time-critical data such as the aforementioned production summary data and batch and recipe analysis, where non-time relationships between variables are of key inter- est. They are obviously superior to flat-file databases. As a result, many HMIs and SCADA systems are migrating to RDBMSs for data storage. Exception reporting is the predominant method of data compression used combined with recording averages.

However, RDBMSs have several important disadvantages compared to modem data compression-based historians:

- The RDBMSs carry substantially more overhead due to their multiple- row and table record structures. The use of indices, clusters, and procedures adds even more overhead by slowing down the performance considerably.

- All RDBMS records are equally important; they are not optimized for time

- RDBMSs have no inherent data compression method; they are usually combined with exception reporting and averaging techniques, which may result in data loss and inaccurately, reproduced data.

- The speed of writing to RDBMSs is quite slow compared to data compression-based historians. RDBMSs often talk of very high transactions per second (TPS). This refers to actions taken on the data once it is already in the database, not to the speed at which it is written to the database nor the speed of data retrieval. F or example, one popular HMI vendor states that it can make 3.5 writes per second to Microsoft Access and 18 writes per second to Microsoft SQL Server using ODBC. Those rates are extremely slow compared to the modem data compression-based historian ' s ability to write literally thousands of points per second to its database

- Data statistics are not automatically calculated by the RDBMS. (SQL mathematics is limited to sums, minimums, maximums, and averages.)
- RDBMSs are typically limited to a one-second-time resolution, which is problematic for handling high burst quantities of data with sub second time stamps.

- RDBMS retrieval speed, although faster than that of flat files, is much slower than that of a data compression-based historian, which accesses data extremely fast- nearly instantaneously -and can calculate the statistics for a given time span "on the fly" without any SQL programming.

In order for the RDBMS to minimize storage space, the exception reporting dead band must be widened and/or the data capture rate must be slowed down and/or only averages stored. Widening the dead band, slowing the data capture rate, and/or only storing averages all result in discarding information about the manufacturing process. The data ' s variation, especially fast "spikes," are not captured, and so the corresponding ability to troubleshoot and optimize the manufacturing process is lost.

New Developments in Historian Technologies

The data historians of the 1970s and 1980s typically operated on relatively expensive minicomputer platforms using operating systems that required skilled engineers and programmers to use. The advent of personal computing (PC) technology in the 1980s, combined with more powerful, less costly microprocessors and client/server computing architectures, plus Microsoft's dominance in the PC software industry, has led to an evolution of the modern data historian.

This evolution has resulted in two approaches to date that attempt to offer improvements in data historian technology

1. The first approach offers a combination of either interval processing or the exception reporting data compression method, combined with file compression,

married to modem PC-based client/server RDBMS technology. This approach is referred to in this paper as the Hybrid RDBMS

2. The second approach builds on improvements to the original boxcar- back slope algorithm, combining it with object-oriented technology, true distributed computing, and other client/server technologies. This method is used in the Aspentech's IP21 RTDB.

Both approaches are designed to take advantage of Microsoft's 32-bitWindows 95 and NT operating systems and associated technologies.

Hybrid RDBMS

The Hybrid RDBMS approach consists of the addition of virtual table ex- tensions to the Microsoft SQL Server relational database for the handling of discrete and analog data, combined with standard relational tables for events, text, and summary data. The data is acquired by scanning the data source, then transferring the data locally or across a network to the Hybrid SQL Server. Exception reporting processes discrete data, while analog data either may be sampled at fixed intervals (also called cyclical) or processed by exception reporting. Millisecond time stamp resolution is supported, although RDBMSs normally are time-stamp limited to the nearest second. The virtual tables are stored in RAM for fast processing.

With fixed-interval sampling, every analog data point at the sampling frequency is stored, so there is no data compression. But no time stamp need be stored since the interval is fixed and the time stamp at each interval is easily calculated based on the interval specified and retrieval time span. *Exception reporting*, with a specifiable dead band, results in analog data compression with a compression ratio or MTTR of the same magnitude as previously discussed. Again note that the compression ratio varies with the sampling frequency.

The virtual tables for both analog and digital data typically contain only 24 hours of data prior to being either overwritten or stored to hard disk. When storing to disk, there is an option for compressing the data file. Interval processing and storage without the time stamp saves considerable storage space. With optional file compression even more space is saved. The combination of exception reporting (with the time stamp) and file compression also results in reduced data storage requirements. The maximum space reduction reported is a factor of 50 less than a standard relational database.

Stored data is accessed by means of standard SQL language calls and pre- defined storage procedures, which allow a wide variety of applications to acquire the data easily. Decompression and file expansion are handled at the server. Retrieval speed of SQL

varies but is not known to be exceedingly fast. Again overall performance is a function of a variety of factors.

Aspentech's RTDB IP 21

Aspentech further modified its original algorithm to improve upon the boxcar-back slope method and named the new algorithm "adaptive compression," because the slope envelope (refer to above description of boxcar with back slope) is adaptively modified based on the statistical behavior of the variable. The result is even faster data recalls, higher accuracy, and lower overhead.

Aspentech also added event-driven processing, data compression at the source with "store-and-forward" technology, millisecond time-stamp resolution, and decompression at the client. Processing on event means data points are no longer scanned and processed on a fixed periodic basis (unless forced to scan) but, instead, only when they change. Compression takes place at the acquiring, minimizing the loading on the server node. Some applications, such as sequence of events and machinery monitoring, require an ultra fine resolution time stamp; the millisecond time stamp resolution meets this need. Finally, data is decompressed at the requesting client CPU, thus lowering the network load.

Performance Comparison

Hybrid RDBMS

The Hybrid RDBMS performance is very dependent on how the system is configured to handle the data in addition to the data variation and dead band specification itself. The system is inherently more complicated due to the multiple file structure and data types that must be given consideration when implementing the system.

Disk space is indeed saved in using the time internal option, because a time stamp is not stored. However, this savings is offset by storage of the data at every interval, even when the data does not change. This is not true data compression, unless the file is later compressed.

As previously pointed out, accuracy problems are associated with exception reporting. Its M1TR will not be nearly as high compared to that obtained by the boxcar back slope or adaptive compression algorithms.

The use of file compression with short file time spans (e.g., 24 hours) means that numerous files will have to be accessed, expanded, and searched in order to assemble data for long time spans. This means that for short-term history and low data volumes, the Hybrid RDBMS approach will yield good results. If longer file times are used, fewer files will have to be opened but at the price of longer search times within each file. In

either case, the larger the amount of data and the longer the time frame, the worse the performance will become. True, the data storage volume can be reduced by a factor of 50 compared to a standard relational database, but data storage reduction alone is not the issue. The Hybrid RDBMS is simply not designed for managing high-volume process data where guaranteed data accuracy, minimum storage space, and high performance are all paramount

The performance of Aspentech's adaptive data-compression algorithm builds on proven concepts by allowing access to years-old data just as quickly as to data only 15 minutes old, because data within each month can be accessed without decompressing the entire month. And, further- more, the monthly files can be seamlessly spanned without the need to join the data from adjacent or disparate months. Aspentech test results yield write speeds to the Historian that exceed thousands of values per sub second, while data reads can be as high as several thousand values per sub-second across a network. Compression ratios have been measured to range from 10: 1 to as high as 100: 1 depending on the variability of the incoming raw data. Typical ratios are 40: 1. Such performance permits the amount of physical storage space needed to be minimized yet enables the data to be delivered when and where it is needed amazingly quickly.
To provide maximum flexibility to the user, IP 21 database provides access to the time-series data via OLE Automation, DDE, ODBC/SQL), containing application programming interfaces (APIs). Calls to the time-series data- base are answered in the format in which they are called; for example, SQL calls to populate an Oracle table with production summary results are answered as if the IP 21 RTDB were an RDBMS.

Conclusion

It is not unusual for large manufacturing enterprises to be capturing, man- aging, and providing access to tens of thousands of process and related data values that are acquired at rates from milliseconds to a few minutes, yet stored online for two to three years. The modem adaptive compression-based process data historian is specifically designed to excel at that task, while using minimum storage space and providing open connectivity. Competing technologies, including both standard and hybrid relational databases, have improved but still are not up to the task because of inherent limitations in their structure and data handling. While the use of exception reporting results in a savings in storage space, it cannot provide the same efficiency or accuracy as the more modem data-compression techniques.

Notes:

I. Nyquist's Theorem states that in order to accurately reproduce a given frequency, it must be sampled at least twice as fast.

2. The RDBMS discussion is based on information provided by Oracle@ Corporation and the Oracle 7 Server Concepts Manual.

Appendix A: Evaluating a Process Data Historian

Overview

Appendix A provides a guide to evaluating a process data historian to determine the best fit for a manufacturing information management system. A "data warehouse" historian should fit easily into any computing architecture and excel in functionality and performance. The following checklist and testing outline serves as a guide to selection.

Architectural Considerations

The real power of modem computing technology is unleashed by software designed to be distributed across a network, using the client-server model to create a division of labor among the various modules. When evaluating this aspect of a historian, the following questions should be asked:

Is the architecture truly client/server, i.e., is the data stored in a server that can be physically separate from all client modules (viewers, gate- ways, applications)?

Can multiple servers be seamlessly integrated into the network?

Can servers or data sources (gateways) be moved from node to node without requiring changes to existing applications?

Are network communication issues handled internally in such a way as to shield the user from having to worry about them?

Are standards-based (OLE, ODBC, etc.) means of accessing the data in the server provided, and are these access tools implemented with distributed architectures in mind?

Does the design take into account multiple operating systems within the network and the potential for communication between applications running under different operating systems?

Is the potential for network downtime allowed for in the design such that the potential for loss of data is minimized or eliminated? More specifically, does the design provide for server redundancy as a configurable feature and for client store-and-forward capabilities through the provided software development kit (SDK)?

Is the server database truly open, i.e., can any application store to or retrieve from the server in a straightforward manner (subject to security restrictions), either through an SDK or a standards-based bridge component?