PART

CHAPTER 4

101

07

MDM Architecture Classification, Concepts, Principles, and Components

CHAPTER 5

Data Management Concerns of MDM Architecture: Entities, Hierarchies and Metadata

CHAPTER 6

MDM Services for Entity and Relationships Resolution and Hierarchy Management

CHAPTER 7 Master Data Modeling



In the introductory part of this book, we offered a broad-brush description of the purpose, drivers, and key benefits of Master Data Management and used some specific examples of its customer-focused variant, Customer Data Integration. This part of the book discusses the issues of MDM architecture as a key logical step to building enterprise-wide solutions.

An architecture discussion is important for several reasons:

- A comprehensive end-to-end MDM solution is much more than just a database of customer or product information organized by some kind of a unique key. Some MDM capabilities and components are "traditional" and are a part of a common best-practice design for integrated data solutions, whereas other, new features came to light primarily in the context of MDM problem domains. An architectural vision can help organize the "old" and the "new" features into an integrated, scalable, and manageable solution.
- MDM is not just a technology problem—a comprehensive MDM solution consists
 of technology components and services as well as new business processes and even
 organizational structures and dynamics. There are many architecture viewpoints,
 significant complexity, and a large number of interdependencies to warrant a
 framework-based approach to the architecture. This multifaceted, multidimensional
 architecture framework looks at the overall problem domain from different but
 complementary angles.
- Any solution intended to create an authoritative, accurate, and timely system of
 record that should eventually replace existing legacy sources of the information
 must be integrated with the overall enterprise architecture and infrastructure. Given
 the heterogeneity and the "age" of legacy systems, this requirement is often difficult
 to satisfy without a comprehensive architecture blueprint.

Thus, we organized this part of the book in the following fashion: First, we discuss the architectural genesis of MDM. Then, we take a closer look at the enterprise architecture framework and explain how this framework helps us to see different aspects of the solution as interconnected and interdependent views. This discussion is followed by an overview of traditional data management and emerging concerns of MDM architecture, MDM data modeling, data management architecture, and the newer concept of MDM services.

CHAPTER

MDM Architecture Classifications, Concepts, Principles, and Components

In order to understand "how" to build a comprehensive Master Data Management solution, we need to define the "what" of Master Data Management. We have already offered high-level definitions of MDM and its customer-focused variant, CDI, in Part I of this book. We also stated that CDI and other MDM variants share many architecture principles and approaches; therefore, in this part of the book we concentrate on common architecture aspects of Master Data Management. Where appropriate, we'll mention specific architecture features of key MDM variants—in particular, Customer Data Integration and Product Information Master.

Architectural Definition of Master Data Management

As shown in previous chapters, the scope of Master Data Management by its very nature is extremely broad and applies equally well to customer-centric, product-centric, and reference data-centric business problems, to name just a few. A common thread among the solutions to these problems is the ability to create and maintain an accurate, timely, and authoritative "system of record" for a given subject domain. Clearly, such a definition can be refined further for each situation and problem domain addressed by Master Data Management.

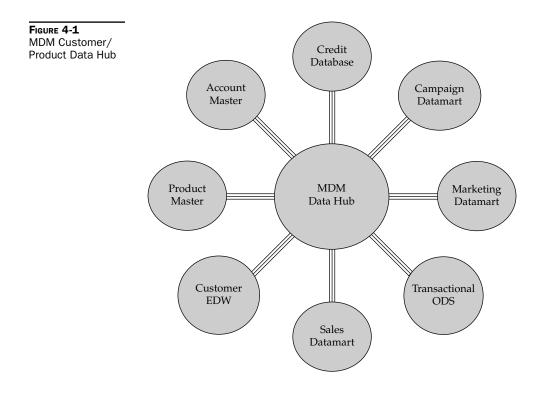
Let's start with a fresh look at the definitions of master data and Master Data Management offered in Chapter 1:

- Master data is composed of those entities, relationships, and attributes that are critical for an enterprise and foundational to key business processes and application systems.
- *Master Data Management (MDM)* is the framework of processes and technologies aimed at creating and maintaining an authoritative, reliable, sustainable, accurate, and secure data environment that represents a "single and holistic version of the truth," for master data and its relationships, as well as an accepted benchmark used

within an enterprise as well as across enterprises and spanning a diverse set of application systems, lines of business, channels, and user communities. To state it slightly differently, an MDM solution takes the master data of a given domain from a variety of data sources' discards redundant data; and then cleanses, rationalizes, enriches, and aggregates it to the extent possible. We can illustrate such an MDM environment as a "hub and spokes," where the spokes are information sources connected to the central hub as a new "home" for the accurate, aggregated, and timely master data (see Figure 4-1). This description helps explain why we often use the term "Data Hub" when discussing an MDM solution space.

Interestingly, using this definition of "what" MDM is does not make our goal of creating architecture much easier to achieve. Indeed, this definition points to the fact that, for example, a CDI solution is much more than just a database of customer information, a solution known by many as a Customer Information File (CIF), a data warehouse of customer information, or an operational data store (ODS). In fact, this definition describes an enterprise-scale system that consists of software components, services, processes, data models and data stores, metadata repositories, applications, networks, and other infrastructure components.

Thus, in order to develop a clear understanding of the "how" of the MDM solution, we will review the historical roots of Master Data Management and its evolution from early attempts to deliver on the MDM promise to what it has become today.



Evolution of Master Data Management Architecture

As we discussed in Chapter 1, the need to create and maintain an accurate and timely "information system of record" is not new, and it applies equally well to businesses and government entities. Lately, a number of regulatory requirements, including the Sarbanes-Oxley Act, the Basel II Capital Accord, and the emerging Basel III Accord (see the discussion on these regulations in Part III of the book), have emphasized this need even further.

In the case of Customer Data Integration, organizations have been engaged in creating customer-centric business models and applications and enabling infrastructure for a long time. However, as the business complexity, number and type of customers (retail customers, individuals, institutional customers, and so on), number of lines of business, and number of sales and service channels continued to grow, this growth often proceeded in a tactical, nonintegrated fashion. As result, many organizations ended up with a wide variety of customer information stores and applications that manage customer data. As an example, one medium-sized service/distribution company maintained no less than eight customer databases that had to be rationalized and cleansed in order to achieve targeted goals for efficiency and quality of the customer service.

The customer data in that "legacy" environment was often incomplete and inconsistent across various data stores, applications, and lines of business. In many other cases, individual applications and lines of business were reasonably satisfied with the quality and scope of customer data they managed. However, the lack of completeness and accuracy and the lack of consistency across lines of business continued to prevent organizations from creating a complete and accurate view of customers and their relationships with the servicing organization and its partners.

Similarly, product information is often scattered across multiple systems. Products and services are modeled in product design and analysis systems where product functionality, bills of materials, packaging, pricing, and other characteristics are developed. Once the product modeling is complete, product information along with product-specific characteristics are released for cross-functional enterprise use.

NOTE In the scope of MDM for customer domain, we often discuss business transformation to achieve customer centricity as a major goal and benefit of MDM. However, given the domain-agnostic nature of MDM, it is more accurate to talk about transforming the enterprise from an account-centric to an entity-centric model, and, where possible, we'll be using the term "entity centricity" when discussing this transformational feature of MDM.

Recognizing the entity-centricity (e.g., customer, product) challenge and the resulting inability to transform the business from an account-centric to an entity-centric model, organizations first developed a variety of solutions that attempted to help move the organizations into the new entity-centric world. Although in general these solutions added some incremental value, many of them were deployed in the constraints of the existing lines of business, and very few were built with a true enterprise-wide focus in mind. Nevertheless, these solutions and attempts to achieve entity centricity have helped define MDM in general and CDI and PIM in particular to become a real enabler of such business model transformations. Therefore, we need to understand what has been done prior to the emergence of MDM, and what, if any, portions of the existing solutions can

and should be leveraged in implementing MDM. The good news is that many of these solutions are not data-domain specific and can be viewed as foundational technologies for MDM in general.

These solutions include but are not limited to Customer Information File (CIF); Extract, Transform, and Load technologies (ETL); Enterprise Data Warehouse (EDW); an operational data store (ODS); data quality (DQ) technologies; Enterprise Information Integration (EII); Customer Relationship Management (CRM) systems; and Product Master environments, to name just a few. Although some of these solutions and technologies were discussed briefly in Chapter 1, we want to offer a slightly deeper and more architecturefocused review of them, with a view toward their suitability to act as components of a Master Data Management platform.

• **Customer Information File (CIF)** Many companies have established LOB-specific or company-wide customer information file environments. Historically, CIF solutions used older file management or database management systems (DBMS) technology and represented some very basic point-in-time (static) information about the customers. In other words, CIFs offer limited flexibility and extensibility and are not well suited to capturing and maintaining real-time customer data, customer privacy preferences, customer behavior traits, and customer relationships. Moreover, traditional CIF does not support new complex business processes, event management, and data element-level security constraints known as "data visibility" (see Part III for a detailed discussion on this topic). Shortcomings like these prevent traditional CIF environments from becoming a cross-LOB integration vehicle of customer data.

Although CIF systems do not deliver a "single version of the truth" about the customer, in most cases existing CIF systems are used to feed the company's Customer Relationship Management systems. Moving forward, a CIF can and should be treated as a key source data file that feeds a new Master Data Management Customer Data Hub system.

• Extract, Transform, and Load (ETL) These tools are typically classified as dataintegration tools and are used to extract data from multiple data sources, transform the data to a required target structure, and load the data into the target data store. A key functionality required from the ETL tool is its ability to perform complex transformations from source formats to the target; these transformations may include Boolean expressions, calculations, substitutions, reference table lookup, support for business rules for aggregation and consolidation, and many other features. Contemporary ETL tools include components that perform data consistency and data quality analysis as well as the ability to generate and use metadata definitions for data attributes and entities. Many tools can create output data in XML format according to the predefined schema. Finally, the enterprise-class ETL tools are designed for high scalability and performance and can parallelize most of their operations to achieve acceptable throughput and processing times when dealing with very large data sets or complex transformations.

Although many ETL processes run in batch mode, best-in-class ETL tools can support near-real-time transformations and load functionality. Given that description, it is quite clear that an ETL component can and should be used to transform and load

data into an MDM platform—Data Hub—both for the initial load and possibly for the incremental data updates that keep the Data Hub in sync with existing sources. We discuss MDM data synchronization approaches using ETL in Chapter 16 of the book.

• Enterprise Data Warehouse (EDW) Strictly speaking, a data warehouse is an information system that provides its users with current and historical decision-support information that is hard to access or present using traditional operational data stores. An enterprise-wide data warehouse of customer information can become an integration vehicle where most of the customer data can be stored. Likewise, an enterprise data warehouse of product information can act as an integration point for many product-related transactions. Typically, EDW solutions support business intelligence (BI) applications and, in the case of customer domain, Customer Relationship Management (CRM) systems. EDW's design, technology platform, and data schema are optimized to support the efficient storage of large amounts of data and the processing of complex queries against a large number of interconnected data tables that include current and historical information. Traditionally, companies use EDW systems as informational environments rather than operational systems that process real-time, transactional data.

Because EDW cleanses and rationalizes the data it manages in order to satisfy the needs of the consuming BI and CRM systems, an EDW becomes a good platform from which data should be loaded into the Data Hub.

• **Operational data store (ODS)** This technology allows transaction-level detail data records to be stored in a nonsummarized, query accessible, and long-lasting form. An ODS supports transaction-level analysis and other applications that deal with the low level of details. An ODS differs from a data warehouse in that it does not maintain summarized data, nor does it manage historical information. An ODS allows users to aggregate transaction-level data into higher-level attributes but does not support a drill-down into the underlying detail records. An ODS is frequently used in conjunction with the Enterprise Data Warehouse to provide the company with both historical and transactional real-time data.

Similar to the EDW, an ODS that contains customer or product data can and should be considered a valuable source of information for constructing an MDM solution.

• Data quality (DQ) technologies From the point of view of a business value proposition, the focus of data quality technologies and tools is to help all applications to produce meaningful and reliable results. These tools are especially important for delivering accurate business intelligence and decision support as well as improving customer retention, sales and customer service, customer experience, risk management, compliance, and fraud detection. Companies use data quality technologies to profile data, to report anomalies, and to standardize and "fix" data in order to correct data inconsistencies and known data quality issues, such as missing or invalid data.

Although data quality tools are especially effective when dealing with the name and address attributes of customer data records, they are also very useful for managing data quality in other data domains. Thus, data quality tools and technologies are key components of most Master Data Management solutions.

• Enterprise Information Integration (EII) Enterprise Information Integration tools are frequently used to aggregate subsets of distributed data in memory or nonpersistent storage, usually in real time. Companies use EII solutions to perform search queries across distributed databases and aggregate the results of the queries at the application or presentation layer. Contrast that with the data-integration solutions that aggregate and persist the information at the back end (that is, in a data warehouse or an MDM Data Hub). An EII engine queries a distributed database environment and delivers a virtualized aggregated data view that appears as if it came from a single source. EII engines are also used often in a service-oriented architecture (SOA) implementation as the data access and abstraction components (we discuss SOA later in this chapter).

Some MDM implementations use EII technologies to provide users with a virtualized total view of a master data without creating a persistent physical image of the aggregation, thus providing additional data model flexibility for the target Data Hub.

- **Customer Relationship Management (CRM)** Customer Relationship Management uses a set of technologies and business processes designed to help the company understand the customer, improve customer experience, and optimize customer-facing business processes across marketing, sales, and servicing channels. From the architecture perspective, CRM systems often act as consumers of customer data and are some of the primary beneficiaries of the MDM Data Hubs.
- **Product Master** Manufacturing companies manage a variety of complex products and product hierarchies. Complex products consist of multiple parts, and those parts contain lower-level components, materials, or parts. This hierarchy represents what is often called a "Bill of Materials" (BOM). BOM management software helps centralize and control complex BOM processes, reduce error rates, and improve control over operational processes and costs.

An MDM system that is integrated with BOM management software can significantly enhance an integrated multidomain view of the master data. For example, a product characterized by BOM components can be integrated with suppliers' component data.

MDM Architectural Philosophy and Key Architecture Principles

MDM has evolved from and is a direct beneficiary of the variety of solutions and approaches described in the previous section. In this context, MDM enables an evolutionary approach to constructing a comprehensive architectural vision that allows us to define many different viewpoints, each of which represents a particular architecture type.

Moreover, we can create an MDM architecture view that addresses a variety of architectural and management concerns. Specifically, we can develop an architectural view that defines components responsible for the following functional capabilities:

- Creation and management of the core data stores
- Management of processes that implement data governance and data quality
- Metadata management

- Extraction, transformation, and loading of data from sources to target
- Backup and recovery
- Customer analytics
- Security and visibility
- Synchronization and persistence of data changes
- Transaction management
- Entity matching and generation of unique identifiers
- Resolution of entities and relationships

The complexity of the MDM architecture and the multitude of architectural components represent an interesting problem that is often difficult to solve: how to address such a wide variety of architectural and design concerns in a holistic, integrated fashion. One approach to solving this type of challenge is to use the classical notion of a top-down, abstracted representation of the MDM functionality as a stack of interdependent architecture layers, where a given layer of functionality uses services provided by the layers below and in turn provides services to the layers above.

Defining Service-Oriented Architecture

Service-oriented architecture (SOA) is the architecture in which software components can be exposed as loosely-coupled, fine-grained, or coarse-grained reusable services that can be integrated with each other and invoked by different applications for different purposes through a variety of platform-independent service interfaces available via standard network protocols.

We can further enhance the notion of the layered architecture by expressing the functional capabilities of each of the architecture layers in the stack as a set of abstracted services, with a degree of abstraction that varies from high (at the upper layers of the stack) to low (for the bottom layers of the stack). The notion of abstracted services is very powerful and provides architects, designers, and implementers with a number of tangible benefits. We discuss these benefits and the principles of service-oriented architecture (SOA) later in this chapter.

Applying the notion of service-level abstraction to the MDM architecture, we now define its key architecture principles as follows:

- An effective MDM solution should be architected as a metadata-driven SOA platform that provides and consumes services that allow the enterprise to resolve master entities and relationships and move from traditional account-centric legacy systems to a new entity-centric model rapidly and incrementally.
- We can define several key tenets of the information management aspects of the MDM architecture that have a profound impact on the design, implementation, and use of the MDM platform:
 - Decouple information from applications and processes to enable its treatment as a strategic asset.
 - Support the notion that the information content (master data) shall be captured once and validated at the source to the extent permissible by the context.

PART II

- Support propagation and synchronization of changes made by MDM system to key master attributes so the changes are available to the consuming downstream systems.
- Enable measurement, assessment, and management of data quality in accordance with information quality standards established by the organization and articulated as part of business needs and data governance.
- Ensure data security, integrity, and appropriate enterprise access.
- Support the retention of data at the appropriate level of granularity.
- Provide an effective vehicle for standardizing content and formats of sources, definitions, structures, and usage patterns.
- Enable consistent, metadata-driven definitions for all data under management.
- Preserve data ownership and support well-defined data governance rules and policies administered and enforced by an enterprise data governance group.

Although the notion of supporting these key information management tenets and using service-level abstraction is fundamental and even necessary in architecting enterprise-scale MDM solutions, it is not totally sufficient. Other aspects of the MDM architecture are better described using alternative architecture representations, or architecture viewpoints, that differ in the content, context, and levels of abstraction. In order to formalize the process of defining and using various architecture viewpoints, we need to introduce the notion of a multidimensional enterprise architecture framework. Readers already familiar with the principles and concepts of the architecture framework can skip the following section.

Enterprise Architecture Framework: A Brief Introduction

As stated earlier in this chapter, the complex multifaceted nature of an MDM solution cannot be described using a single architecture view, but instead requires a number of architectural perspectives organized in a multidimensional architecture framework. Let's illustrate this framework notion using an analogy of building a new community within existing city boundaries. In this case, the city planners and the architects need to create a scaled-down model of the new area, including buildings, streets, parks, and so on. Once this level of architecture is completed and approved, the building architects would start developing building blueprints. Similarly, the road engineers would start designing the streets and intersections. Utilities engineers would start planning for underground cabling, water, and sewerage. City planners would start estimating the number and types of schools and other public facilities required to support the new community. And this list goes on.

Clearly, before the work can get started, the city planners will have to create a number of architecture views, all of which are connected together to enable a cohesive and complete picture of *what*, *when*, *where*, and *how* the individual parts of the new city area will be built.

To state it differently, any complex system can be viewed from multiple angles, each of which can be represented by a different architecture perspective. To organize these various architecture perspectives into a holistic and connected picture, we will use the enterprise architecture framework first pioneered by John Zachman. This framework helps architects, designers, and engineers to develop a complex solution in a connected, cohesive, and comprehensive fashion.

Zachman's principal insight is the way to solve the complexity of the enterprise architecture by decomposing the problem into two main dimensions, each of which consists of multiple subcategories. The first dimension defines the various levels of abstraction that represent business scope, conceptual level (business model), logical level (system model), and physical level (technology model). The second dimension consists of key decision-driving questions—what, how, where, who, when, and why. In the context of the enterprise architecture, these questions are considered at the different levels of the first dimension as follows:

- "What" answers the question about what data flows throughout the enterprise.
- "How" describes the functions and business processes performed by the different parts of the enterprise.
- "Where" defines the network that provides interprocess and intercomponent connectivity and information delivery.
- "Who" defines the people and organizational structures affected by the target architecture.
- "Why" represents business drivers for this architecture-based initiative.
- "When" defines the timing constraints and processing requirements.

Each question of the second dimension at every level of the first dimension represents a particular architecture viewpoint—for example, a logical data model view or a physical network architecture view. All these 30 viewpoints are organized together in the framework to comprise a complete enterprise architecture. Figure 4-2 shows a graphical representation of Zachman's framework.

The representation in Figure 4-2 is based on the work published by Zachman's Institute for Framework Advancement (ZIFA).¹

The value of such an architecture framework is its ability to act as a guide for organizing various design concerns into a set of separate but connected models. The framework benefits become apparent as the complexity and the heterogeneity of the system that is being designed increase. In the case of Master Data Management, this framework approach helps address the complexity of the individual functions and components; the integration of the new MDM environment with the legacy systems; and the need to implement an effective, efficient, secure, and manageable solution in a stepwise, controlled fashion.

Architecture Patterns

The other approach to solving complex system design and architecture challenges is the notion of architecture and design patterns. A pattern is a proven, successful, and reusable approach to solving a well-defined problem. Here are some specifics:

- A pattern is an approach to the solution that has been implemented successfully a number of times in the real world to solve a specific problem space.
- Typically, patterns are observed and documented in the course of successful real-life implementations.

Why	Motivation	Lists of business goals' strategies Ends' means = Major business goals' strategy	Eg. Business plan	Eg. Business role model Eg. Business role model End = Structural assertion Means = Action assertion	Eg. Role design	Eg. Role specification Eg. and Sub-condition End = Sub-condition Means = Step	Strategy
When	Time	List of events' cycles significant to the business Time = Major business event/cycle	Eg. Master schedule	Eg. Processing structure	Eg. Control structure	Eg. Timing detention	Schedule
Who	People	List of organizations important to the busines	Eg. Work flow model	Eg. Human interface architecture People = Role Work = Deliverable	Eg Presentation architecture People = User Work = Screen formats	Eg. Security architecture	Organization
Where	Network	List of locations in which the business operates	Eg. Business logistics system System Node = Businesses location Link = Businesses location	Eg. Distributed system architecture Node = 1/5 furction (procesory storage etc.) Link = Line characteristics	Eg. Technology architecture architecture architecture architecture Node = Hdw/system Node = Hdw/system Link = Line specifications	Eg. Network architecture	Network
How	Function	List of process the business performs	Eg. Business piocess model Process = Business process (O = Business reportces	Eg. Application architecture Process = Application Interior IO = User views	Eg. System design Process = Computer function Interior	Eg. Program Eg. Program Process = Language statement O = Control block	Function
What	Data	List of things important to the business Entity = Class of business	Eg. Semantic model	Eg. Logical data modet Entity = Data entity Relationship = Data relationship	Eg. Physical data model Entity = Segment/table/etc. Relationship = Pointer/ key/etc.	Eg. Data definition Eg. Data definition Emity = Field Relationship = Address	Data
		Scope (contextual) Planner	Business Model (conceptual) Owner	System Model (logical) Designer	Technology Model (physical) Builder	Detailed Representations (out-of-context) Subcontractor	Functioning Enterprise

r



٦

- Patterns don't solve every single aspect of every problem, and typically are focused on core, main aspects of the problem (following the law of the "trivial many and the critical few," better known as Pareto's Law, or the 80-20 Rule).²
- When defined correctly, patterns are easy to apply to service-oriented architectures because they leverage object-oriented design principles, especially the notion of inheritance, by often inheriting components (objects) from already defined patterns.

When we discuss architecture patterns, their primary benefit is in helping architects of complex systems such as MDM to identify various design options and understanding which options are most appropriate for a given problem domain. More often than not, individual patterns have to be combined with other patterns to define a solution to a particular business problem.

Patterns are different from architecture viewpoints. They tend to solve a well-defined, discrete problem space by providing proven, reusable choices. Architecture viewpoints, on the other hand, offer an opportunity to the architects and designers to view the problem from different angles to understand the interrelationships among the components, services, and other system objects and to formulate an approach to solve problems when the directional decisions have been made. In other words, a typical architecture viewpoint represents an aspect of the problem space that needs to be broken into a set of patterns that can be implemented with high confidence.

In MDM space, we use both architecture viewpoints and patterns, although patterns, by representing a more constraint problem domain, tend to provide a lower level of technical design.³

MDM Architecture Viewpoints

Because of its broad coverage of the business-to-technology dimensions, an architecture framework can help organize and promote different points of view for an enterprise. Different groups within the organization may express these points of view based on their organizational affiliation, skill sets, and even the political landscape of the workplace. Because a full-function MDM solution tends to be truly an enterprise-scale initiative that spans organizational and lines-of-business boundaries, one benefit of using the framework approach is to help gain organizational buy-in and support for expensive and lengthy MDM projects.

Of course, we do not want to create an impression that any MDM solution has to be architected using Zachman's framework. In fact, very few enterprise-wide initiatives use this framework in its entirety with all its 30 viewpoints. Many architecture-savvy organizations use a subset of the complete enterprise architecture framework or different architecture viewpoints. The goal of the preceding discussion was simply to illustrate the principles and benefits of the enterprise architecture framework and patterns approach as a way to solve the design and implementation challenges of any large and complex software system.

We would like to use the principles of the architecture framework to define the most relevant architecture viewpoints for a successful design and implementation of an MDM solution, with a specific emphasis on the MDM Data Hub implementations. In this context, we will focus the framework viewpoints discussion on the conceptual and logical levels of the architecture, and shall consider the following set of architecture viewpoints:

- Architecture viewpoints for various classification dimensions, in particular the consumption and reconciliation dimension and the use pattern dimension
- Conceptual architecture
- High-level reference architecture
- Services architecture
- Data architecture

From the framework perspective, we recognize many different but equally important architecture viewpoints. However, because describing a complete framework set is beyond the scope of this book, we'll focus the follow-on discussion in this chapter on three viewpoints: the services view, architecture views of MDM classification dimensions (we introduced this topic in Chapter 1), and the reference architecture view. We discuss additional architecture details and specific data architecture views in Chapters 5, 6, and 7, whereas data security and visibility architecture views are discussed in Chapter 11.

Services Architecture View

A services architecture viewpoint is probably one of the most relevant to the architecture discussion of the MDM system. Indeed, we have stated repeatedly that an MDM system should be an instance of the service-oriented architecture (SOA). Using this viewpoint has an additional benefit in that it helps us illustrate how we can extend the very approach of the enterprise architecture framework to describe complex systems such as MDM systems. Indeed, even though Zachman's framework does not explicitly show a services architecture viewpoint, we will define such a viewpoint for a Data Hub system and show how this viewpoint can be mapped to Zachman's framework.

Introduction to Service-Oriented Architecture

We define *service-oriented architecture* (*SOA*) as an architecture in which software components can be exposed as loosely-coupled, coarse-grained, reusable services that can be integrated with each other and invoked by different applications for different purposes through a variety of platform-independent service interfaces available via standard network protocols.

This is a practical definition but not the only valid definition of SOA. There are a number of alternative definitions of SOA,⁴ and it's beyond the scope of this book to described them all or offer arguments about the merits of individual definitions. Therefore, we should consider a standard bearer in the SOA space. The World Wide Web Consortium (W3C) has developed a comprehensive definition of the service-oriented architecture in its February 2004 Working Group publication.

W3C Definition of Service-Oriented Architecture

A service-oriented architecture (SOA)⁵ is a form of distributed systems architecture that is typically characterized by the following properties:

- **Logical view** The service is an abstracted, logical view of actual programs, databases, business processes, and so on, defined in terms of *what* it does, typically carrying out a business-level operation.
- **Message orientation** The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves.
- **Description orientation** A service is described by machine-processable metadata.
- **Granularity** Services tend to use a small number of operations with relatively large and complex messages.
- **Network orientation** Services tend to be oriented toward use over a network, although this is not an absolute requirement.
- **Platform-neutral** Messages are sent in a platform-neutral, standardized format delivered through the interfaces.

Similar to the architecture framework discussion, we can define SOA in a way that recognizes multiple views of service orientation and clearly relies on the messaging paradigm implemented over a network. Moreover, because services are composed from service components, and can be organized to work together to perform a given task, we need to introduce two additional concepts: service orchestration and service choreography. These concepts are key for the notion of service management. There are numerous, often conflicting definitions of these terms. We offer here one definition set as a reference. Readers interested in this subject can review other definitions available on the Web.⁶

SOA and Service Management: Orchestration and Choreography

Orchestration refers to the automated execution of a workflow. An orchestrated workflow is typically exposed as a set of services that can be invoked through an API. It does not describe a coordinated set of interactions between two or more parties. *Choreography* refers to a description of coordinated interactions between two or more parties.

The definition of SOA and its key concepts help define a services view of the MDM system in a way that makes it clear which services, functions, and components need to be considered and included for a full-function MDM SOA implementation. We discuss this point in more detail later in this chapter.

In addition to the regular SOA viewpoint, we can also show that the service-oriented architecture can be mapped to the viewpoints of an enterprise architecture framework. Specifically, consider that SOA is not a specific technology or product. Rather, it can be described as a design philosophy for the application architecture portion of the framework. If we use the SOA definition to represent information technology assets as services, then SOA can be mapped to the framework at the Logical level within the Function domain.

We can logically extend this approach to show that the set of functional services represents business processes, and because SOA is based on the network-aware messaging paradigm, the notion of the service orientation can be realized in several architecture framework viewpoints that connect process models and network-based messaging.

We offer these considerations simply to demonstrate that the framework approach and service-oriented architecture are closely connected and continuously evolving concepts that together can be used to help describe and plan the design and implementation of complex systems such as Master Data Management.

SOA Benefits

Additional insights into the SOA include the following key principal benefits:

- SOA offers access mechanisms to the application logic as a service to users and other applications where
 - Service interfaces are independent of user interfaces.
 - Services are business-process-oriented.
 - Business-level services are coarse-grained and can be easily mapped to business functions.
 - Coarse-grained services can be combined or assembled from lower-level, finegrained service primitives at run time.
 - Services are published in a standard fashion for discovery and execution.
 - Services can be used and reused by existing applications and systems.
- SOA permits the construction of scalable applications over the network.
- SOA supports asynchronous communications.
- SOA supports application-level conversations as well as process and state management.

SOA can significantly simplify and accelerate the development of new applications by invoking a variety of published services and organizing or orchestrating them to achieve the desired business functionality. Because SOA allows business-level services to be assembled at run time, developers do not have to design all possible variations of services in advance. This reduces the development time and helps minimize the number of errors in the application code.

One of the benefits of SOA is its ability to leverage the power and flexibility of Web Services across the enterprise by building loosely-coupled, standards-based applications that produce and consume services.

Introduction to Web Services

Web Services is another important concept that enables a shift in distributed computing toward loosely-coupled, standards-based, service-oriented architectures that help achieve better cross-business integration, improved efficiency, and closer customer relationships.

The short definition of *Web Services* offered here states that Web Services are *encapsulated*, *loosely-coupled*, *contracted* software objects that are published and consumed using standard interfaces and protocols.

The true power of Web Services lies in three related concepts that describe how Web Services change the fundamental nature of distributed computing:

- Web Services offer a standard way of supporting both synchronous and asynchronous messages—a capability essential to perform *long-running* B2B transactions.
- Web Services are loosely coupled, enabling a *reduction in the integration costs* as well as facilitating a *federation of systems*.
- Web Services support *coarse granularity* of the application programming interfaces (APIs). A coarse-grained interface rolls up the functions of many different API calls into a small number of business-oriented messages—a key to *business process management and automation*.

A good discussion on Web Services, SOA, and Web Services Architecture (WSA) can be found in the W3C Architecture documents.⁷ For simplicity, we'll define Web Services as encapsulated, loosely-coupled, contracted software objects that are published and consumed using standard interfaces and protocols.

Web Services

Web Services are encapsulated, loosely-coupled, contracted software objects that are published and consumed using standard interfaces and protocols.

A high-level view of a service-oriented architecture is shown in Figure 4-3.

Another, more structured view of the service-oriented reference architecture has been developed by a standards organization called the Organization for the Advancement of

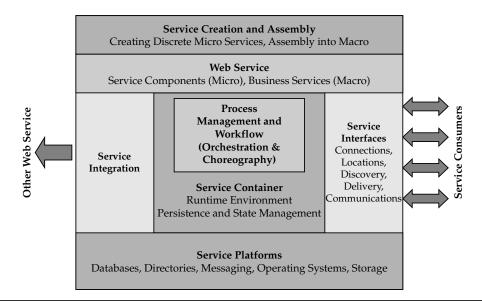


FIGURE 4-3 Service-oriented architecture

10/15/10 4·11·55 PM

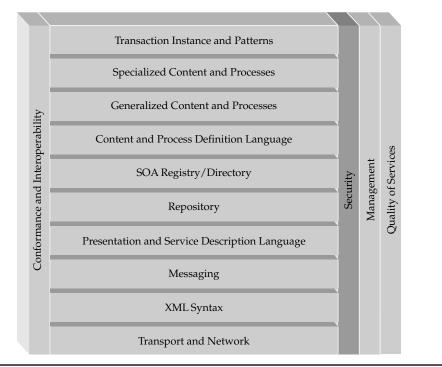


FIGURE 4-4 OASIS service-oriented reference architecture

Structured Information Standards (OASIS).⁸ One of the OASIS SOA reference architecture views is depicted in Figure 4-4.

SOA and Web Services are rapidly evolving from intra-enterprise usage to inter-enterprise communities of interest to general-purpose business-to-business environments, thus enabling significant reductions in the cost of integration among established business partners. SOA and Web Services have changed the way companies do business. For example, business transactions that use Web Services can offer new per-use or subscription-based revenue opportunities by exposing value-added services via public, Internet-accessible directories.

Combined with the benefits of the entity-centric transformations offered by the MDM Data Hub solutions, Web Services and SOA are powerful tools that can have a direct and positive impact on the design, implementation, and benefits of new entity-centric business strategies.

MDM and SOA MDM is a direct beneficiary and at the same time an enabler of the serviceoriented approach and Web Services. Indeed, MDM's complexity and variability of features and options all benefit from the ability to "assemble" or compose an MDM system from a pallet of available services by leveraging service reusability, service granularity, and loose coupling. SOA and Web Services by their very nature promote standards compliance as well as service provisioning and monitoring.

Moreover, SOA requires and enables service identification and categorization—features that represent a natural affinity to the capabilities of the MDM platform. Services categorization by itself is a valuable concept, because it helps define and understand service taxonomy, which in turn can guide architects and designers to the most effective placement and composition of services and their interdependencies. We show how these capabilities can be mapped onto the MDM system's services view later in this chapter.

In other words, there are significant synergies between MDM and SOA. At a high level, these synergies can be summarized as follows:

- SOA defines a fabric that helps deliver operational and analytical master data from the MDM system to all business application systems and users.
- MDM is a core engine of SOA master data services (MDS) and uses SOA components and principles to make master data available to it's applications and users via services.

CAUTION These synergies between MDM and SOA are not automatic. It is important to understand that SOA programs aimed at Master Data Management sometimes fail because the enterprise group responsible for the services framework does not align the SOA strategy, framework, and components with the enterprise data strategy and specifically the MDM strategy. MDM, with its cross-functional context, is a perfect area of application for SOA. When an SOA does not support MDM data services, the value of the SOA, even if it is implemented well from the technology perspective, is marginal.

Applying SOA principles to MDM solutions, we can construct a high-level serviceoriented view of the MDM Data Hub (see Figure 4-5). Here, the Data Hub acts as a services platform that supports two major groups of services: internal, infrastructure-type services that maintain Data Hub data integrity and enable necessary functionality; and external, published services. The latter category of services maps well to the business functions that can leverage the MDM Data Hub. These services are often considered business services, and the Data Hub exposes these external business services for consumption by the users and applications.

As we stated in the section on defining MDM architectural philosophy, we can organize all the services into a layered framework, with the services consumers on the top requesting and using the coarse-grained business services on the second layer. These published, business-level services invoke appropriate internal, fine-grained services in the layer(s) below. In this context, Data Hub internal services enable data access and maintain data integrity, consistency, security, and availability. The internal services interact with the Data Hub as a data service provider, and potentially with other data stores for the purpose of data acquisition, synchronization, and delivery.

The services invoke executable components and implement methods that perform requested actions. Following the principles of the service-oriented architecture and Web Services, the lower-level Data Hub services can be combined to form more coarse-grained, composite, business-level services that execute business transactions. In general, the service-oriented nature of the Data Hub platform would allow this service assembly to take place at run time. In this case, a Data Hub would help establish an appropriate execution environment, including the support for transactional semantics, orchestration/choreography, composition

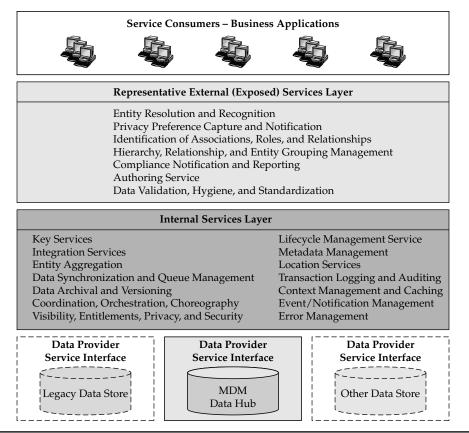


FIGURE 4-5 MDM Data Hub as a service platform

and remediation of failed actions, and security services. A full-function MDM Data Hub system would deliver these features through a dedicated set of internal services and functional components.

An important benefit of the SOA approach is that it allows for significant flexibility in the way the MDM services are selected and delivered, and the SOA approach does not require or imply that all services have be a part of a single product. In fact, SOA allows MDM designers to select best-in-class solutions for the services. For example, an organization may select an MDM vendor product implemented as a service platform but may decide to use an entity-matching engine and data integration services from another vendor based on features, price points, existing inventory, familiarity with the product, and a host of other reasons.

Although many MDM vendor products support service-oriented functionality, support for the scalability, flexibility, and granularity of the business services varies significantly from product to product.

We discuss additional details of the Data Hub as an instance of a service-oriented architecture in Chapter 6. We describe the Data Hub components view in the context of the reference architecture later in this chapter.

MDM and SOA Misconceptions One of the key differences between modern services-oriented MDM solutions and their ODS predecessors is that an MDM Data Hub is much more a service platform than just a data repository. The term "MDM Data Hub" is often inaccurately used to mean the same thing as the more traditional operational data stores of the 1980s and 1990s.⁹ Misusing the term adversely affects understanding of the modern design options of Enterprise Data Management (EDM) and MDM solutions that are enabled by MDM Data Hub systems.

There are some key characteristics and features of Data Hubs that often are underestimated or misunderstood by enterprise architects and systems integrators. Here are two of the most common misconceptions:

• Misconception 1: An MDM Data Hub is just another data repository or a database used for storage of cleansed data content, often used to build data warehousing dimensions.

Indeed, data must be cleansed and standardized before it is loaded into the Data Hub. For many professionals brought up on the concepts of operational data stores, data warehouses, and ETL (Extract, Transform, and Load), this is an undisputable truth. But it's not the only concern of Data Hub data content. Modern MDM architectures support a much more active approach to data than just the storage of a golden record. The Data Hub makes the best decisions on entity and relationship resolution by arbitrating the content of data created or modified in the source systems. Expressed differently, a Data Hub operates as a master data service responsible for the creation and maintenance of master entities and relationships.

The concept of a Data Hub as the enterprise master data service (MDS) applies the power of advanced algorithms and human input to resolve entities and relationships in real time. In addition, data governance policies and data stewardship rules and procedures define and mandate the behavior of the master data service, including the release of reference codes and code translation semantics for enterprise use.

The services nature of the MDM Data Hub provides an ideal way for managing data within an SOA environment. Using a hub-and-spoke model, the MDS serves as the integration method to communicate between all systems that produce or consume master data. The MDS is the hub, and all systems communicate directly with it using SOA principles.

Participating systems are operating in a loosely-coupled, federated fashion and are "autonomous" in SOA parlance, meaning that they can stay independent of one another and do not have to know the details of how other systems manage master data. This allows disparate system-specific schemas and internal business rules to be hidden, which greatly reduces tight coupling and the overall brittleness of the MDM ecosystem. It also helps to reduce the overall workload that participating systems must bear to manage master data.

Misconception 2: The system of record must be persisted in the MDM Data Hub.

The notion of a Data Hub as a data repository erroneously presumes that the single version of the truth, the golden record, must be persisted in the Data Hub. The notion of the MDM Data Hub as a service platform does not make this presumption. Indeed, as soon as the master data service can deliver master data to the enterprise, the Data Hub may persist the golden record or assemble it dynamically instead.

One of the arguments for persistently stored master data in the Data Hub is that performance for master data retrieval will suffer if the record is assembled dynamically on request. The reality is that the existing Data Hub solutions have demonstrated that dynamic master data content can be assembled with practically no performance impact if the master data model is properly implemented.

One of the advantages of dynamically assembled records is that the Data Hub can maintain multiple views of the master data aligned with line-of-business and functional requirements, data visibility requirements, tolerance to false positives and negatives, and latency requirements. Mature enterprises increasingly require multiple views for the golden record, and the dynamic record assembly works better to support this need.

Conversely, we can offer a strong argument in favor of a persistently stored master data records. This argument is driven by the need to support the history and auditability of the master data life cycle. There are at least two major usage patterns for the history of master data:

- The first pattern is driven by audit requirements. The enterprise needs to be able to understand the origin, the time, and possibly the reason for a change. These audit and compliance requirements have to be supported by the Data Hub at the attribute level. MDM solutions that maintain the golden record can dynamically address this need by supporting the history of changes in the source system's record content.
- The second usage pattern for history support results from the need to support database queries on data referring to a certain point in time or certain time range—for example, what was the inventory on a certain date, or sales over the second quarter? A classic example of this type of history support is the implementation and management of slowly changing dimensions in data warehousing. In order to support this usage pattern, the golden version of the master record must be persisted. It is just a question of location. Many enterprises decide this question in favor of data warehousing dimensions while avoiding the persistently stored golden record in the Data Hub.

In short, modern MDM Data Hub systems function as active components of serviceoriented architecture by providing master data services, rather than being passive repositories of cleansed data. This consideration should help the enterprise architects and systems integrators build sound Master Data Management solutions. An additional discussion about these differences can be found in Chapter 6.

Architecture Viewpoints of Various MDM Classification Dimensions

As we defined in Chapter 1, MDM addresses complex business and technical problems and, as such, is a complex, multifaceted framework that can be described and viewed from various angles. The amount of information about MDM goals, benefits, design viewpoints, and challenges is quite large, and in order to make sense of various, sometimes contradictory assertions about MDM, we introduced several MDM classification dimensions that allow us to organize available information, and we discussed various aspects of MDM according to

a well-defined structure. In this section, we consider the architectural implications of various classification dimensions introduced in Chapter 1, as follows:

- The Design and Deployment dimension (consumption and reconciliation architecture viewpoint)
- The Use Pattern dimension
- The Information Scope or Data Domain dimension

MDM practitioners and industry analysts see these dimensions as persistent characteristics of any MDM solution, regardless of the industry or master data domain.

MDM Design and Deployment Dimension

The Design and Deployment viewpoint addresses MDM consumption and reconciliation architecture concerns, and the resulting MDM architecture styles. Armed with the architecture framework approach, we can recognize that these "styles" represent architecture viewpoints that determine the way the MDM system is intended to be used and be kept reliably in sync with its data providers (data sources) and data consumers. These viewpoints represent an intersection of the functional and data dimensions of the enterprise architecture framework at the logical, conceptual, and contextual levels. The resulting design constructs are a direct consequence of the different breadth and depth of the MDM data model coverage. We will discuss master data modeling in more detail in Chapter 7.

The architecture styles vary in the context of other dimensions of the enterprise architecture framework, including the organizational need and readiness to create and fully deploy a new system of records about customer data. And, of course, these architecture styles manifest themselves in different service-oriented architecture viewpoints.

Let's briefly describe the four predominant MDM architecture styles in the context of master data scope management, consumption, and reconciliation services. These styles have been introduced by several prominent industry analysts, including the Gartner Group.¹⁰ We discuss the implementation concerns of these architecture styles later, in Part IV of the book.

MDM Architecture Styles

The MDM architecture, design, and deployment styles include the following:

- External reference
- Registry
- Reconciliation engine
- Transaction hub

The underlying principle behind these styles is the fact that an MDM Data Hub data model may contain *all* data attributes about the data domain it manages, or just *some* attributes, while other attributes remain in their original data stores. It is logical to assume that the Data Hub can be the "master" of those master entities whose data attributes it manages or just arbitrates the entities and attributes across operational systems where the master data is created and maintained. This assumption is one of the drivers defining the MDM architecture styles. Let's look at this issue in detail.

External Reference Style In this case, an MDM Data Hub is a reference database pointing to all source data stores but does not usually contain actual data for a given domain—for example, customer data for a customer domain, product for product domain, and so on:

- This is the most extreme case, where a Data Hub contains only a reference to the source or system of record data that continues to reside in the legacy data stores. In this case, the Data Hub acts as a special "directory" and points to the master data that continues to be created and updated by the existing legacy applications. This design option, known as the "External Reference Data Hub," is the least complex of the Data Hub styles.
- One of the main architecture concerns of this style is the ability of the MDM Data Hub to maintain accurate, timely, and valid references to the master data at all times, which may require design focus on a reliable, just-in-time interconnection between source systems and the Data Hub, perhaps by using an enterprise-class messaging mechanism.
- A significant limitation of this architectural style is that the Data Hub does not hold any attributes, even those needed for matching and entity resolution. The Data Hub service responsible for matching has to access matching attributes across multiple systems in a federated fashion.

Even though this design is theoretically possible and a few attempts have been made to implement it, federated matching has been proven ineffective and most MDM Data Hub vendors discontinued its support.

Registry Style This style of the MDM Data Hub architecture represents a Registry of unique master entity identifiers (created using identity attributes). It maintains only the identifying attributes. These attributes are used by an entity resolution service to identify which master entity records should be linked because they represent the same entity (i.e., customer, product, location, and so on). The Data Hub matches and links the records that share the same identity. The Data Hub creates and maintains links with data sources that were used to obtain the identity attributes. The MDM Data Hub exposes a service that returns a fully assembled holistic entity view to the consuming application either as retrieval or an assembly operation (for example, a customer, at run time). Using MDM for customer domain as an example, a Registry-style Data Hub should support the following features:

- Maintain some, at least matching customer profile attributes that it uses to generate a unique customer identifier. Such attributes may include customer name, address, date of birth, and externally assigned identifiers (social security number, an employer identification number, a business reference number such as a DUNS number, and so on).
- Automatically generate and maintain links with all upstream systems that maintain data about the customers. Consuming applications query the Registry for a given customer or a set of customers, and the Registry would use its customer identification number and legacy pointers or links and record merge rules to allow the application to retrieve and construct a view of the customer from the underlying data.
- Act as the "master" of the unique identifiers, and support arbitration of data conflicts by determining which attribute values in the source systems are better than others by applying attribute survivorship rules across multiple systems.

A limitation of this MDM architecture style is that it relies on the data available in the operational systems to assemble the best possible view of data. The Data Hub is not used for data entry and does not own the master data, but rather arbitrates the values that should be available in the operational source systems to be displayed by the Data Hub. If the data is not available in the source systems, the Registry-style Data Hub cannot create the right attribute values by itself. The records and correct attribute values have to be created and maintained in one of the feeding operational systems. Then the Data Hub will process the changes originated in the source system in real time and display an improved view of the benchmark record.

Reconciliation Engine This MDM architecture style is a system of record for some entity attributes; it provides active synchronization between itself and the legacy systems.

- In this case, the Data Hub is the master for those data attributes that it actively maintains by supporting authoring of master data content. The Reconciliation Engine Data Hub style relies on the upstream source systems to maintain other data attributes. One implication of this approach is the fact that *some* applications that handle source or master data may have to be changed or redesigned based on the business processes, application interfaces, and the data they use. The same is true for the corresponding business processes. The other implication is that the Data Hub has to maintain, create, and change those data attributes for which it is the master. The Data Hub has to propagate changes for these attributes to the systems that use these attributes. The result is a data environment that continuously synchronizes the data content among its participants to avoid data inconsistencies.
- A shortcoming is that the complexity of synchronization increases as some of the data attributes maintained in the Data Hub are derived from the data attributes maintained in other systems. For example, a typical Reconciliation Engine–style Data Hub for customer domain has to create and maintain unique customer identifications as well as references to the legacy systems and data stores where the customer data is sourced from or continues to reside.

This architecture style is more sophisticated than the Registry-style Data Hub, and in many situations is a viable evolutionary step toward the full Transaction Hub.

Transaction Hub This is the most sophisticated option, in which the Data Hub becomes the primary source of and the system of record for the entire master data domain, including appropriate reference pointers:

- This is the case where the Data Hub maintains practically *all* data attributes about the entity. For a given entity domain, such as a customer domain (individuals or businesses), the Data Hub becomes a "master" of the master entity information, and as such should be the source of all changes to any attribute about the master entity. In this case, the Data Hub has to be engineered as a complete transactional environment that maintains its data integrity and is the sole source of changes that it propagates to all downstream systems that use the customer data.
- The Transactional Hub has some profound implications for the overall environment, the existing applications, and business processes already in place. For example, an existing account maintenance application may have to undergo modifications to update the Data Hub instead of an existing legacy system, and appropriate synchronization mechanisms have to be in place to propagate and apply the changes from the Data

Hub to some or all downstream systems. Moreover, most of the previously deployed transactions that change entity information should be redesigned to work directly with the Data Hub, which may also change existing business processes, workflows, and user navigation. This is the most complex case, which is known as a Full Transaction Hub.

- Practically speaking, the intrusiveness of the Transaction Hub style makes it a viable choice mostly in two scenarios:
 - When dealing with a new enterprise that does not have a massive legacy infrastructure maintaining the master entity the Data Hub is supposed to resolve.
 - When the current processes and applications already manage the master entity as a Transaction-style Data Hub. In this scenario, the new Data Hub is built to replace the existing master entity management system with a new system (for example, a customer-centric solution). For instance, it can be the case where the enterprise has already been using a home-grown Transaction-style MDM Data Hub and is looking to replace it with a more advanced vendor solution.

With the exception of the first, the External Reference style, these architecture and design styles have one thing in common—they define, create, and manage a centralized platform where master data is integrated either virtually (Registry) or physically (Reconciliation Engine and Transaction Hub) to create a reliable and sustainable system of record for master data.

MDM and Use Pattern Dimension

The Use Pattern classification dimension differentiates MDM architectures based on how the master data is used. We see three primary use patterns for MDM data usage: Analytical MDM, Operational MDM, and Collaborative MDM.

- Analytical MDM supports business processes and applications that use master data
 primarily to analyze business performance and provide appropriate reporting and
 analytical capabilities, often by directly interfacing with business intelligence (BI) tools
 and packages. Analytical MDM tends to be read-mostly, it usually does not change or
 create source data in the operational systems, but it does cleanse and enrich data in the
 MDM Data Hub. From the overall system architecture view, Analytical MDM can be
 architected as a feed into the data warehouse and can create or enrich an accurate,
 integrated view of the master data inside the data warehouse. BI tools are typically
 deployed to access this cleansed, enriched, and integrated data for reporting, perform
 deep analytics, and provide drill-through capabilities for the required level of detail.
- Operational MDM allows master data to be collected, changed, and used to process business transactions; Operational MDM is designed to maintain the semantic consistency of the master data affected by the transactional activity. Operational MDM provides a mechanism to improve the quality of the data in the operational systems, where the data is usually created. By design, Operational MDM systems ensure that the accurate, single version of the truth is maintained in the MDM Data Hub and propagated to the core systems used by existing and new processes and applications.
- Collaborative MDM allows its users to author master data objects and collaborate in the process of creation and maintenance of master data and its associated metadata.

These Use Pattern–based architecture viewpoints have common concerns and often use common or similar technologies, especially the components of technology related to data extraction, transformation, and load, as well as data quality.

At the same time, we can clearly see how the architectural implications of these three Use Pattern dimensions impact the way the MDM Hub has to handle data synchronization concerns, implement cross-application interoperability, deliver data changes to upstream and/or downstream systems, detect and improve data quality issues, and enable and support data governance processes.

Data Domain Dimension

The Information Scope or Data Domain dimension describes the primary data domain managed by the MDM solution. In the case of MDM for the customer data domain, the resulting solution is often called Customer Data Integration, or CDI. In the case of MDM for product data domain, the solution is known as Product Information Management, or PIM. Other data domains may not have formal acronym definitions yet, but could have an impact on how the MDM solution is designed and deployed. Primary architectural implications related to implementing customer, product, or other domains include:

- Design for entity resolution and identification. Techniques for these data domains can vary drastically based on the requirements for semantic consistency, speed, accuracy, and confidence.
- Ability to acquire and manage sources of external entity references, such as authoritative sources of individual names and addresses, business names, as well as identifiers and industry classifications (for example, D&B DUNS numbers).
- Information security and privacy concerns that apply differently to different data domains based on a particular risk profile of a given data domain within the context of business requirements as well as those governed by a variety of rules, policies, and governmental regulations.

Reference Data and Hierarchy Management

When we discuss the architectural implications of an MDM solution in the context of the data it manages, we need to recognize that the data scope alone does not address all variations of what data needs to be managed in what way. For example, most MDM implementations deal with creating a master environment of *reference* data, such as product reference, account reference, customer reference, and so on. However, it is not unusual for an organization to try to build an authoritative master data environment that supports enterprise-wide business attributes, such as customer revenues, expenses, risk exposure, and so on. Technically speaking, this is not traditional reference data, and the MDM Data Hub architecture should provide for features, functions, and services that can calculate, maintain, and ensure the quality of these key business metrics. Clearly, this adds an additional layer of complexity to an already complex system. This is where proven architecture patterns for creating such metrics can be inherited from existing business systems and "adopted" into the MDM Data Hub.

MDM and Hierarchy Management Many business problems addressed by the MDM architecture include the management of data domain hierarchies. It is a common situation when an organization manages multiple views of the business based on a specific business focus, such as marketing view of customers, financial views of a global organization, various views of products, and so on. In these cases, we see an organizational hierarchy that consists of a parent (for example, legal entity) and multiple dependents (for example, accounts or other legal entities). Similarly, businesses tend to structure their sales organizations based on either products or geographies or cost centers. The challenge here is that these hierarchies are

not static over time, and can and do change with business restructuring, mergers and acquisitions, new product introductions, and other events. Several formal definitions of hierarchies are available, but the following working definition of hierarchies is most relevant to general data management, and Master Data Management in particular.

In the context of MDM, we define a *hierarchy* as an arrangement of entities (parties, accounts, products, cost centers, and so on) where entities are viewed in relationship to each other as "parents," "children," or "siblings/peers" of other entities, thus forming a conceptual tree structure where all leaf nodes in the hierarchy tree can be rolled into a single "root."

Further, the entities of a given domain can often support several hierarchical arrangements based on a particular classification schema (legal entity level, geography/ location, role/rank, scope of authority, and so on). A direct consequence of this fact is that changes in a classification schema or the introduction of another schema will result in the creation of a different hierarchy, sometimes referred to as an alternate hierarchy.

In order to create and maintain an authoritative, verifiable system of record, an MDM system has to be able to recognize and manage hierarchies based on the classification schemas; to compare, match, and link master entities that may exist at different levels of hierarchy; to manage the creation, maintenance, and versioning of different alternative hierarchies; and to provide relevant and timely changes in the hierarchies of reference data to the MDM users and consuming applications.

MDM Hierarchy Management and Data Warehousing The discussion on hierarchy management of reference data offered in the preceding section is particularly relevant to the relationship between MDM and data warehousing. Let's compare the principles of hierarchical structures with the concepts of facts and dimensions in the data warehousing discipline.¹¹ Indeed, the notion of a hierarchy applies directly to the dimensions in a data warehouse's data model, frequently referred to as a dimensional data model in the form of a "star" or "snowflake" schema, with the fact entities organized in a set of central tables that are "surrounded" by dimension tables, where the dimensional data contains attributes used as keys that point to the facts in the Fact Table.¹² For example, a customer data warehouse may contain information about customer account values (facts) and dimensions such as customer identifiers, customer locations, and time. As the dimensional attributes change, the facts may change or new facts may get created. And in cases where dimensional values change infrequently, the data warehousing discipline recognizes the concept of Slow Change Dimensions, or SCD, the constructs that allow a data warehouse to maintain the historical view of the values of the facts (sometimes referred to as "time travel").

Hierarchies and Data Dimensions

In the context of denormalized dimensional data models such as the star or snowflake schemas widely used in data warehousing, *hierarchies* are arrangements of records in the data model's dimensions.

Data warehousing is a complex and mature technical discipline, and a detailed discussion of this topic is beyond the scope of this book. However, we briefly discuss the relationship between MDM hierarchy management and data warehousing concepts for the following reasons:

 As stated in preceding chapters, data warehousing is one of the predecessor technologies to MDM.

• In many instances, an MDM system is implemented "upstream" from data warehouses and data marts that are used to collect and aggregate master data and to provide reporting, analytical, and business intelligence capabilities to support an organization's business and financial management needs.

Therefore, it is important to understand what MDM architecture features are required to support a large multidimensional data warehouse as a downstream system. Architecturally, these features are organized into a collection of hierarchy management services, and these services are used to maintain the integrity and accuracy of various hierarchies; to work in conjunction with entity resolution services to properly recognize, match, link, and aggregate entities in accordance with their hierarchical relationships; and to enable the efficient delivery of hierarchy changes to appropriate downstream consuming applications. Hierarchy management services and their uses are discussed in more detail in Chapters 5 and 6.

NOTE The classification domains introduced in this chapter have clear implications on MDM architecture. Specifically, although MDM architecture styles defined by these various viewpoints are different, they have many things in common. In reality, it is not unusual to find an MDM implementation that exhibits properties of one or more architecture styles at the same time—for example, acting as a Registry for some master data domain while being a coexistence-style MDM Data Hub for others. Likewise, aside from some very specific capabilities and implementation patterns, the architecture of an MDM Data Hub for a customer domain is significantly similar to that of the product domain, and so on. The latter is one of the enablers of evolving MDM from a single-domain master data management solution to a multidomain Data Hub operating on the same technology platform.

The relevance of this note is in that it points to the significant flexibility and versatility of the MDM architecture. It also confirms our previous discussion on the value of the architecture frameworks and architecture viewpoints that provide different insights into the same large and complex system.

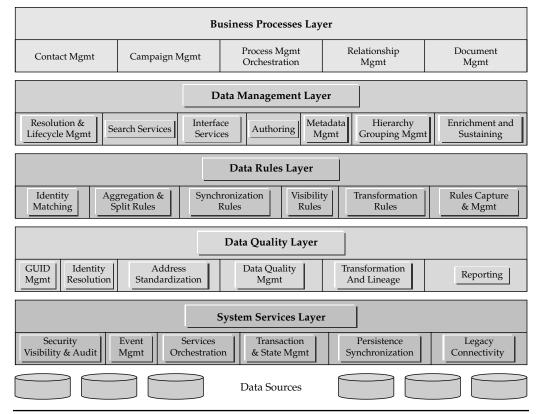
Reference Architecture Viewpoint

In the previous sections we looked at the key components and architecture viewpoints of the MDM architecture, and showed its complexity and the variety of approaches you could take to select, build, and implement an MDM solution.

However, this discussion would not be complete if we didn't consider another key architectural artifact—a reference architecture viewpoint. Reference architecture is one of the best-known complexity-reducing architecture viewpoints. Let's informally define reference architecture as follows:

Reference architecture is a high-level abstraction of a technical solution to a particular problem domain; it is a set of interlinked components, services, processes, and interfaces organized into functional layers, where each layer provides services to the layers above and consumes services from the layers below. As such, reference architecture does not define specific technologies or implementation details.

The key value proposition of reference architecture is in its ability to help architects and designers to define the functionality and placement of all architecture components in the context of the overall system and problem domain. In other words, reference architecture provides a blueprint and helps create a set of patterns for designing specific solution/system





components and their interactions. That is why a reference architecture viewpoint is such a powerful tool for designing systems of MDM-level complexity and interdependencies.

Using this definition of the reference architecture, we can define an MDM reference architecture viewpoint as an industry- and data domain–agnostic architectural multilayered abstraction that consists of services, components, processes, and interfaces (see Figure 4-6).

As an instance of an SOA, this MDM reference architecture contains a significant number of key service components. Some of these services are discussed in further detail in Chapters 5 and 6 of the book, but we offer a brief list of higher-level service layers in this section for the purpose of completeness.

- The Data Management layer includes:
 - Interface services, which expose a published and consistent entry point to request MDM services.
 - Entity resolution and lifecycle management services, which enable entity recognition by resolving various levels of identities, and manage life stages of master data by supporting business interactions including traditional Create, Read, Update and Delete (CRUD) activities.
 - Search services, for easy access to the information managed by the MDM Data Hub.

- Authoring services, which allow MDM users to create (author), manage, customize/change, and approve definitions of master data (metadata), including hierarchies and entity groups. In addition, Authoring services enable users to manage (CRUD) specific instances of master data.
- The metadata management service, which provides support for data management aspects of metadata creation, manipulation, and maintenance. The metadata management service supports a metadata repository and relies on and supports internal Data Hub services such as attribute and record locator services and even key generation services.
- Hierarchy, relationships, and groupings management services, which deliver functions designed to manage master data hierarchies, groupings, and relationships. These can process requests from the authoring services.
- Enrichment and sustaining services, which are focused on acquiring and maintaining the correct content of master data, controlled by external data references and user-driven adjustments.
- The Data Rules layer includes key services that are driven by business-defined rules for entity resolution, aggregation, synchronization, visibility and privacy, and transformation.
- The Data Quality layer includes services that are designed to validate and enforce data quality rules, resolve entity identification and hierarchical attributes, and perform data standardization, reconciliation, and lineage. These services also generate and manage global unique identifiers as well as provide data quality profiling and reporting.
- The System Services layer includes a broad category of base services such as security, data visibility, event management (these are designed to react to predefined events detected within the master data by triggering appropriate actions), service management (orchestration, choreography), transaction and state management, system synchronization, and intersystem connectivity/data integration services, including Enterprise Information Integration services for federated data access (discussed in more detail in Chapters 5 and 6).

Despite this long list of services defined in the MDM reference architecture viewpoint, at a high level this reference architecture appears to be deceptively simple. However, a closer look will reveal that most of the components and services of the architecture have to be present in order to accomplish the goal of creating an MDM system. Moreover, many of these components are complex objects that, in turn, contain many lower-level components and services. We will offer a more detailed discussion of some of the components in the subsequent chapters of the book. To set the stage for the detailed discussion, we will organize the components, services, and layers of this high-level conceptual reference architecture into two major groups: traditional architecture concerns of information management and new, advanced concerns driven by the goals of Master Data Management.

The traditional architecture concerns focus on the area of data and data management. These concerns include data architecture and data modeling; data extractions, transformation, and loading; metadata repository and metadata management; database management system performance and scalability; transaction management; backup and recovery; and others (see Figure 4-7).

Advanced MDM-specific concerns include areas such as identity recognition, matching and generation of global unique entity identifiers, persistence of entity identification,

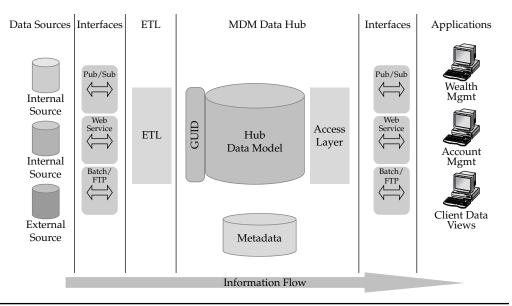


FIGURE 4-7 Traditional data-centric view of MDM architecture.

rules-based and data content–based synchronization to/from legacy, reconciliation and arbitration of data changes, data security and data visibility, service implementation and management integration with legacy environments, and many others (see Figure 4-8).

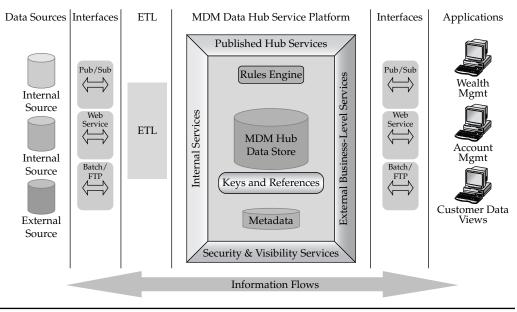


FIGURE 4-8 Adding new MDM-specific architecture concerns

We discuss these traditional and advanced concerns of the MDM architecture in more detail in the remaining chapters of this part of the book and also in Part III. The material in these chapters offers additional insights and architecture viewpoints that should help MDM managers, designers, and implementers to achieve measurable results using a structured and disciplined architecture approach.

References

- 1. Zachman Enterprise Architecture Framework, Zachman's Institute for Framework Advancement (ZIFA).
- 2. http://homepage.newschool.edu/het//profiles/pareto.htm.
- 3. Dreibelbis, Allen, Ivan Milman, Paul van Run, Eberhard Hechler, Martin Oberhofer, and Dan Wolfson. *Enterprise Master Data Management: An SOA Approach to Managing Core Information*. IBM Press (2008).
- 4. "IBM SOA Foundation, An Architectural Introduction and Overview," 2005. http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/.
- 5. http://www.w3.org/TR/ws-arch/.
- 6. http://www.infoq.com/news/2008/09/Orchestration.
- 7. http://www.w3.org/TR/ws-arch/.
- 8. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.
- Dubov, Larry, "Guest View: Old Thinking Does a Disservice to New Data Hubs," SD Times, October 15, 2009. http://www.sdtimes.com/GUEST_VIEW_OLD_ THINKING_DOES_A_DISSERVICE_TO_NEW_DATA_HUBS/By_LARRY_ DUBOV/About_DATABASES/33828.
- 10. www.gartner.com.
- 11. Berson, Alex, and Stephen J. Smith. *Data Warehousing*, *Data Mining*, & OLAP. McGraw-Hill (1997).
- Kimball, Ralph, Margy Ross, Bob Becker, and Joy Mundy. Kimball's Data Warehouse Toolkit Classics: The Data Warehouse Toolkit, 2nd Edition; The Data Warehouse Lifecycle, 2nd Edition; The Data Warehouse ETL Toolkit. Wiley (2009).

CompRef8 / Master Data Management and Data Governance / Berson / 458-4