

GAP ANALYSIS: THE CASE FOR DATA SERVICES

Data services gather and integrate data from many different sources in real time, bridging gaps to create consolidated information views while hiding data source complexity from applications. A data service approach makes data access and integration scalable by streamlining development and maintenance, whether applications speak Web services, SQL, or both.





TABLE OF CONTENTS

ABSTRACT	Page 3
1. THE NEED FOR DATA SERVICES	Page 4
The Growth of SOA	Page 4
SOA Meets the Reality of Enterprise Data	Page 5
Bridging the Gap	Page 6
Working at a Higher Level: The New Data Tier	Page 8
2. ANATOMY OF A DATA SERVICE	Page 9
What is a Data Service	Page 9
Application Access to Data	Page 10
Data Integration on Demand	Page 11
Reuse of Data Services	Page 11
Models and Layers of Abstraction	
Ease Integration and Reuse	Page 12
Data Services Complement	
Other SOA Technologies	Page 12
Controlling Data Services:	
The Data Service Lifecycle	Page 13
3. WHAT IS A DATA SERVICE MANAGEMENT SYSTEM	Page 14
Creating Data Services: The Design Environment	Page 14
Managing Metadata: From Data Source	
Descriptions to Standard Data Models	Page 15
The Runtime Environment: Control and Efficiency	Page 15
CONCLUSION	Page 16

ABSTRACT

The broad, cross-functional scope of newer business applications, changing regulatory and security requirements, and the demanding, real-time nature of today's business interactions are making the roll-out of new service-oriented architecture (SOA) based applications exceptionally challenging. Web services protocols directly address component-based applications, but what about the data? Most corporate data is currently stored in departmental relational databases. Database schemas have been optimized for lines of business, but many new applications require data across business lines. Meeting these new information requirements requires not only streamlining access to data and ensuring efficient delivery of bits over the network, but also transforming and integrating data from multiple, diverse sources to create new consolidated views of information that are meaningful for today's applications and decision-makers—views of data that drive key initiatives in customer service, marketing, financial reporting, and compliance. Creating this critical information requires gathering data from many sources. On each project, work must be done to bridge many different gaps—between relational or legacy application data sources and new XML-based data structures, between databases and non-relational data sources with different, overlapping semantics, and between these existing data sources and the data structures, semantics, vocabularies, and formats needed by new applications.

Use of many different data sources can place an undue burden on applications. Adoption of SOA and Web services, even when accompanied by industry-standard XML schemas, does not in itself address the need for uniform data access or the means by which mediation between different semantics and vocabularies should occur. As a result, many projects incorporate extensive custom coding efforts to transform and integrate data. The high volume of work required leads to extended development cycles, and the complexity of data transformation and integration requirements leads—in spite of the high skill level of developers—to code that is necessarily complex and difficult to maintain. This labor-intensive integration effort exacerbates the organization's perpetual struggle to standardize data—to create and manage the consistent, up-to-the-minute "single version of the truth" urgently needed at all levels of operations. Customer service, marketing, regulatory compliance, new product development, and financial operations can suffer when consistent, consolidated views of data are not available.

Forward-thinking organizations are overcoming these challenges today by creating a reusable layer of data services to mediate between data sources and applications, either within a single critical application project or across a division or enterprise. By hiding the complexity of data access, transformation, aggregation, and integration behind a standard data service layer (whether Web service-based, SQL-based, or both), it is possible to decouple data sources from applications and simplify development and integration of applications that use the data. Data services lead to shorter project timelines, less costly projects, and reduced maintenance costs. They can easily provide interfaces that conform to corporate or industry standard data models (expressed as either entity-relationship models or XML schemas), enabling developers to work with a standard, canonical view of data regardless of the number and type of data sources used to create that view.

A data service management system such as MetaMatrix Enterprise Data Services Platform allows organizations to create, deploy, execute, and manage data services that encapsulate data access and integration, foster scalable development, and support the organization's SOA evolution. A system flexible enough to support both SQL-based and Web service-based data interactions will ease architectural evolution and serve the broadest array of projects, while a system that allows multiple levels of data abstraction can provide the flexibility developers and analysts need to express the required semantic resolution, support the highest degree of reuse, and make explicit the organization's standard data model. Finally, a design environment that enables developers to define data services using a declarative, graphical mapping process avoids custom coding and accelerates development. With a data services layer in place, the organization can complete projects faster and at lower cost, use information more effectively, and ultimately reduce the total cost of ownership for enterprise data and applications.

1. THE NEED FOR DATA SERVICES

THE GROWTH OF SOA

In response to growing complexity, many application projects are choosing to implement service-oriented architectures. The trend continues upward as early adopters begin to experience the promised streamlining of development cycles.

Substantial savings in time and effort have been documented, chiefly based on the ability to assemble applications from loosely coupled components with well-defined interfaces, reducing the need to reinvent the wheel and increasing the efficiency of development in general. When a skilled developer creates a high-quality component that others can discover easily and reuse often, everybody wins. Investments in development and testing produce greater returns, maintenance times and costs are lower, and projects are completed with less time and more modest skill sets. Of course, realizing these benefits requires component interfaces to be standard, self-documenting, and easy to understand, so a developer who did not create the component can quickly grasp how to plug it in and use it. Without straightforward interfaces developers go back to reinventing the wheel—a process that has become prohibitively expensive.

Loose coupling of components enables quick assembly of composite applications and can make it much easier for two different systems to interoperate regardless of underlying technologies. The result is less overall time spent on development and maintenance tasks and a larger number of projects that can be completed (or at least partially completed) using existing systems and components.

This is good news so far. But how are service-oriented architectures to interact with enterprise data? How can an organization bring the same kinds of improvements to data access and integration, as well as the development of business applications?



Where can you turn for integration help?

- ETL approaches have stretched the data replication paradigm to its limit, and many companies are finding it hard to manage the explosive growth of data marts and replicas. Data redundancy must be controlled if organizations are to avoid spiraling data management costs.
- EAI tools have migrated to composite application construction based on XML, leaving behind the ability to optimize interactions with data sources.
- ESB tools incorporate the ability to integrate data at an application level; however, this approach lacks crucial optimizations needed to deliver data efficiently. The principles of service-oriented architecture lead us toward a data service layer that hides data source complexity from applications.
- A data service layer offers substantial benefits to organizations at any point on the SOA adoption spectrum, helping to decouple applications from data sources.

SOA MEETS THE REALITY OF ENTERPRISE DATA

As a new application is created, it frequently must use one or more existing legacy systems or existing data sources. A system with straightforward inputs and outputs can perhaps be “wrapped” so that it presents a standard, service-oriented interface. Data sources offer a different challenge; the new application may require or expect data in a format different than the way it is stored. Is it sufficient to simply “wrap” a data source with a service interface? This may solve one problem—the need for a standard interface—but it leaves an entire range of challenges for the developer who must then decide how to incorporate data into a new system to meet new requirements. Consider, for example, the developer/integrator for a new customer service system that requires data to be gathered from several existing systems to create a “single view of the customer.” Even using a Web services standard, how is this single view to be achieved?

To answer this question, consider that over the decades, designers and developers have evolved increasingly more efficient ways of working with data. Initially data was carried in the computer’s memory. Variables were defined and populated by the program to produce a result. Then came data “master files” stored separately from the application (on punched cards, tape, and finally disk storage), followed by database systems based on hierarchical, network, and finally relational access structures. Each innovation in data manipulation had three core design imperatives: maintain data integrity, maximize efficiency of storage and retrieval, and keep development tractable for application developers.

Suppose that today a developer creates a new application, Application A, that needs to consolidate data from three data sources B, C, and D. Traditionally there are two ways of solving this problem:

1. Create a new data store that contains the data, pumped from the existing sources through a transformation process and stored in a form tailored for Application A.
2. Connect Application A directly to data sources B, C, and D so that it can integrate the appropriate data dynamically (an approach sometimes referred to as data federation because it does not require a new persistent data store).

The first approach, creating a new data store, results in some overhead in storage and management, and, more importantly, creates yet another redundant data store. With redundancy come opportunities for data inconsistency, which in turn must be taken into account in an overall enterprise data management scheme, creating more overhead. It’s not surprising that with the proliferation of many different redundant data stores throughout the enterprise, emphasis has gradually shifted from designing and creating new data stores to making better use of the data already in existing systems.

Reluctant to create and manage more data stores, many prefer the second option – connecting applications to multiple data sources so they can obtain data dynamically. The challenge is how and where to do it. Many application teams are still burying data access and integration logic in application code, where it is difficult to reuse and maintain. In an effort to make this process efficient and scalable, forward-thinking architects are evolving a third alternative – connecting the application to the various data sources through a layer of data services.

BRIDGING THE GAP

When data from multiple existing stores must be used together, developers need to bridge a number of different technical and semantic gaps to create required views of data. The need to bridge data gaps in a more straightforward, streamlined, and scalable way is becoming urgent due to the convergence of three factors:

- **More and more data.** Today's organizations need real-time information at all levels to enable the agile performance that will help them compete. Most organizations are managing more data than ever before. Growth in the number and complexity of data sources, inside and outside the organization, is having a profound effect on the requirements for new information systems. Projects increasingly require data to be accessed from multiple systems of record and require transformation and integration of that data. The sheer volume and wide distribution of data among diverse enterprise sources are increasing security concerns; access to data must be controlled and audited, which is harder when multiple copies abound. This concern plus the growing cost of data management are fostering renewed interest in reducing and controlling data redundancy.
- **The growing need for consolidated, consistent views of information.** The need for consolidated views of information across the organization is driving companies to grapple with the master data management problem. Not only must data be brought together, but it must also be consistent across systems, product lines, and divisions. A financial services firm cannot gain efficiencies from offering many different product lines to the same group of customers unless it can obtain a single view of the customer relationship. A high level of customer service is not possible unless the company representative has all relevant information available in real time. The compelling business requirement for consolidated views thus creates more projects involving data transformation and integration. Projects that focus on reference data management or customer data integration (CDI) lead inevitably to discussions about how to create a single view of the truth that can be shared by all relevant systems.
- **The drive toward interoperability and standards.** Interoperability goals, regulatory requirements, and the sheer complexity of today's information are forcing adoption of standard information vocabularies and schemas. These increasingly take the form of Web service/XML interfaces, while most existing data sources today are relational. The gap between data source schemas and new formats and vocabularies also contributes to an increase in the number of data transformation and integration projects.

The data access gap

Each system may access data in a different way. Relational databases require SQL and specific or general relational database drivers. Spreadsheets are read from a file system, as are certain other file-based data sources. Data feeds may come through publish-and-subscribe systems; some systems may use Web services and XML-based data, while others still might require data access through a URL. Diverse security protocols make the problem even more complex.

Application developers, who generally prefer to focus their attention on the logic behind business processes, are instead spending time resolving the logistical issues of data access. Embedding data access explicitly in applications leaves the code vulnerable to changes in system configuration. Because these logistical challenges have nothing to do with the business application itself, the specifics of data access should not need to be a concern for the application consuming the data. If instead data access is encapsulated in a data service layer that is responsible for providing a consistent interface to applications, a data source can be reconfigured without affecting the applications that use it.

The data definition gap: Data type, semantics, and vocabulary

Each database has typically been designed to serve the needs of one primary application. So there's a good chance that another application needing data from that database will need it in a different form. The logical design of the database may supply the right data element, for example Cust_ID, but it may be implemented differently from Cust_No in another system. Combining the two will require an analyst to resolve differences in length, data type, name, and other aspects of the definition. A project of this nature generates an opportunity to create and use a canonical "Customer Identifier" element. The challenge then is a simple matter of determining how to map elements in the existing data sources to that canonical definition.

In addition to simple differences of format and data type, developers must address semantic differences between systems. The question of who is included as a customer may be answered differently by the two systems. One system, for example, includes everyone who has directly purchased the company's products under the term Customer, while a reseller is called a Distributor. Another system might include both Customers and Distributors under Customer. While tools can assist with some parts of this process, completing the work requires an application of business-specific knowledge. Resolving differences requires careful attention from someone who understands the meaning embedded in the systems. As a result, this work can be time-consuming and expensive, requiring time from the most experienced members of the project team. To preserve the value of the investment, the work should be completed as efficiently as possible, and the resulting artifacts should be sufficiently documented (or self-documenting) that they can be reused where applicable. In contrast to this ideal approach, typically too much re-invention of the wheel occurs, with developers unable to know about or build upon work that has gone before.

The gap between different types of data structure

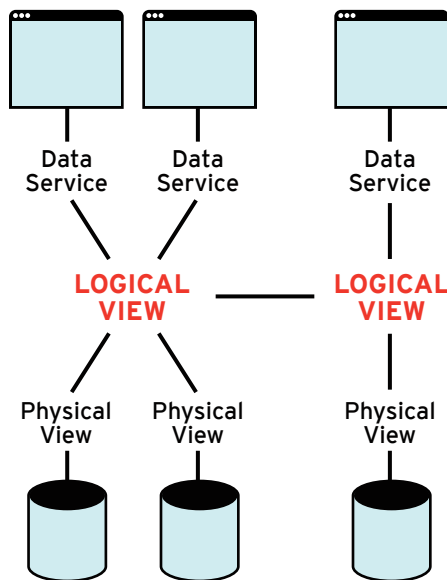
To further complicate matters, the basic data structures of sources (or a source and target) are not always as similar as they would be in, say, two relational databases. It is true that, with the widespread use of XML and Web services, XML-based data structures have become much more common as new application targets, and XML-based standards are driving standards for a number of different industries and system types in order to promote interoperability. But in reality, the bulk of existing databases are relational, with some using older network and hierarchical data structures. Today's projects find diverse legacy data sources being mapped to XML-based data structures, either homegrown corporate standards or industry, government, regulatory, or application-specific standards. Organizations need to be able to map from a range of data sources to even the most complex XML schema. The challenge here is to automate as much of the process as possible and facilitate the human intervention that must be done to create the right mappings.

The data standards gap: Data integration meets the data model

For decades, data architects have sought a canonical description of enterprise data—the enterprise data model—a common vocabulary and basic structure for enterprise information that would enable people and systems to work together more effectively. The introduction of computer-based data models and metadata repositories beginning in the late 1980s brought these efforts out of isolated architectural groups and into the rest of the enterprise, while data warehouse projects gave these data analysis efforts concrete expression. When properly scoped and supported, data standardization efforts improve an organization's understanding of the information it manages. The ability to share data models and database designs improves data consistency across diverse systems.

In many organizations, standard data models at enterprise, divisional, subject-area, or project levels are providing value to application projects. When a project requires data to be transformed or integrated, analysts have an opportunity to ensure that applicable data is brought in line with the standard. These standard views of data may be directly usable by an application, or they may need tailoring to meet specific application requirements. But deviation from the standard is a carefully controlled and conscious decision.

METAMATRIX DATA ABSTRACTION



WORKING AT A HIGHER LEVEL: THE NEW DATA TIER

Database application development best practices have generally incorporated the notion of a data tier—a layer of processing that concerns itself with data access and manipulation—that is isolated from the rest of the business logic. In some very simple applications, the data tier might be just the database, perhaps augmented by some validation or processing in triggers and stored procedures. In other applications, a data tier might be a set of application services that interact with the database, handling certain types of data validation, transaction logic, or exceptions not managed by the database itself. In still another approach, a data architect might enforce a policy of allowing interaction with the data only through a layer of stored procedures. In all cases, a data access layer sits on the border between data and the application.

Application designers utilizing service-oriented architectures must decide what the data access layer will be for a service-oriented architecture. If (as in the default case) the data tier is simply the diverse set of data sources used by the application, then substantial custom coding work must be done in each application to pull data together from multiple sources. Inevitably this logic, which executes data access and integration to create the data views required, will be redundant across some applications, and only rarely will logic be reused.

In a service-oriented application, it would still be possible to use this traditional approach and overload business services with data access and integration logic. But as the number of data sources grows, it becomes more obvious that this approach will not scale. If instead a data tier can be isolated, both outside the business application context and outside the context of a single data source, to address details of data access, transformation, and integration, then application developers can focus on creating business services that support a domain-specific business process. With well-designed data services, those business services will be able to interact with data at a higher level of abstraction.

The data logic – which handles data access, integration, semantic resolution, transformation, and restructuring to address the data views and structures needed by applications – is best encapsulated in artifacts called data services. Data services concern themselves entirely with data access, transformation, and integration, assembling the forms of data that can be manipulated by applications. A layer of data services can act as the data tier in a service-oriented architecture, hiding from the application the complexity of diverse and conflicting data sources and ensuring that the various data access and transformation activities are performed in a way that conforms to an appropriate logical view of enterprise data.

A data service approach allows organizations to simplify development projects and isolate difficult and challenging data integration and semantic mediation tasks so that the appropriate resources can be applied to each part of the solution.

2. ANATOMY OF A DATA SERVICE

The notion of a data service has grown in popularity both because of the growing acceptance of SOA – because project teams are realizing that custom-coding data access, transformation, and integration in business application code is costly and does not scale – and because organizations are reluctant to continue proliferating redundant data stores across the organization. What exactly must these data services do in order to address the various requirements outlined?

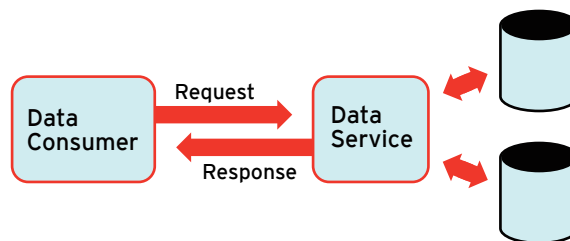
WHAT IS A DATA SERVICE?

Like other types of service, a data service must provide a service to an application through a well-defined request/response mechanism. A data service is essentially a specialized service that:

- Encapsulates data-oriented logic: logic that enables the orderly storage and retrieval of relevant enterprise information and is not concerned with a specific application of that information. By encapsulating all of the data logic necessary to provide the appropriate view of data (including data access, transformation, integration, aggregation, reformatting, restructuring, and so on), data services are able to hide substantial complexity from applications and reduce duplication of effort.

- Enables loose coupling between the application and the data stores. The point of contact between the application and the data store is the data service only. With this loose coupling, a change to the data store should not necessarily affect the application; a change to the “guts” of the data service should usually be able to occur without a change to the application interface presented by the service.
- Manages real-time connectivity to data stores, hiding the details of the diverse connections from applications.
- Offers a higher level of data abstraction (compared with interacting directly with data stores). A data service provides consolidated, logical views of data that represent either 1) a canonical view of a certain subset of enterprise data useful across applications or 2) a view tailored for a specific application. In either case, it supports standard CRUD operations against the view of data presented. (This need not mean supporting all operations for all views; a substantial amount of the access might be read-only.)
- Provides the appropriate views of data on demand, in real time, without creating that data in persistent form.

METAMATRIX DATA SERVICE



In order to meet these requirements, the data service must have the ability to:

- Connect to diverse types of existing data sources so as to be able to read and write data.
- Combine data from multiple sources in meaningful ways, which in turn requires the ability to specify explicitly how semantic and vocabulary differences are resolved.
- Transform existing data structures and formats into desired structures and formats.
- Coordinate transactions to execute the coarse-grained CRUD operations, which may include coordinating writes to multiple data sources.

APPLICATION ACCESS TO DATA

One major advantage of a data service approach is that it vastly simplifies an application’s ability to read and write data. The application accesses the data service, which in turn is responsible for accessing data in the various sources. However, the exact mechanism by which a data service is accessed depends on the requirements of the specific application. Two primary data interaction protocols are used in today’s application architectures:

- **A Web service interaction involving data interchange in XML format.** In this case the Web service's inputs and outputs are well defined so that any other Web service can interact with it using data in the defined format. For example, WS1 makes a request from WS2, supplying data in the form required as input by WS2, which in turn provides an agreed-upon response and data in its specified output format, all via XML.
- **A SQL statement submitted to a relational database management system.** The typical case here is a SQL SELECT statement issued by an application to a specific database, which assembles the required data and returns a relational result set to the application. In addition to direct application-to-database SQL queries, the SQL access modality may also be implemented as part of a Object-Relational persistence/access layer, such as Hibernate, to express the data in an object form.

A data service approach, likewise, is not limited to one type of architecture and can be used with either of these two interfaces or with a combination. (Other types of interfaces such as file reads are also possible.) Most organizations, even those wholeheartedly adopting service-oriented architectures, employ a combination of Web services and traditional architectures today, so any data service approach should take both types of systems into account to ensure applicability across the widest range of projects.

DATA INTEGRATION ON DEMAND

Data integration generally refers to two processes. The first is the process of specifying how data is to be integrated, where one might define, for example, how to combine Cust_No and Cust_ID into one field called Customer_Identifier. The second is the physical execution process of creating (and possibly persisting) an integrated view of data that can be used by an application.

When the "guts" of a data service are defined, the developer specifies how data transformation and integration will occur, providing a recipe for finding data from diverse sources and preparing it for consumption. The data service then finds the ingredients and performs the appropriate processing on them before serving the result to the consuming application. The process of pulling data together from two or more sources is sometimes referred to as "data federation" because it represents a just-in-time, loose connection to the sources rather than a hard integration of data into a new persistent store. While data services federate data, they also do much more; they perform a range of semantic resolution tasks that prepare data for consumption by the application.

REUSE OF DATA SERVICES

A chief benefit of a data service approach is the ability to reuse the work that has been done to define data access and integration. Because applications are loosely coupled with data sources through data services, and because the data service layer can be made up of multiple levels of transformation and integration, developers have a range of existing artifacts they can use in approaching a new data requirement for an application. These include representations of existing data structures in untransformed or transformed states, representations of data that conform to a logical data model, representations of the same data in relational or XML formats, and so on. Over time, as project teams complete their work for specific applications, they build up a library of self-documenting data services that can be discovered and used by others. The benefits of this reuse are many: greater data consistency, reduced testing burden, and a reduction in the amount of work and time needed to address new application requirements for data. Another way to think of this benefit is that the investment in one project may have substantial return down the road for other projects that use the same information.

MODELS AND LAYERS OF ABSTRACTION EASE INTEGRATION AND REUSE

Bridging the gaps between data structures, formats, and vocabularies, or between an existing database structure and the standard data model, requires work. But when this work is performed, often it is buried in application code or expressed implicitly in data mart structures. An important goal when using data services to solve this problem, though, is making the data service more visible to and maintainable by people who may not have created it. Several elements are needed to make this happen:

1. Explicit models of data for the sources and targets, based on implementation realities.
2. A clear, declarative, visual process for mapping sources to targets or intermediate structures.
3. Multiple layers of abstraction, to enable the incorporation of canonical data definitions as part of the integration process while allowing for the tailoring that might be needed for specific applications.

A data service is explicitly a bridge; it can be built from either side: from data sources, where certain views of data are defined and exposed to applications, or from the target, where sources of data are found to compose target data structures. In all these cases, a data model (or an XML schema) will be able to simplify and streamline the design process, either by specifying the structure required for the target or by specifying a logical layer to which source and target should be mapped. The more this process can be automated, by automatically importing models, the easier the developer's task will be and the more straightforward the process of conforming to the standard.

Some systems incorporate visual mapping directly from a source format to a target format. While this is better than having to write code, it does not protect the application from changes to the data source, and its reusability is limited. Further, it requires that all transformations and integrations occur in one step, simply putting the "miracle logic" in another place. A more natural and effective approach is to factor out each type of transformation or integration into its own explicit step with an associated definition artifact (a virtual data structure or model element). This way, definitions are straightforward to understand, and a change to the mapping in one step need not affect other steps of the transformation and integration process. The ability to express arbitrary levels of abstraction is valuable because it enables effective expression of the required transformations and enables reuse at many levels.

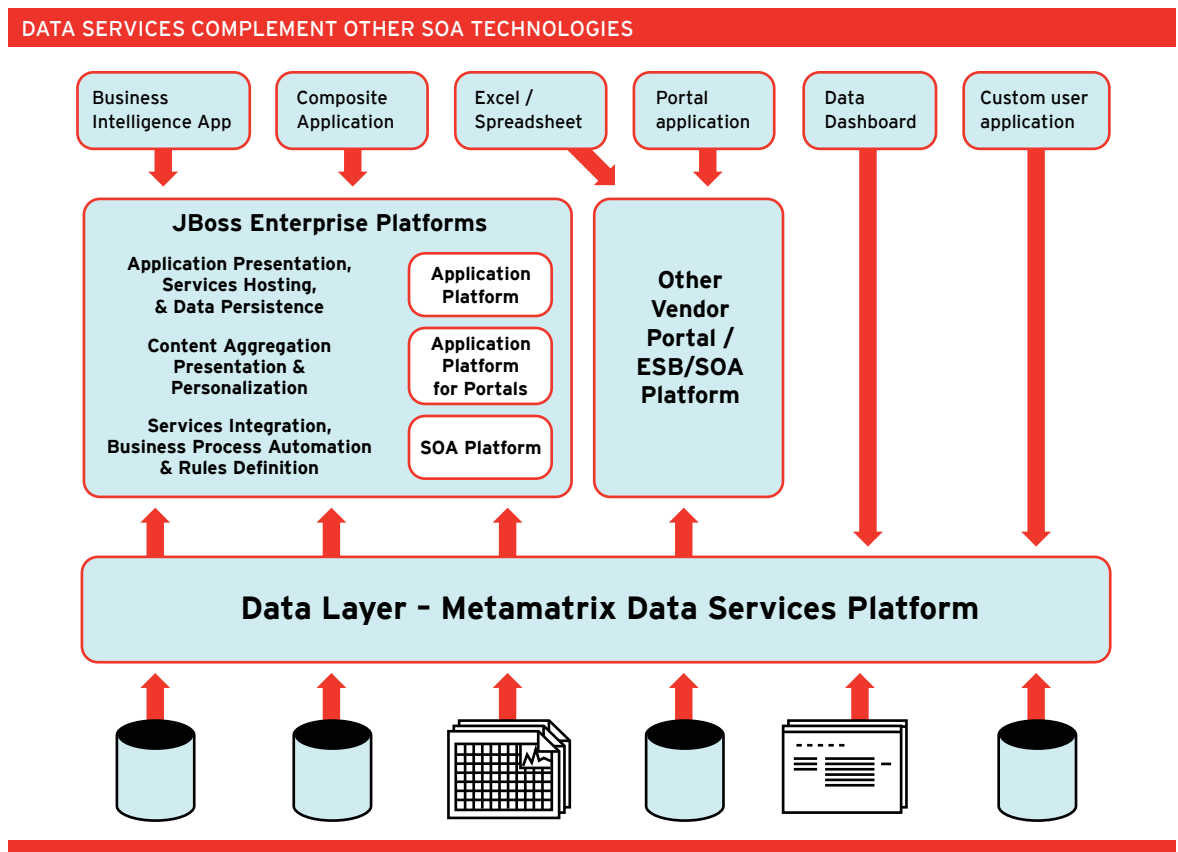
Multiple levels of abstraction also make it easy to incorporate explicitly a set of data structures that conform to the organization's accepted logical model. If data source definitions are mapped to a logical data model before being tailored for a specific consuming application, then other applications will be able to begin by taking advantage of the data services representing the canonical model. Project teams will have no need to return to each data source and build up everything from scratch. This means time savings, which can result in substantial project budget savings.

DATA SERVICES COMPLEMENT OTHER SOA TECHNOLOGIES

A data service layer is a foundation layer for a service-oriented architecture, providing services that comprise a data tier so that business analysts and developers can define business services that execute only business-process-specific logic. A data tier eases the creation of business-specific services because they can interact with data at a higher level of abstraction.

As organizations embrace service-oriented architectures, many are adopting technologies such as UDDI registries and enterprise service buses (ESBs). A layer of data services complements these technologies by factoring data-oriented logic out of the business services. If desired, data services can be published in a UDDI registry, making them available to a wider constituency.

ESB technology can manage connectivity for application transactions, orchestrating and brokering interaction among services and mediating between different system protocols. Some ESBs incorporate the ability to transform data as it moves between systems, but this is not their exclusive focus. Often it is a limited capability that involves custom coding in Java or XSLT and does not offer the productivity and reusability provided by a data service implementation. While ESBs are not sufficient to implement scalable and reusable data services, an ESB complemented with a data service management solution such as MetaMatrix Enterprise Data Services Platform is an ideal support for an organization's migration to service-oriented architecture. The data service layer can define in a comprehensive and maintainable way the various steps required to mediate between forms of data and can encapsulate the protocols for accessing the various data sources.



CONTROLLING DATA SERVICES: THE DATA SERVICE LIFECYCLE

A data service layer can make visible and concrete an organization's use of data from a wide range of sources. As a result, it can contribute to an organization's information management and stewardship initiatives. For this to occur, the data services themselves must be managed in a way that takes into account the entire lifecycle of the data service including requirements, design and testing, deployment, maintenance, modification, discovery and reuse, and retirement. In addition, just as a database system is managed, so must the day-to-day execution of the data service be managed, including provisioning and access control, security of data access, performance in accordance with service-level agreements, and availability. Tools that facilitate these activities streamline data service management and enable efficient use of data across the organization.

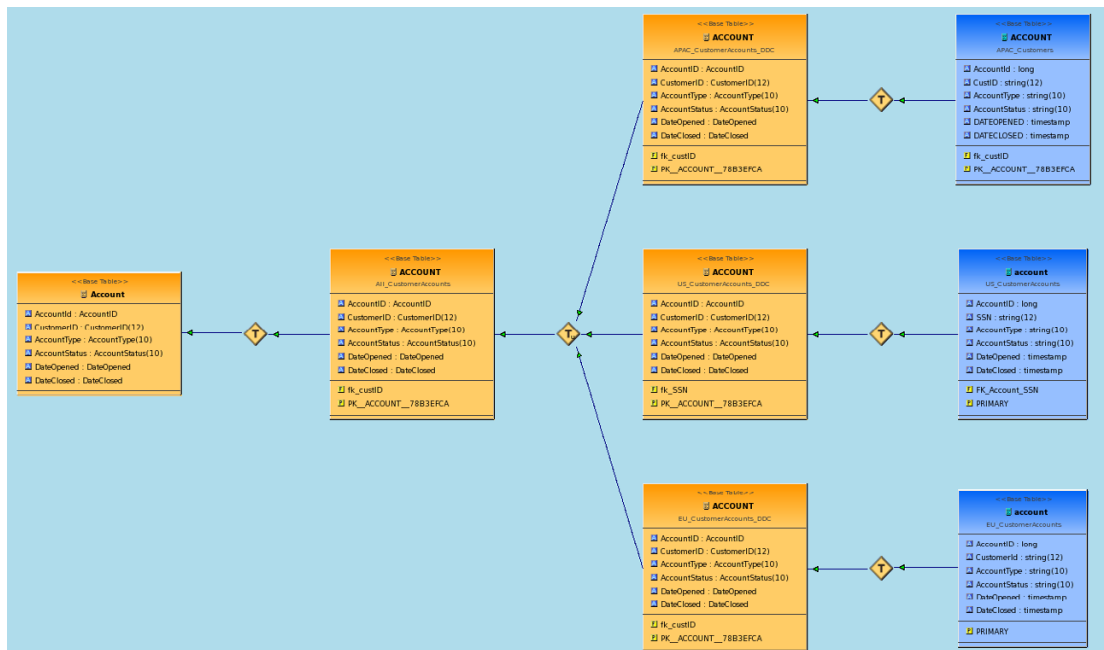
3. WHAT IS A DATA SERVICE MANAGEMENT SYSTEM?

Given the need for data services, the next logical question to ask is how an organization can get started creating, deploying, and managing them. What are the requirements for data service lifecycle management?

CREATING DATA SERVICES: THE DESIGN ENVIRONMENT

The process of creating a data service can consist of declarative specification and need not require procedural coding. It should be a straightforward mapping of elements from physical data sources to logical data structures and a mapping of logical data structures to the specific views of data needed by applications. As mentioned earlier, this process should allow for multiple levels of abstraction so that the various transformation and integration steps can be defined in an explicit modular way (not overloading any one operation), and so that the structures thus defined can support maximal reuse.

METAMATRIX DESIGNER TOOL (PARTIAL SCREENSHOT)



Given these requirements, it is necessarily to be able to import metadata for physical data sources and also any relevant logical models. As the developer maps the elements, it should be possible to generate the appropriate transformation and integration logic automatically. For example, in relational terms the ideal visual paradigm is a graphical data model with the ability to show not only the source and target models but also data dependencies. For Web services, we need a way to represent XML schemas and map them to the data source structures, which are typically relational data structures.

Models help designers see the relationships among the various data sources, and dependency diagrams and other features that show the lineage of each data service and data element can be important aids to impact analysis when new requirements arise or when a data source must be modified or retired. Finally, an explicit, visual map of data use across the organization can be a valuable part of managing and enforcing policies with respect to data confidentiality and other forms of compliance related to information use.

MANAGING METADATA: FROM DATA SOURCE DESCRIPTIONS TO STANDARD DATA MODELS

A wide range of metadata is valuable in the process of creating and managing data services. A data service management system should be able to import and manage metadata including that from existing data sources and the standard data models used by the organization. While the imported data source definitions and logical models will be used directly in the design process, they can also provide input to broader data analysis projects in which existing systems are evaluated for redundancy, relevance, quality, or potential retirement. In this context, the ability to store and report on metadata and models is an important means to understanding how data is interrelated across systems. In addition to data source schemas and logical models, any specifications, notes, or other documents relevant to data management should be able to be stored in a metadata repository to be available to designers for later iterations or other projects. A multi-user repository facilitates sharing, discovery, and reuse of data services and related information.

THE RUNTIME ENVIRONMENT: CONTROL AND EFFICIENCY

In addition to the useful definition aspects of the data service and their relationship to existing data sources and metadata, a data service management system must also provide a means by which data services are deployed in an execution environment. Remember, data services are useful only when applications can use them to interact with data. This can be done in any of several ways, but the following factors must be taken into account:

- **The ability to connect to multiple, heterogeneous data sources.** Enterprise data sources are diverse, including multiple RDBMS systems, non-relational databases, OLAP systems, third-party data feeds, other structured data such as spreadsheets, packaged application data (which may be stored in diverse ways and available through multiple interfaces), and semi-structured data in content management systems. All of these can contribute to views of information needed by decision makers at all levels.
- **The ability to optimize performance.** The efficiency of data transformation and delivery depends on various factors. Performance optimization may include built-in query optimization, the ability to cache data, the ability to stage data in cases where it does not change frequently, and the ability to balance execution load among multiple servers.

- **Management of data security.** Any system that governs access to enterprise data must take into account the authentication of users, authorization to execute specific operations on specific data, and protection of data privacy and integrity.
- **Facilitation of relevant system management tasks.** Facilitation of system management tasks such as failover and availability management, load balancing, and connection pooling, as well as detailed logging.
- **Opportunities for auditing.** Given the scope of a data service management system, there should be potential to audit executions of data services, which in turn provides the opportunity to audit which data has been accessed by which systems and users. This information can support a range of business processes including resource allocation and regulatory compliance.

CONCLUSION

Migration to service-oriented architecture is catalyzing new thinking about how data access and integration occurs and how data flows through an application. The standard SOA stack does not, in itself, solve the data access and integration challenges that are growing with the number of enterprise data sources and the breadth of application challenges. As organizations enthusiastically adopt SOA, there is growing danger that the data access and integration portion of the application's stack will be embedded in the business application code, resulting in brittle and poorly-performing systems. If this happens, a new SOA infrastructure will fail to achieve its objectives.

While service-oriented architecture decouples application components from one another, organizations still need a way to decouple data access and integration from applications without creating either brittle code or additional costly persistent data stores. A data service approach hides the complexity of diverse data sources from the applications, enables the isolation of data-oriented logic in a data layer that is independent of applications, and renders the costly work of data transformation and integration much more valuable to the organization by making it more visible, reusable, and maintainable.

The set of capabilities needed to create, deploy, execute, and manage data services – also known as a data service management system – is possible with today's technology and can provide substantial benefit to organizations as they gradually migrate toward service-oriented architectures. A data service layer will be an important foundation layer for any service-oriented architecture, providing the bridge between legacy relational data and the new architectural paradigms. MetaMatrix Enterprise Data Services Platform is a data service management system that is currently in production use in large corporations and government agencies and is helping those organizations make the best use of enterprise data.

JBoss SALES AND INQUIRIES NORTH AMERICA

1-888-REDHAT1
www.jboss.com