



PART



Architectural Considerations

CHAPTER 4

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

78 Part II: Architectural Considerations

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.

4

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

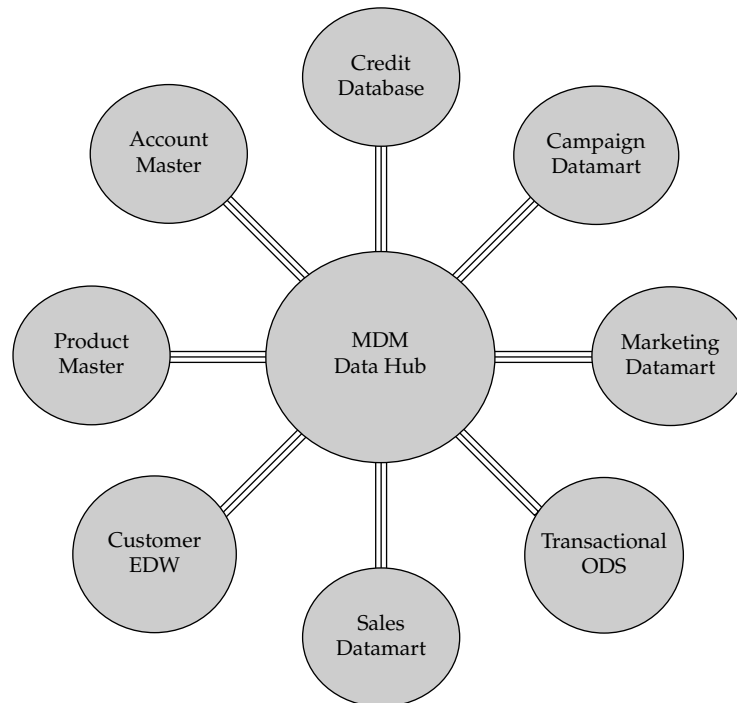
80 Part II: Architectural Considerations

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.

FIGURE 4-1
MDM Customer/
Product Data Hub



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

82 Part II: Architectural Considerations

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 architecture-focused 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 data-integration 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.

84 Part II: Architectural Considerations

- **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.

86 Part II: Architectural Considerations

- 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.

88 Part II: Architectural Considerations

| | What | | How | | Where | | Who | | When | | Why | |
|--|--|---|--|---|--|--|-----|--|------|--|-----|--|
| | Data | Function | Network | People | Time | Motivation | | | | | | |
| Scope (contextual) Planner | List of things important to the business Entity = Class of business thing | List of process the business performs Process = Class of business process | List of locations in which the business operates Node = Major business location | List of organizations important to the business People = Major organizational unit | List of events/cycles significant to the business Time = Major business event/cycle | Lists of business goals/strategies Ends = means = Major business goals/strategy | | | | | | |
| Business Model (conceptual) Owner | Eg. Semantic model Entity = Business entity relationship = Business relationship | Eg. Business process model Process = Business process IO = Business resources | Eg. Business logistics system Node = Business location Link = Business linkage | Eg. Work flow model People = Organizational unit Work = Work product | Eg. Master schedule Time = Business event Cycle = Business cycle | Eg. Business plan End = Business object Means = Business strategy | | | | | | |
| System Model (logical) Designer | Eg. Logical data model Entity = Data entity Relationship = Data relationship | Eg. Application architecture Process = Application function IO = User views | Eg. Distributed system architecture Node = I/S function (processor, storage, etc.) Link = Line characteristics | Eg. Human interface architecture People = Role Work = Dataorable | Eg. Processing structure Time = System event Cycle = Processing cycle | Eg. Business role model End = Structural assertion Means = Action assertion | | | | | | |
| Technology Model (physical) Builder | Eg. Physical data model Entity = Segment/table/etc Relationship = Pointer/key/etc. | Eg. System design Process = Computer function IO = Data elements/sets | Eg. Technology architecture Node = Hdw/system software Link = Line specifications | Eg. Presentation architecture People = User Work = Screen formats | Eg. Control structure Time = Execute Cycle = Component cycle | Eg. Role design End = Condition Means = Action | | | | | | |
| Detailed Representations (out-of-context) Subcontractor | Eg. Data definition Entity = Field Relationship = Address | Eg. Program Process = Language statement IO = Control block | Eg. Network architecture Node = Address Link = Protocol | Eg. Security architecture People = Identity Work = Job | Eg. Timing definition Time = Interval Cycle = Machine cycle | Eg. Role specification End = Sub-condition Means = Step | | | | | | |
| Functioning Enterprise | Data | Function | Network | Organization | Schedule | Strategy | | | | | | |

FIGURE 4-2 Zachman's enterprise architecture framework

- 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.

90 Part II: Architectural Considerations

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.

92 Part II: Architectural Considerations

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, fine-grained 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.

Chapter 4: MDM Architecture Classifications, Concepts, Principles, and Components 93

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

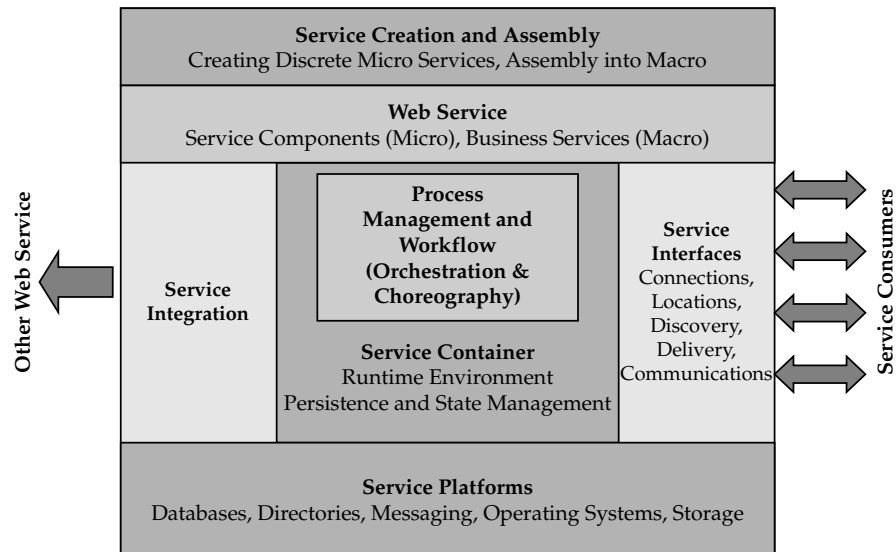


FIGURE 4-3 Service-oriented architecture

94 Part II: Architectural Considerations

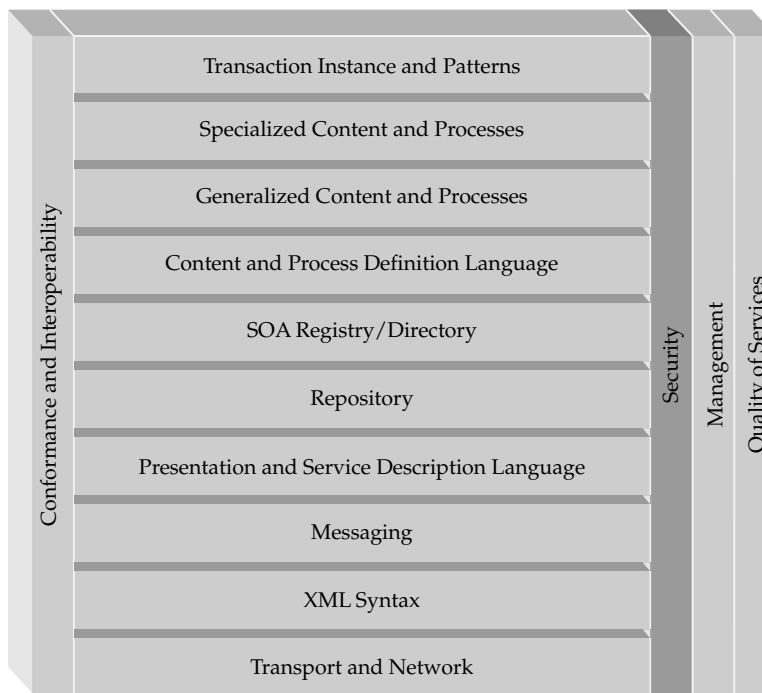


FIGURE 4-4 OASIS service-oriented reference architecture

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 service-oriented 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.

Chapter 4: MDM Architecture Classifications, Concepts, Principles, and Components 95

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 its 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 service-oriented 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

