



Master Data Management System architecture

By Pritesh Desai and Ahmad Damra

Master Data Management or MDM initiatives are often considered very complicated and, sometimes, even elusive. One would not be entirely wrong in thinking that the architecture and design for a complicated MDM system should be fairly complicated. However, one would also be accurate in saying that the architecture and design of a complex MDM system is not very different from that of any other complex system. As you read on, you are likely to encounter concepts you may already know but you will certainly encounter a difference in how they and apply and why they apply.

Types of MDM

Before we get right into MDM architecture and design concepts, it is important to take a moment to understand the different types of MDMs that can be built. Essentially there are three types of MDMs, two primaries, namely Repository Type and Registry Type and a hybrid which we shall refer to simply as Hybrid Type.

The **Repository** approach involves storing the entire data set for an enterprise in a single database. When we say the entire data set, we mean all the data attributes used in any and all applications across the enterprise. From a design perspective this is as involved as the complexity within the organization and requires that all the applications functions that interact with the database need to be modified to read the one single MDM database.

In a retail context, an example of this might involve the same customer Mr. Joe Smith who transacts business with both the Credit Card and Mortgage lines of business. In a Repository approach, both these would be unified to a single Customer in the MDM with all the attributes being brought across from the Credit Card and Mortgage applications. The source applications are, at the same time, modified to read and write data to the MDM system rather than the source system databases. A clear advantage of this approach is in retaining and maintaining only one record of any one thing within the enterprise, since all applications use a single copy of the data. There are none of the issues

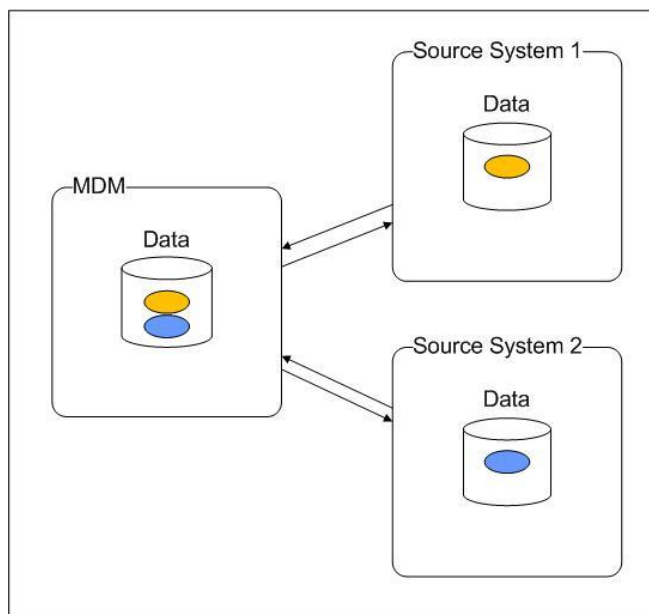


Figure 1: Repository

associated with synchronizing multiple copies of the same information across different systems once the source systems have been modified. This particular model might also be referred to as an “Ideal State” for MDM.

The **Registry** approach, on the other hand contains only the keys to the attributes pertinent to the customer, product etc., the actual attributes being stored in their source systems. Since all the data attributes continue to be stored in the source systems, there are minimal changes required to the source systems, basically to translate source system IDs to Registry ID and vice versa. While there are fewer up front changes required in implementing this model this is counterbalanced by the processing overhead from each MDM query requiring one or more source system queries to obtain the full scope of attributes.

Using the same example as before, the Mr. Joe Smith would continue to be recognized

separately in the Credit Card and Mortgage applications. However, it would become possible for each of the lines of business to be able to relate the customer to the other line of business and therefore potentially up-sell that customer or better manage the risk exposure from that customer.

The third approach is the **Hybrid** approach. This approach recognizes that it is often impractical to change every single system in an enterprise to use a single database and data model within that database, whether it's a matter of inability to justify business value from making such widespread changes or just a matter of the complex logistics required for such an exercise. This approach takes a more pragmatic view by storing the keys to the same record in the different source systems as in the “Registry” but also storing some critical attributes that can be used to identify a unique record and describe it to a reasonable degree.

Taking the earlier example, this can mean storing the keys to Mr. Joe Smith in each of the source systems, as well as key identification attributes for the customer. As well, the MDM

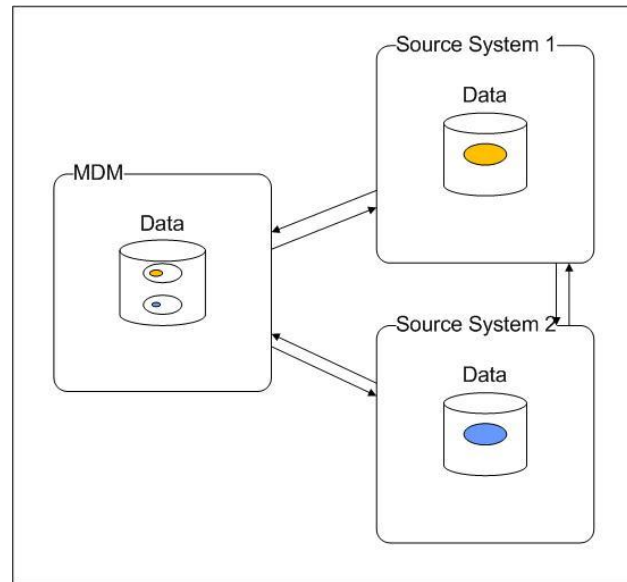


Figure 2: Registry

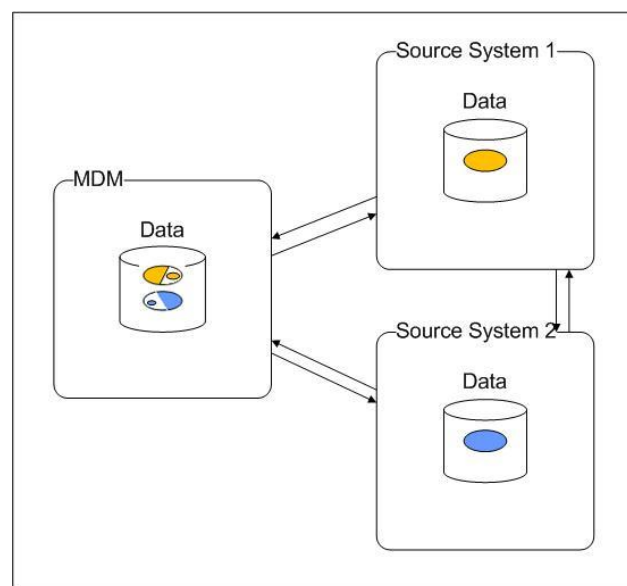


Figure 3: Hybrid

might store some descriptive information about the nature of the business relationship between Mr. Joe Smith and the enterprise, such as the types of services consumed, or the billing amount for the last few months. The precise attributes to be stored in the MDM would really depend on the intent of the MDM and may be expanded over time. The challenge naturally is in maintaining any attributes that are stored in the MDM as well as those stored in the source system, ensuring that any duplication checks etc. are performed each time. The maintenance of information in the MDM is compounded by standard database issues such as update conflicts, latency etc.

All the disadvantages notwithstanding, the Hybrid approach is perhaps the one that permits the most flexibility for organizations of differing sizes in terms of gradually scaling up the MDM rather than trying to implement the MDM using a “Big Bang” approach. *(See other white papers on this site for more information on the theme of implementation approaches)*

Real time capability

It is easy to think of an MDM as a large data warehouse – though that would be inaccurate because while Data Warehouses, by definition are not designed to be accessible real-time, an MDM does need to be capable of real-time operational use. What this means that source systems throughout the enterprise should be able to query and update the MDM in real-time, in the midst of their transactions. For instance, while the Mortgage system is being used to process a mortgage, it should be possible for the mortgage system to call out to the MDM to identify the customer and obtain the customer’s global identity which the Mortgage system would use when creating a record for the customer in the Mortgage system and thereby making it easier for the Mortgage system to interact with other systems within the enterprise in future. Furthermore, the Mortgage system may update the MDM with current information about the customer; thereby adding value to the MDM. Clearly, this is a symbiotic relationship but the key point here is that all this is expected to be performed real-time and therefore calls for real-time capable MDM architecture and design.

Data Model

Since the most important aspect of an MDM is common data, a single data model for the MDM database is critical to the success of the MDM. The development of the MDM model should involve a mapping from the source systems / applications to ensure that key attributes that have been identified as candidates for the MDM are included. Since there are multiple source systems, some design choices are typically necessary to accommodate differences within reason. For instance, Country may be stored as a free form text field in a few source systems and as a code-value pair in other system but you would design the MDM data model to store a standard code-value pair, which might necessitate mapping data from different source systems differently to achieve the same standard outcome.

MDM Design Independence

There is a lot to be said for making an MDM model specific to the nature of business that an organization transacts. However, that line of thinking should not be extended to mean that the MDM model should start to look like any one of the source systems from which the data is derived. The data model for the MDM should, instead be based on industry best practices for data modeling as well as the different needs the organization has in relation to the MDM being designed. For instance, a source

system might store a customer's address in 3-4 lines of text, whereas, there may be reporting or transacting needs that require the address to be broken down into a Canada Post or US postal address format or even both formats.

Flexibility and Extensibility

Systems across an enterprise are not going to remain stagnant over the course of time. Specifically databases undergo slight changes from time to time and occasionally are significantly changed to better suit business need. Any changes to source systems can imply changes to the data transfer/query processes and potentially the MDM. MDMs must be designed in such a way as to facilitate changes to both these technical elements, within reasons. It is important to remember that all architectural and design criteria are interrelated and an overemphasis on any one could be detrimental to another. For instance, if the MDM database were designed in a completely configurable manner, it could become too cumbersome or unwieldy to be managed as an efficient real-time database.

An MDM could start out as a collaborative effort between two lines of business; say Mortgage and Credit Card. The Banking line of business may decide to join the collaboration at some point in the future. The MDM should be flexible enough to allow the extension of its use by other lines of business or other applications within an existing line of business.

Service-oriented Architecture

The MDM system is essentially a service provider to the source systems and should be designed as such. The service provider should provide an open interface using service-oriented architecture. This enables any external system to use the MDM using standard technologies and goes a long way in reducing integration issues. Most good MDM products come prepackages with a web services layer to access the MDM data model using standard protocols and methods. It's generally a good idea to ensure that this service layer is extendable through the addition of new services for increased efficiency of complex queries against the MDM database.

Open Architecture

The MDM system should be based on standards and technologies that are broadly accepted in the industry. While it is quite possible that MDMs are built using technologies from several different providers / vendors, the technologies should generally support interoperability with existing systems in the organization as well as systems that may be acquired in the future. While it's not fair to expect any system to support all future unknown technologies, if the MDM application is built using widely accepted open standards and frameworks, one can reasonably count on the existence of good alternatives that also support these same widely accepted standards. The adoption of widely accepted open standards and frameworks should be throughout the MDM application not just at what the MDM vendor thought were the interface points with proprietary stuff in between.

Security, Privacy and Compliance

In an MDM, like all other applications and systems, security, privacy and compliance are important. However, in the case of MDM, privacy takes on a slightly magnified meaning because there may be regulatory reasons why certain attributes about a customer cannot be shared between different lines of

business. Oft times, even the fact that a customer transacts with one line of business may be protected by privacy from being communicated in any way to certain other lines of business. Furthermore, each line of business may have compliance requirements that they need to meet and which the MDM should allow.

Typically, different MDM solutions permit different levels of control over the information stored within the MDM. The critical aspects are the ability to control access based on users or groups thereof, the ability to enforce privacy of data by user group and the ability to record an audit trail when data is accessed – and more importantly when data is added, updated, or deleted. Typically, each source system will take care of privacy at the consumer level, and that is not something that the MDM should be concerned with. For instance, it is possible that within the Mortgage line of business, the customer is not permitted to see confidential information that the organization retains for internal processing. That sort of information would be available to the source system from the MDM and the source system would be expected to take care of what can and cannot be seen by their customer.

Enterprise Architecture

There is such a thing as an architecture designed to support an enterprise. While most architecture will support a smaller size organization, supporting a large enterprise can involve some architectural finesse and the use or ability to use enterprise-grade technologies in the arena of messaging, middleware, application server, enterprise service bus etc. is important and must be incorporated. The converse is also true because the MDM should not be designed in such a way as to “price it out of the market”. In other words, the MDM should be able to support a smaller implementation, whether it is the infancy stage within a large enterprise or in a generally smaller organization. And the MDM should also be able to grow through the incorporation of the right technologies to support a much larger organization consisting of numerous lines of business and applications.

More specifically, the MDM application should be able to make use of existing technologies in use within the enterprise. For instance, the enterprise may have a Messaging and monitoring technology that was purchased from a different vendor. The MDM application should be able to adopt that existing technology when required rather than being tied into a proprietary technology, the acquisition of which may become a prohibitive obstacle at a later stage in expanding the use of the MDM application within the enterprise.

Incremental Implementation support

Any MDM application should support an incremental implementation approach. *(See other white papers on this site for more information on the theme of implementation approaches)* The reason for this is simple. In the beginning, it is likely that the enterprise wants to evaluate the MDM application in a contained setting. That may involve 2-3 applications and that too within a single line of business. That alone could require a fair amount of investment of resources and funds. If the contained evaluation proved to be a success, the organization would typically want to proceed to a larger implementation. If the MDM architecture did not support an incremental implementation approach that could mean the entire contained implementation must be scrapped and a significantly larger initiative started up to get

the entire enterprise to use the MDM application. That sort of “wasteful” management is typically frowned upon in most organizations.

Design for performance

MDM databases will hold a lot of data – no really – the data they hold will grow. The growth may be gradual or rapid but there will be growth, of that you can be sure; growth in the volume and distribution of data, number of users etc. While it isn’t always possible to foresee all forms of growth, it is possible to accommodate the obvious types of growth. For instance, in the database, this could take the form of the right balance between normalized and fully de-normalized model. And, in the application, it could take the form of performance-tuned code in key areas. It is not unusual to hear stories about databases that were designed to handle a certain volume but wherein the system proved to be so useful that the volumes tripled within the first year. It is imperative that your MDM does not suffer from such myopic design.

Without dwelling on it too long, it’s important to also note that there is such a thing as over-design. Performance is important, but not so much that everything stalls while performance meets an arbitrary standard. For instance, if the expectation is that the MDM will need to handle full load in 5 years, it’s quite reasonable, in a technical sense, to design for mid-peak-load at first, identifying the performance limitations up front.