

The SOA Magazine

Feature Article



Relating Master Data Management to SOA

by Chris Madrid

Published: April 15, 2008 (SOA Magazine Issue XVII: April 2008)

[Download this article as a PDF document.](#)

Abstract: Distributed application development has proliferated business data throughout the enterprise. Customer information is frequently spread across financial information systems (FIS), enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, sales automation systems, Web applications, and interactive voice response (IVR) systems. Complex enterprise application integration (EAI) solutions have been erected to keep these data silos synchronized. The resulting web of dependencies has adversely affected enterprise agility.

Many enterprise architects assumed that by merely adopting SOA these complexities would be magically abstracted and a method for seamlessly aggregating this data for consumption would manifest itself by an actionable interface like a call center application or a customer self-service portal.

However, it turns out that for many large IT operations challenges such as variations in data quality, differing primary keys, and multiple systems of record have created a situation where real-time aggregation of data is impractical. Master Data Management (MDM) has emerged in the last few years as a potential solution for removing this barrier. This article discusses what MDM is, how it is related to SOA, and how they can be used together to deliver agility even in the most complex environments.

Introduction

Starting in the 1990's, internal line-of-business applications began moving from two-tier architectures to more distributed n-tier architectural models. The goal was simple: abstract the complexities of the data layer from the presentation layer to achieve agility. Data models could now evolve to support new features and business functions without requiring massive rework throughout the application. Seeing this benefit, architects quickly realized that they could provide the same abstraction to business logic and began shifting more and more logic from the presentation layer to the application layer (Figure 1).

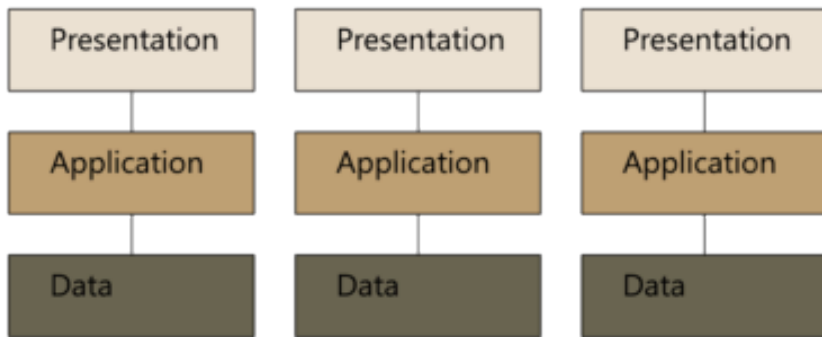


Figure 1: Three different distributed solutions, each establishing its own silo.

Vertical line-of-business application silos soon littered the IT landscape. These silos commonly duplicated business logic and data, leading to wide-spread redundancy and unnecessarily bloating the IT enterprise.

Seeking to alleviate this burden, distributed technologies of the day (DCOM, RMI-IIOP) were looked upon as a means of enabling reuse. Although this approach sometimes worked well with similarly architected systems, it lacked the reach to be a true enterprise solution. Realizing the technical challenges of sharing the business logic across a heterogeneous enterprise, EAI solutions were developed to pump data across technology and application ownership boundaries.

Towards the late 1990's, development teams saw the interoperability of the Internet's standard protocols and began leveraging them to combat the lack of reach in the enterprise. Many projects adopted plain old XML (POX) Web services and soon SOAP Web services would gain popularity. By then, data had been strewn across the enterprise where a single logical business entity had attributes stored in multiple systems.

SOA quickly shifted from a paradigm of developing good services to an approach of abstracting the business process layer, or business models, from the complexities of the integration/persistence layer, or technology models. The reach of Web services, the abstraction of backend complexity, and service lifecycle management (SLM) adoption now define what enterprise SOA is today. However, as teams build out this abstraction layer across the enterprise, they often encounter difficult data-related challenges.

For all the time development teams save by quickly consuming Web services, they lose time determining if they are reading or writing a particular attribute from the proper system of record. Even teams that have a clear understanding of where trusted data lives often encounter challenges aggregating data stored with different keys and redundant attributes that may not be consistent. Organizations should expect the development of an SOA infrastructure to be difficult. The benefits are realized on subsequent projects leveraging that infrastructure. Yet, challenges of this difficulty are often unaccounted for and may end up jeopardizing SOA projects. Fortunately, solutions to these challenges can be found in a discipline known as Master Data Management (MDM).

MDM normalizes, rationalizes, and seemingly centralizes distributed enterprise data. The concept is not new. Since the development of the relational database management system (RDBMS), organizations have attempted to establish a central data repository. Where central data repositories relied on modifying applications and developing new applications to take advantage of the shared database, MDM grants that data silos will exist and any attempt to completely eliminate them will lead to failure due to the complexity of the effort or the potential lack of agility after the effort is complete.

A Typical Scenario

Take the case of a fictional organization that stores:

- a customer's billing information in an ERP system
- customer ordering data in a custom LOB application
- customer interaction data from a call center in a CRM system

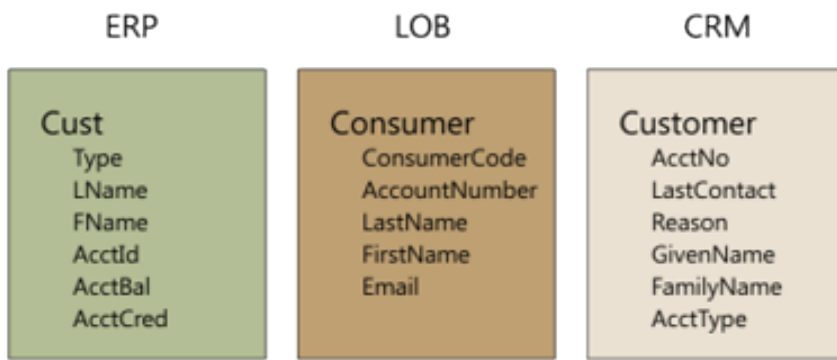


Figure 2: Three different bodies of customer-related data stored in three different environments.

Figure 2 shows how duplicated data ends up residing in each system with additional attributes supporting system specific functions. This is further complicated by the fact that the same data will tend to get referenced by a different name or by the same logical attribute containing different data. For example, the ERP Cust.Type field may contain codes like “BUS” for business customers but the CRM Customer.AcctType might contain “Business” for business customers. Identifying identical entities across the three systems is difficult and can require human intervention if the primary keys are different. In large systems with many similar records and varying degrees of data quality, this task becomes even more difficult. Let's say that you find records like these:

- John Smith
- Jon Smith
- Jonathan Smith
- Jon Smyth

Not only will you probably not be able to identify identical entities, you are more likely to falsely identify identical entities. EAI or ETL processes frequently keep other systems synchronized if one is updated. In large IT shops, this can grow to be a complex and delicate orchestration where any changes to a business entity's attributes can become a Herculean effort unto itself.

New systems will constantly struggle to access this customer information efficiently. If an interactive voice response (IVR) solutions is being developed to check order status and account balances, for example, it would need to access data persisted in the ERP and LOB applications. However, representatives in the call center would like to know if a caller had just been using the IVR and could not complete their desired activity.

Developers of this system would ask these questions:

- Which system is the system of record?
- If there isn't a clear system of record, should we update all of them at once?
- If we update all of them at once, should we do it in a distributed transaction?
- How will this affect our availability?
- How will this affect our performance?
- Where can we find customer data in the LOB system?

Abstracting this complexity for service consumers is the role of the service layer. However, for service providers the complexity of the enterprise data layer requires further attention. Not only must service providers negotiate integration hurdles, but the business must trust that the proper logic is executed and the proper data is returned.

Thus, providing trustworthy services demands adoption of new techniques that readily apply existing concepts, technologies, and processes. For example, as in the customer data scenario, it is common for systems to duplicate account balances across systems. However, in most cases the business will only trust the data in the account balance field from one system. Therefore, if the developers had ready access to another system, there is the potential that consumers of the service may display inconsistent data. When this occurs in a project, any gains in integration

development efficiency are countered by data mapping activities demanded by business owners.

As SOA continues to gain mindshare with technical decision makers of complex, heterogeneous, and distributed IT environments, the ability to effectively manage data is fundamental to the delivery of trustworthy services.

Master Data

Establishing master data is the first objective. The goal is to establish a high quality data store where enterprise information has been aggregated, mapped, normalized, and standardized. Usually master data is set up in a new database because of the state of existing systems where naming conventions are not ideal and data quality may not be high. In some instances, a system may be of sufficient quality to take on the role of master data and attributes only found in other systems may be added.

Data Aggregation

Enterprise information organically spreads throughout the organization. While core attributes are common across disparate systems, unique attributes are usually found to support system-specific functionality. To get a truly enterprise-wide view of a core business entity, like a customer, a distinct list of attributes must be identified and stored, as shown in Figure 3.

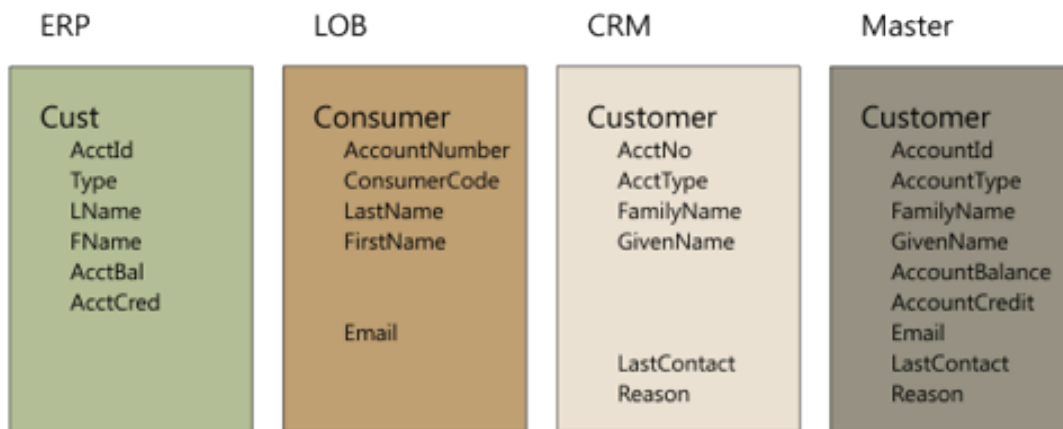


Figure 3: An expanded list of customer attributes.

In this scenario, customer master data would include the common attributes found in the ERP, LOB, and CRM systems plus the attributes unique to each system, like account balance, account credit limit, email, date of last contact, and the contact reason.

Data Mapping

When data has not been actively managed, one frequently finds that it has evolved independently, leading to differing table and column names, and foreign naming conventions. The data mapping task is one of identifying which columns represent the same entity attribute across systems. The prototypical example is that surname, last name, and family name all refer to same information.

Data Normalization

Different systems inevitably mean varying data quality and consistency. Master data should represent the single point of truth and must be trustworthy. Data normalization leads to consistently formatted phone numbers, social security numbers, and postal codes. Addresses may also be validated and formatted to postal standards.

Data Standardization

Data elements may be properly mapped and normalized but still remain inconsistent in the context of global systems and the way status codes are managed differently in systems. For example, transaction dates could be mapped

across systems and formatted consistently but represent different time zones. (Dates and times should be standardized according to GMT and status codes should also be standardized.)

Master data should provide an aggregate view of a business entity. It is important to note that some attributes will be specific to a given system and will not warrant being included in master data, and thus not require inclusion in the data cleansing and moving process.

Not only does this result in an aggregate view of customer data, it also increases data quality by formatting phone numbers, validating addresses, standardizing times on GMT, etc. Now that master data has been established, defining the topology and merge strategy are the next objectives.

Determine a Topology

Availability and performance characteristics of master data and the constituent systems will largely drive the topology. It is important to note that participating systems may use the master data tables directly or the master data tables may simply exist as a data hub for the enterprise and would therefore be transparent to existing systems. A centralized topology model is the simplest because it requires a single copy of master data.

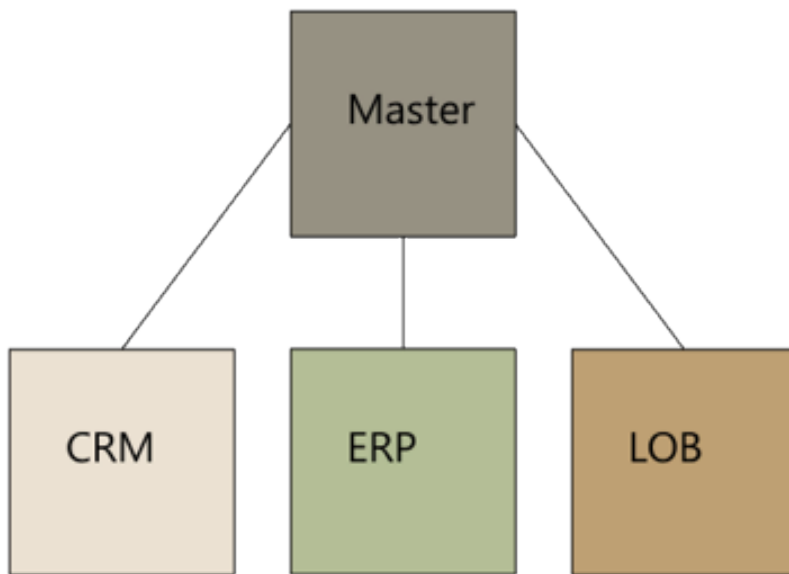


Figure 4: A centralized topology model.

A distributed topology model may deploy a local copy of master data to each constituent system. This approach distributes the load placed on master data and for systems modified to access master data directly is potentially increased due to the local data store.

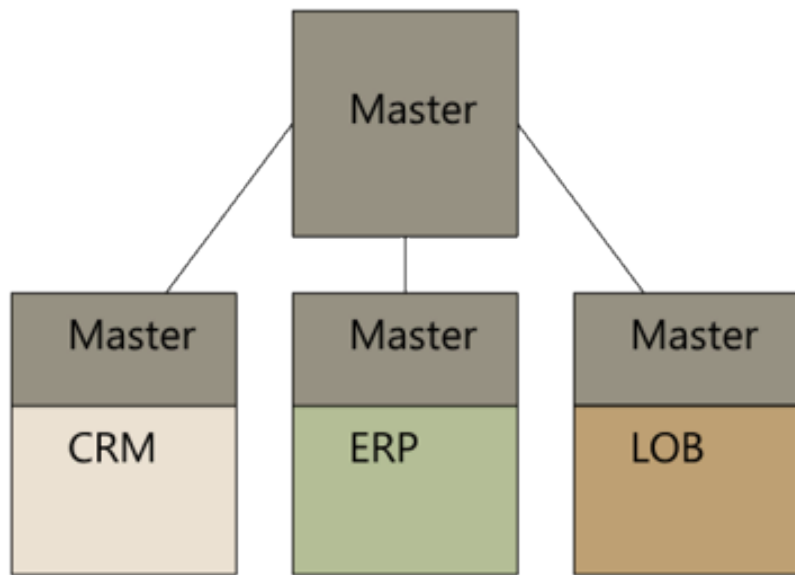


Figure 5: A distributed (decentralized) topology model.

A common concern with the decentralized model is the amount of storage required for multiple copies of master data. To counter this issue, each system can be designed to only subscribe to a subset of master data that only pertains to the application in question.

Deciding on a Merge Model

In the implementation of a master data management solution, nothing will affect the long term success and maintenance efforts more than an effective (or ineffective) merge model.

Two common mistakes when deciding on a merge model are:

1. Assuming that the chosen topology should drive the merge model.
2. Assuming that writes should always be performed against the central master data store.

For example, deploying distributed master data does not necessarily imply that each system would be able to update their copy of local data. However, certain scenarios could dictate that local updates to master data be made and propagated to subscribing systems.

It all need to begin with the identification of updatable nodes. The number of updatable nodes could be anywhere from one to all of the participating nodes. If there is only one updatable node, it could be any participating node.

Let's take a look at some options:

Single Updateable Node – Central Master Data

Whenever possible, services should read and write directly against master data but that may not be feasible in every case. The resulting context is one where participating systems may come and go with little or no disruption. Service development may also gain efficiency from the well structured data.

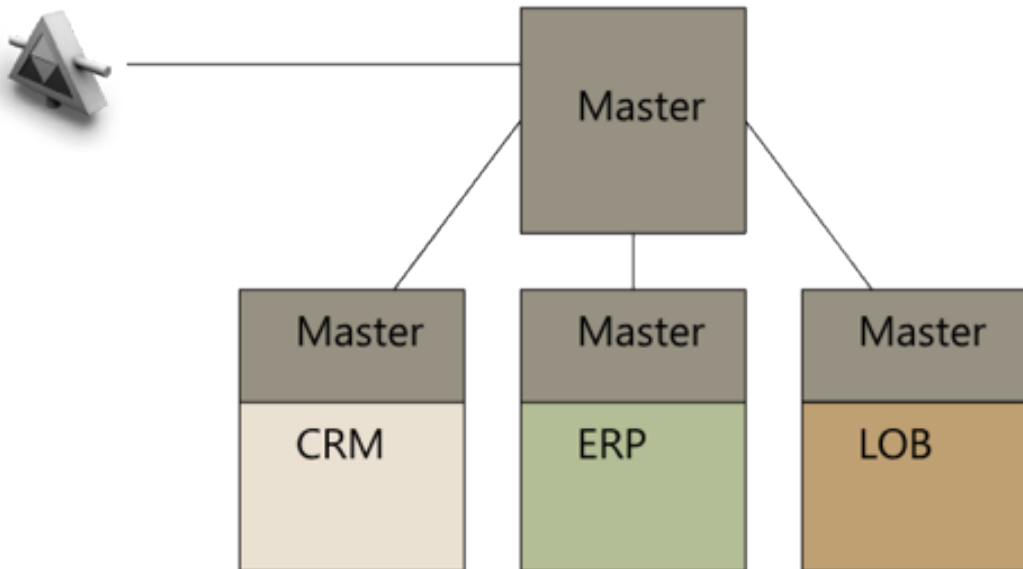


Figure 6: The single updateable node/central master data model.

Single Updateable Node – Local Master Data

This strategy is common for critical applications that have been remediated to use local master data. A typical scenario is a billing system used by call center agents for conducting adjustments. Local master data is leveraged for performance reasons and the business may only trust the business logic in the application to make the update.

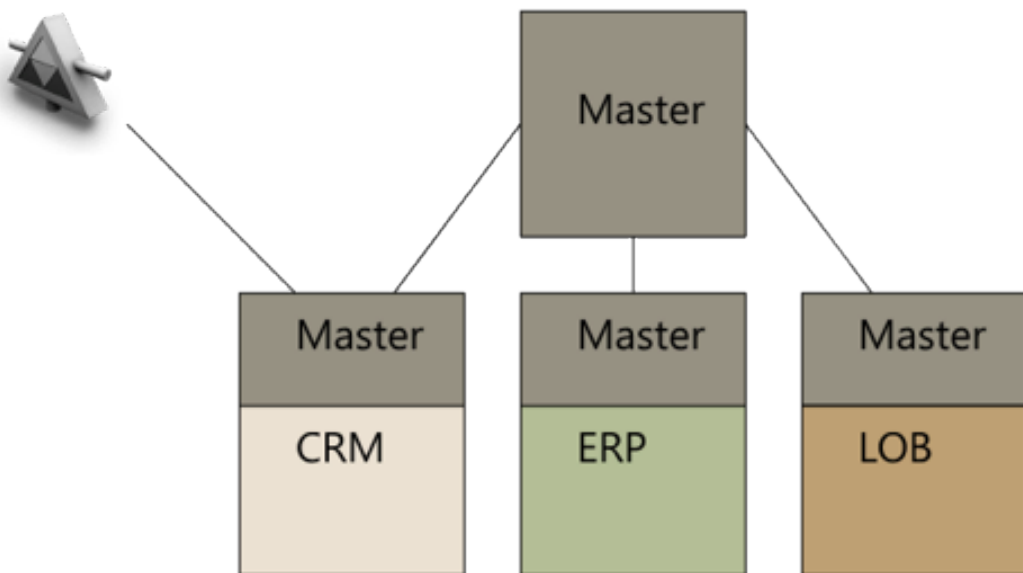
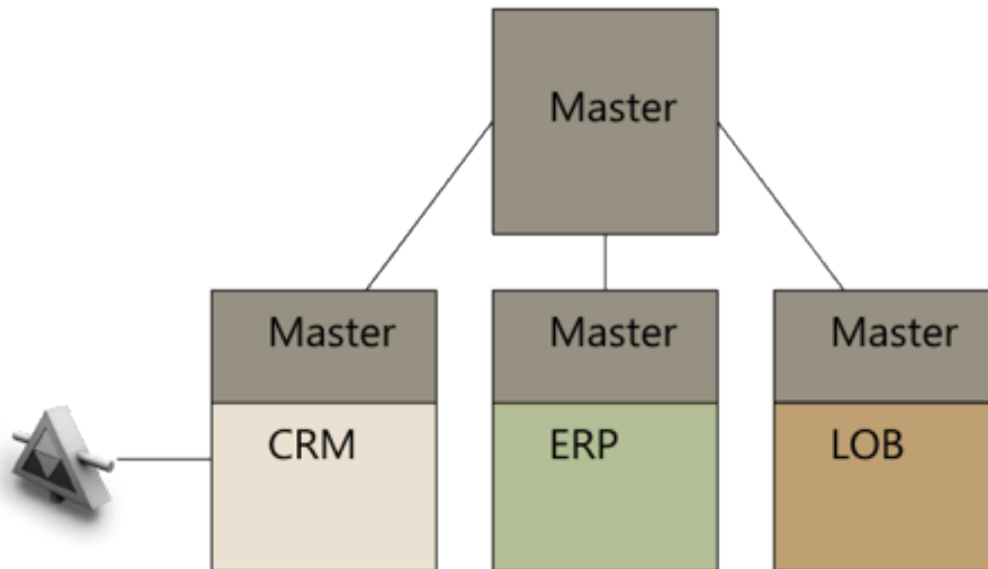


Figure 7: The single updateable node/local master data model.

Single Updateable Node – Local Data

This strategy is common for applications that have not been remediated and participate with master data transparently through an ETL process. The scenario is usually similar to Single Updateable Nodes that use Local Master Data.

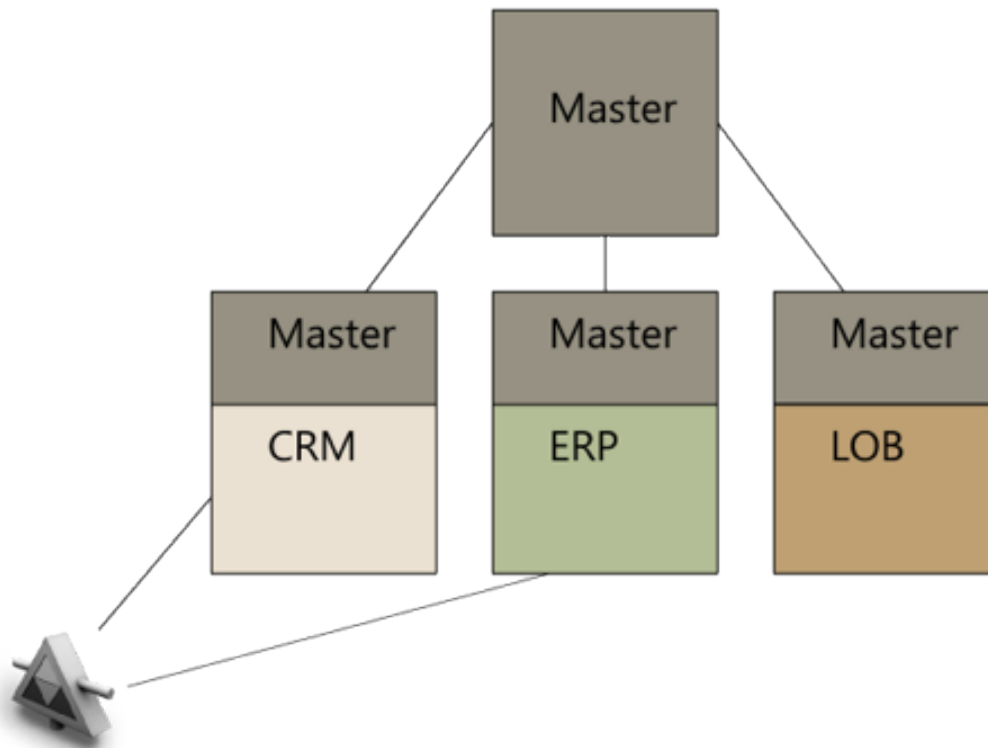
Figure 8: The single updateable node/local data model.



Multiple Updateable Nodes

This strategy is accompanied by potential merge conflicts that must be manually resolved and accounted for in the merge process. Although merge conflicts may occur, this method requires the least amount of effort on the part of participating systems.

Figure 9: The multiple updateable nodes model.



In any scenario with local master data (or not with single or multiple update nodes), master data serves as a hub for keeping participating systems in sync in addition to the role of storing the aggregated, normalized, and standardized data. These systems may move data using the same technologies, but it is unnecessary. The CRM system may leverage techniques from business intelligence (BI) and use ETL tools. The ERP system may leverage the tried and true method of transferring FTP files. The newer LOB application may receive updates as they happen from an EAI tool like BizTalk. Like most architecture decisions, the approach will depend on the unique applications, tools available, and the skill of the team.

EDW Considerations

Strong master data will also simplify enterprise data warehouse (EDW) efforts, because an EDW that loads data from master data has the data aggregation, normalization, and standardization performed on its behalf. While EDW and MDM solutions provide an enterprise view of business entities and leverage many of the same concepts and technologies, the relational model of an MDM solution is transactional and not dimensional.

Conclusion

Service-orientation has quickly been adopted for the purpose of abstracting backend complexity from actionable interfaces, LOB applications, and business partners. Once the Service Façade pattern has been applied, backend optimization for performance and maintenance becomes possible. MDM is one technique to assist in that optimization. Existing services are easier to maintain and will perform better. New services will be easier to develop and will become trustworthy to the business without the need for time-consuming data mapping activities.

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Glossary](#) [SOA School](#) [SOA Books](#) [About](#) [Legal](#)

Copyright © 2006-2008
SOA Systems Inc.