Enterprise Data Services in SOA using ODI Suite

An Oracle White Paper
January 2008

ORACLE

Enterprise Data Services in a Service Oriented Architecture using Oracle Data Integration Suite

OVERVIEW

With all the focus on the Service Oriented Architecture (SOA) vision in our industry, people sometimes forget that SOA is mostly about plumbing, and that a substantial portion of the top-line business value of SOA derives directly from the data it pushes around.

Today, the data universe of Business Intelligence, Data Warehousing and Master Data Management still lives largely outside the scope of the emerging SOA vision. But an Enterprise view of Data Services may well bring these data and SOA worlds together.

When properly architected and executed, Data Services can provide the missing link that unifies conventional data systems with the emerging SOA paradigm. By offering a decoupled data façade and an easily virtualized API, Data Services can give SOA the opportunity to establish system control – *without always forcing enterprise data through an inefficient XML layer.*

This whitepaper will demonstrate why Data Services are a foundation requirement for enterprise SOA deployments by describing how just how valuable, and complex, business data is for the businesses that rely on it. Further, a solution for enterprise-class Data Services will be presented using Oracle Data Integration Suite, which is the market's only SOA-based Data Service platform that combines the full power of SOA with conventional best-of-breed data management tools.

PART 1: THE PRIMACY OF DATA

It's always been about the data. Decades of punditry about EAI, ETL, MDM and SOA still lead us to the same conclusions – data matters. If content is king in the consumer Web, then data is king in Enterprise Software.

Sometimes the Enterprise Software sector loses sight of that simple reality. In the past fifteen years, with the rise of Java, the hype surrounding EII, EAI and SOA, and the rapid rise of XML, and quietly, the billions spent in ETL projects – it's all too easy to forget why we build and buy all that infrastructure. We do it for the data.

Without the data, there would be no need for process orchestration. There wouldn't be any purpose to all those SOAP envelopes, all those service bus' wouldn't have anything to publish and application servers wouldn't serve anything. Data is king.

But data presents huge, looming, non-trivial problems. First, businesses have figured out how to collect more of it but still can't effectively understand it all. Second, with more of it around, the infrastructure and tooling is bursting at the seams to manage it effectively. Third, the approaches used to define it in small architectures simply won't scale out to large business sized problems. Finally, enterprise architects too often get caught up in the buzz of new technology and forget that thirty-years of hard-fought lessons about data management still apply.

There has been more data created since 2000 than in all of human history preceding then.

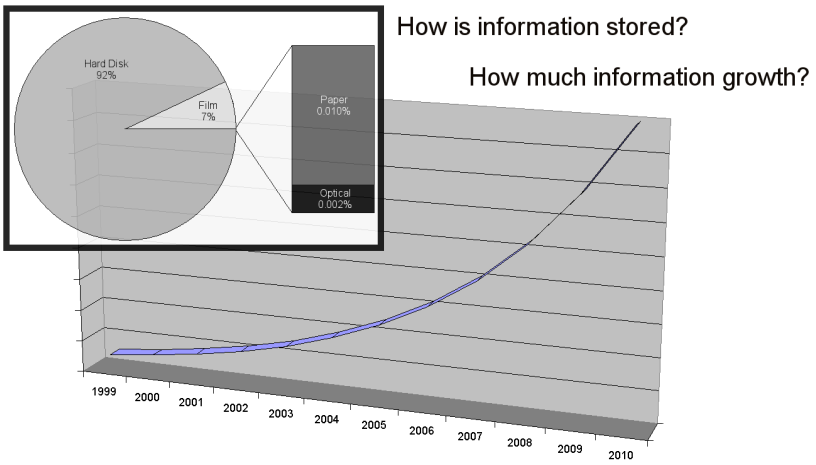


Figure 1: Information Explosion (Adaptive Information, Wiley & Sons 2004)

For our businesses and governments, the rise of sensors means that we can monitor anything in realtime: from where your shipments are, to the temperature of your factory, or your very own heart rate. All that data ends up somewhere. It is stored indefinitely, used for realtime dashboards, historical analytics, or put somewhere just in case. But we can now collect more data, at faster rates, than we can successfully interpret. And the rate of data collections, the use of sensors like RFID and other monitors, is growing exponentially. In other words, the data problem is getting worse, not better.

But enterprise infrastructure is surprisingly unchanged since the early 1990's. Back then, Message-Queues (MQ), Transaction Processing Systems (TPS), and ETL tools were really the backbone of enterprise software. Guess what? They still are. Despite the growing adoption rates of BPM, SOA, ESB, and EII – the MQ, TPS, and ETL backbones are still there.

The strain of all that new data and the demand for mature tooling has paradoxically made the existing, proven software infrastructure look pretty attractive. Many new systems will try to put all the data in XML, or perhaps try to use Java Entity Beans as the data management tier. While these are acceptable for smaller applications or for specific use cases, neither of these approaches scales to the mult-terabyte sized problem that is typical of a Global 2000 business. Thus, a knowledgeable architect will revert to the proven patterns of RDBMS as the

backbone of a data architecture using MQ, TPS, and ETL interfaces as the pipes for pushing all that data around.

But the buzz of SOA is deadening. Why not SOA for data-centric architectures?

PART 1: A ROLE FOR SOA IN DATA-CENTRIC ARCHITECTURES?

When the Service-Oriented Architecture craze started somewhere back in 2001, we thought it was magic. Remember the promises of dynamic discovery? Human readable messaging? Simple XML data objects? But soon enough, the problems started: competing vendor specs, security loopholes, performance problems…and so on.

Here in 2008, the good news is that SOA has finally matured into an Enterprise class infrastructure. Far from the original hope of solving all integration problems, the main tooling for SOA (Enterprise Service Bus and Business Process Engine) is almost at a level to realistically supplant the long-held dominance of MQ and TPS systems. Both the reliability and performance of basic SOA is strong enough for all but the most demanding problems. However, SOA is still not best for ETL and data integration.

Data integration use cases span from the simple to the impossible. On the simple side of things, transforming some small amount of data and putting it somewhere, a regular SOA with XSLT based transformation services running on a Service Bus can usually handle things. It helps if the data formats are supposed to be XML because converting data to XML just to transform it into some other non-XML format is non-optimal, SOA can work just fine for those simple XML-centric data integration cases.

But the average data integration use case is beyond SOA's core strengths. An average use case might involve loading a few gigabytes of data from one database to another, applying transformations to change the shape of the data from third-normal-form (3NF) to a multi-dimensional (Star) model. This average use case is to support line-of-business demands like: Reporting, Business Intelligence, Performance Management, Financial Planning and other analytic capabilities. SOA is wildly inappropriate for this average use case because of poor bulk data transformation performance and inefficiency.

Nearly all SOA frameworks operate in a Java container, which is a substantial disadvantage when gigabytes of data need to be consumed into a Java Virtual Machine. Likewise, the SOA paradigm for working with data is XML – nearly all SOA frameworks require the data to be converted to XML for it to be orchestrated and transformed. But a single gigabyte of data will multiply to five or ten gigabytes of XML data merely because of the additional tags, schema and angle brackets. (see Figure 2)

After all, XML is still best as a document and message format. For a while, the SOA buzz fooled everyone into thinking that XML is a data language, but it's not. A simplification of SGML, XML was only ever intended to provide a well-structured, standard way of marking up documents and messages. The core model of XML and XSD is actually called Infoset, a tree-like structure to define what kinds of XML items are allowable. But XML Infoset is not supposed to be a data model in the same way that relational and graph data models are. This is one

reason why pure XML databases are exceedingly rare, and still far less technically preferable for general data management.

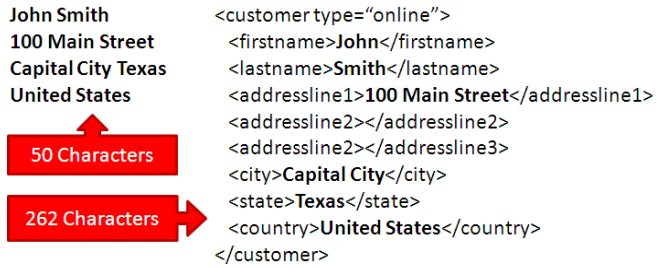


Figure 2: Added Data Bulk With XML Markup

In fact, neither of the early definitions of SOA Data Services are truly scalable to Enterprise sized problems. The Java definition, largely heralded from a host of standards like JDO, SDO, DAS, DTO, etc really is about (a) trying to define patterns for interfacing Java with relational data and (b) standardizing the APIs for moving those objects or components around between applications and Java containers. But few enterprise solutions federate Java objects using containers as the primary means of enterprise data integration or federation.

The other early SOA Data Services definition is an XML oriented view of Data Services dependent upon XSD-based Canonical Models for data exchanges. This approach advocates the use of XSLT based mappings to canonical message formats and sometimes the use of XQuery and XPath (or SQL) to federate queries across unions of data from various sources. But as aforementioned, XML is a poor data model, inefficient, and the federated query approach only works well with highly optimized caching.

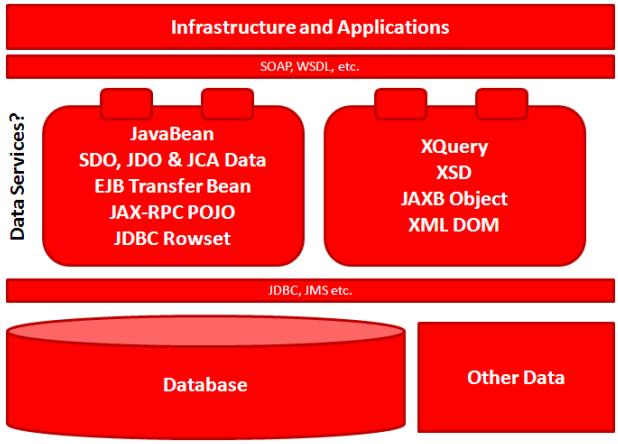


Figure 3: A Conventional Java/XML View of Data Services

The simple and unfortunate reality is that enterprise data requirements are hard, and the dreams of SOA only solution for all enterprise data are likely to remain dreams only.

Enterprise data requirements are fundamentally too complex and too closely driven by the high-volume, mutli-dimensional nature of business intelligence systems to entirely be serviced from a messaging layer alone. Further, valuable

patterns and lessons about enterprise data services actually precede the invention of SOA, and can exist in harmony or completely independently from the SOA infrastructure itself.

So, given this decoupling of data services from SOA, what does SOA have to do with them?

PART 2: WHAT ARE SOA DATA SERVICES?

Despite the inability of SOA to crack into that data management's foundation, there is mounting evidence that suggests that harmonizing enterprise data services with newly deployed SOA infrastructures may yet generate substantial new benefits. These benefits derive not from the replacement of traditional data management systems, but rather the use of SOA as a control point for them. Thus, SOA Data Services are not services operating solely on XML, SOA Data Services are enterprise data management end-points that expose highly optimized engines for working on all types of data.

Data services themselves need not employ SOA to be rightfully be called a service. In fact, all the key data services attributes, including contract-based development, data encapsulation, and the use of declarative APIs pre-date SOA by quite some amount of time.

Depending on how you may personally define data services, it is quite easy to claim that data services have been an institutionalized part of software infrastructure since the rise of EDI (Electronic Data Interchange) services between financial institutions in the 1960's. Later, key data service patterns became commonplace in the 1980's with the rise of Object-Oriented design principles. Most recently, data services in Java actually pre-date the notion of SOA data services by a few years.

Technically, a data service should exhibit several of the following attributes:

- **Contract-based bindings** – for design-by-contract, WSDL/SCA for example
- **Data encapsulation** – access to data via APIs only, indirectly
- **Declarative API** – some type of query-able API in addition to regular bindings
- **Decoupled binding metadata** – API descriptors are themselves part of a model
- **Decoupled data schema metadata** – data schema is separate from API

But perhaps the notion of a data service is more about an ideal. Data services may be about the ideal that there can be a single, shared control point for all important business data. Data services should expose control points for data that are easy to access, publish, and discover. So, in a most basic way, the data service may simply be a stereotype – a label, or tag – used to mark a particular software component's purpose for existing.

Unfortunately, the power of marketing has ingrained some popular notions of data services that are both too narrow and too shallow for real Enterprise work. First, there is the myopia of data services as only providing Enterprise Information Integration (EII) style federated queries. Several small vendors have staked a claim that EII by itself supplies data services as federated queries and XQuery or SQL-based data views. But these cache-based delivery mechanisms equate to a data hub in practice – and the hub-and-spoke data hub is a very old pattern indeed. In fact,

business requirements for true (non-cache-based) query federation are exceedingly rare in actual practice, and only a very small aspect of real world data services.

The other popular notion sometimes sold alongside the EII vision is the idea of Canonical XML schema for data services. From the previous section, it should be clear that while valuable, XML-based data models are no substitute for real data models, and should only be thought of as a temporary manifestation of data during certain kinds of transactions.

Taken as a whole and with an eye towards Enterprise sized problems, data services can encompass several different data delivery styles. Too many SOA pundits assume that XML is the only desirable data delivery format, but for a data solution to be truly useful for the Enterprise, it must support several different delivery styles. Data delivery is simply the way in which a software client can engage a service for data.

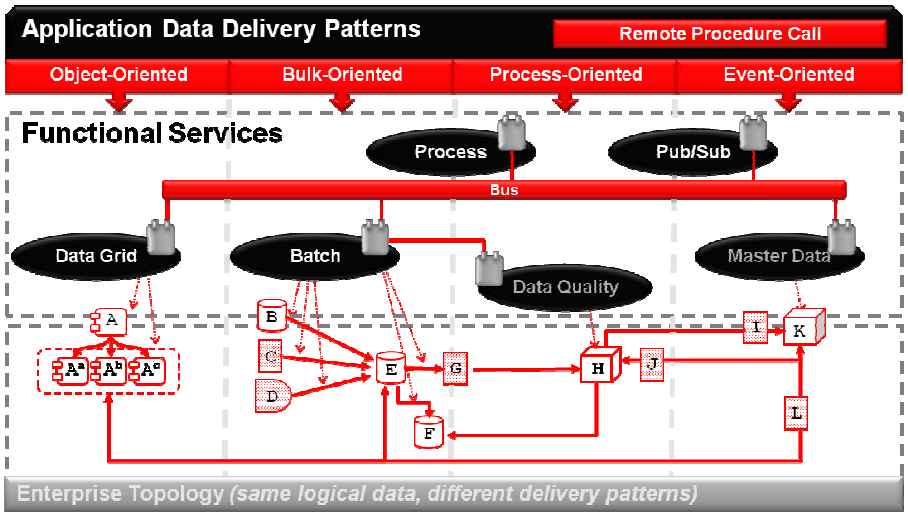


Figure 4: Data Delivery Patterns, Functional Services and Topology

Here are some typical data delivery patterns for working with data:

- **RPC-style Delivery** (remote invocation) – the basis for most delivery styles, the basic pattern simply suggests that a call made to a remote process should return some data, in some cases the call itself may contain a declarative query like SQL.
- **Event-based Delivery** (publish/subscribe) – this can be a traditional SOA Enterprise Service Bus type of delivery, or potentially the lower-level Change Data Capture type of publish and subscribe pattern.
- **Process-based Delivery** (transactions via BPEL) – this delivery style may involve long-lived and multi-step transactions with relatively sophisticated logic such as transaction compensation, call-backs, and hooks to common business rule libraries.
- **Object Delivery** (via marshaled objects) – this is the regular way a software application works with data objects, as marshaled Java, C++, or C objects held in memory. Modern JVM, J2EE, and .NET caches can allow for shared object pools that span 100's of machines and terabytes of RAM.

- **Bulk-style Delivery** (low level) – typically accessed and commanded via a regular API, the actual data work occurs at a very low level, sometimes pushing direct to DBMS via bulk loaders, native protocols, and/or JDBC, and may also include watching transactions from DBMS transaction logs.

Taken together, these basic patterns represent the different ways that software applications typically interact with data services. Sometimes they are as simple as sending a SQL query to a Listener service via an RPC style call on top of some protocol like JDBC. Other times they can be much more complex like triggering a low-level process that unloads data from several sources, merges and joins the data in sets, and finally loads a business intelligence OLAP Cube. But in all cases, the role of the data service is to help simplify the steps an application needs to do for working with data.

Client software applications that require data might employ any of the data delivery styles we have mentioned thus far, but what exactly would they be using them for?

Functionally speaking, there are several classes of Enterprise data services that have historically provided features to the enterprise which are starting to appear as foundation data services in medium and large service-oriented architectures.

PART 3: FUNCTIONAL DATA SERVICES FOR ENTERPRISE SOA

On one hand, data services are merely a stereotype that a particular service should be the common point of reference for a particular data item. On the other hand, data services should conform to certain patterns and delivery styles to genuinely fulfill an Enterprise class Service Level Agreement (SLA) on the distribution and delivery of data.

These SLA's can typically be drawn around some type of functional capability, the purpose of the service itself. And these functional capabilities can be classified into various categories that represent some classical function points for data services. But in practice, the actual data service may be more fine-grained than the category. For instance, rather than having an Enterprise service for Master Data Management, an Enterprise might deploy a Customer MDM Data Service that acts as a common reference point, with managed SLA's, for the distribution and delivery of Customer data. Likewise, rather than having a Data Access service, an Enterprise might create a much more fine-grained Tax Code Data Access Service that's published as part of an organizational SOA rollout.

Some typical functional data service patterns might include the following:

Master Data Services – these are data services that focus on the full lifecycle management of high-value business data within an organization. Master Data Management (MDM) may involve the management of Records and Instances of data, or the attribution of Models and Taxonomy for the classification of data. A typical MDM solution will have strong governance controls for the management of changing data values and data structures, often enforcing several levels of workflow and approvals for the modification of trusted business data.

- Motivation: The complexity of enterprise data environments makes it difficult to find or assemble trusted, high quality business data, hierarchies, and data policies.

- Usage: May be used as a reference service during realtime SOA transactions or bulk data movement, typically applied with transformations.
- Variations: Master Data Hub, Master Data Cache, Master Data Applications (Customer Data Integration, Product Information Management, Financial Data Hub…etc)
- References: Oracle MDM, IBM, SAP, Kalido, Siperian, etc.
- Caveats: Conventional MDM providers are still transitioning to SOA architectures and few are beyond the most basic step of exposing MDM services via SOAP and WSDL APIs.

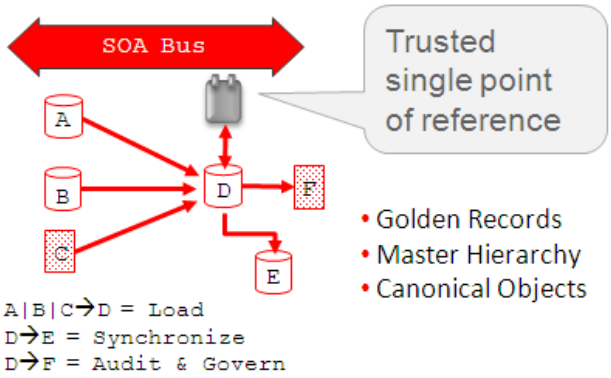


Figure 5: Master Data in SOA Pattern

Batch Data Services – these are data services that provide bulk data movement and transformation services. Typically, a batch data service would expose a Web Service API for SOA-based applications to invoke these bulk data/ETL style jobs from the SOA layer. Several known implementations incorporate these batch data services as sub-processes to a transactional BPEL or ESB process – so that the point of control for the ETL jobs is at the SOA layer, but the delegation of efficient bulk data handling occurs at the most appropriate architecture tier.

- Motivation: ERP, Data Warehouses, Business Intelligence and Performance Management Applications require bulk data movement.
- Usage: May be used for Replication, Bulk Refresh, Data Migration, Large File Transformations, and Changed Data Capture
- Variations: ETL (requires dedicated hardware), E-LT (low cost, high performance, runs on SOA layer), Low-Latency Logminer CDC
- References: See Oracle Data Integrator, DataStage, Informatica
- Caveats: Be cautious about using ETL from SOA, it could create redundant hardware infrastructure and duplicate SOA logic – look for native E-LT implementations that can actually run on the SOA tier.

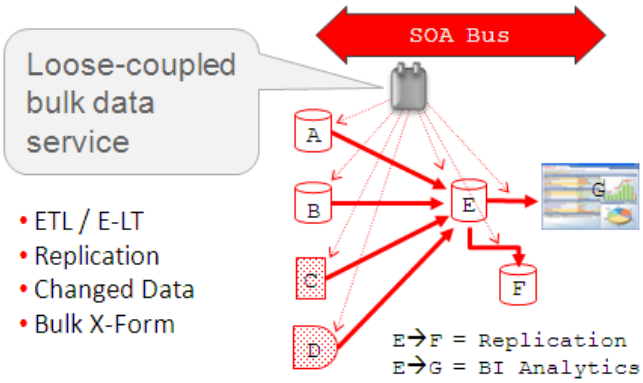


Figure 6: ETL/Bulk Data in SOA Pattern

Data Access Services – these are data services that provide direct access, through a managed (synthetic or physical) view, to the resident location of the data. Data access services may be as simple as a Web Service for fetching data from database. Data access services may also be as complex as issuing queries to synthetic data views and having the service federate data source queries in realtime with aggregated data result sets.

- Motivation: Present a simplified query interface to consuming applications. Usually by combining a shared abstraction (Canonical Model) with instance virtualization (Data Mashup).
- Usage: Traditionally exposed as part of a J2EE/.NET server layer, in a SOAP environment the extra step of conversion to XML (usually Canonical) is added to the process
- Variations: Query Federation, Data Hub & Spoke (Object|SQL|XQuery), Object-Relational Mapping (ORM vis J2EE/Toplink etc)
- References: See Oracle Application Server, Oracle Data Integrator, BEA AquaLogic, Ipedo, Composite Software, Meta Matrix, IBM DB2ii
- Caveats: this category in particular has many technical variations which should be carefully weighed in a cost/performance tradeoff.

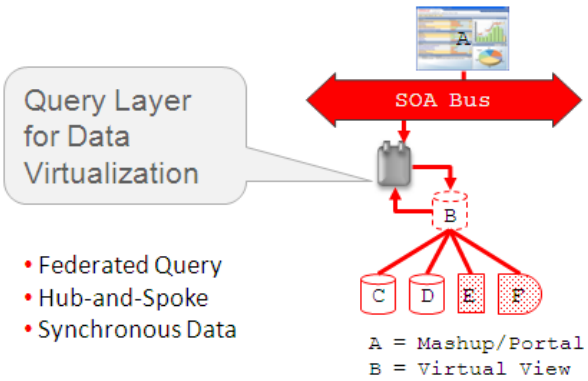


Figure 7: Data Access via SOA Pattern

Data Grid Services – these are data services consumed directly by the application tier. Typically imported as part of the classpath for an application, the data grid services appear to the application as a native object pool. In Java, the data grid might look like POJO's (Plain Old Java Object's), but each object may be

marshaled from a different JVM hosted in a different machine's RAM. Data grid services provide exceptionally fast caching for data access.

- Motivation: Very fast, in-memory data frequently needs to span multiple applications, due to geographical factors, or to overcome the limitations of RAM capacity on a single host.
- Usage: Typically deployed for federated state-full persistence at business object tier, in order to predictably "scale-out" applications while maintaining exceptionally fast performance
- Variations: Java, .Net, C++ variations. Peer-to-Peer and Hierarchical Clusters, UDP/TCP…
- References: See Oracle Coherence, Gigaspaces, Gemstone
- Caveats: Data grid services are not a replacement for persistence, they are typically used in combination with relational databases for storing the data and for maintaining accurate lifecycle controls on the data

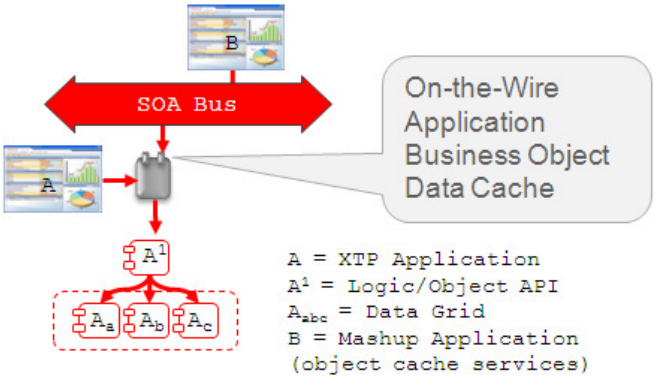
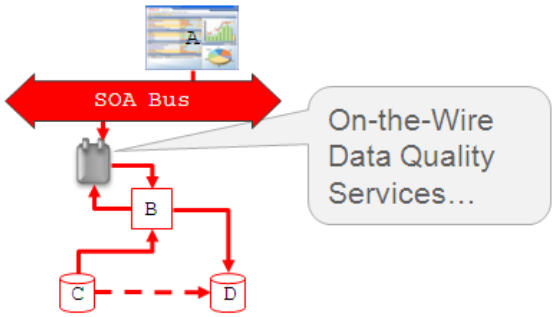


Figure 8: Near Cache Data Grid Pattern (w/SOA end-point)

Data Quality Services – these data services use algorithms and pre-defined business rules to clean up, reformat, and de-duplicate messy business data. Typically these services are used inline with other data services (for example: using a data quality service inline with bulk data/ETL services) or statically on a data source (for example: cleaning up a legacy database). But more recent applications show that hosting a data quality service within a SOA can provide much needed cleansing and standardization services to SOA messages and data.

- Motivation: Automatically improve the quality of bad data so that legacy data resources become more valuable and usable.
- Usage: Traditionally applied in batches to clean up Data Warehouses and BI repositories, the usage is now shifting to realtime and preventative use case, to cleanse the data before it's a problem
- Variations: Declarative/Rule-Driven, Probabilistic or Statistical Learning-based, Domain-Specific and Content-Oriented Data Quality
- References: See Oracle Data Integrator, Oracle Warehouse Builder, Informatica etc.
- Caveats: Data quality services are not magic silver bullets, for the most part, you get out of them what you put in. In other words, expect to put time in to these services for optimization and tuning of the business rules.

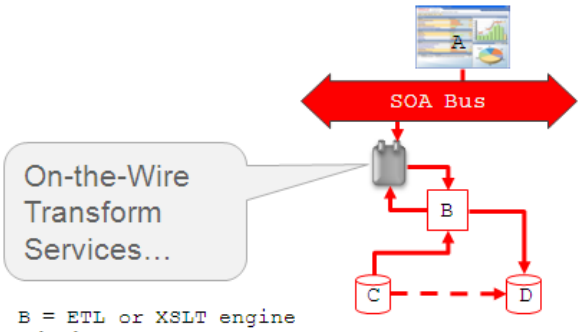


A→B→A = Message Cleansing | Matching
C→D = Bulk Cleansing | Matching

Figure 9: Data Quality via SOA Pattern

Data Transformation Services – these are the classic data services, simply waiting to take one format in, and provide another format out. Historically, in a SOA-only world, these would have been deployed as XSLT libraries, where a consuming application service would send in some data, choose a corresponding XSLT, and receive the data in a new format. In a more mature SOA, transformation services may also include ETL like services that specialize in efficient transformation of bulk data (10-100's of MB) payloads.

- Motivation: Present a reusable service for WSDL-driven data transformation – generally supporting multiple types of transformation (such as: RDB-to-RDB, XML-to-RDB, XML-to-XML, Flat-to-XML, Flat-to-RDB…)
- Usage: Best practice for enterprise systems with centrally maintained service families.
- Variations: XSLT Factory, ETL Engine, Canonical Mediator Service (either XSLT or ETL driven)
- References: See Oracle Application Server, Oracle Data Integrator, Contivo Builder, Tibco Data Exchange, BEA Aqua Logic, etc.
- Caveats: there is rarely a one-size-fits-all transformation service – a mature SOA may have several transformation data services which specialize in different formats and which provide more optimized SLAs.



B = ETL or XSLT engine
A→B→A = Message/Doc Transformation
C→D = Bulk Data Transformation

Figure 10: Transformation Data Service Pattern

Data Event Services – these are data services that monitor, correlate, and propagate events that happen on business data. Data events may occur at the middleware messaging, data integration, and database tiers of the infrastructure. In

a mature SOA implementation, data events can be subscribed to regardless of whether the events are occurring in the database, middleware or elsewhere.

- Motivation: Every part of the data environment must be capable of trapping actions, checking policies and taking action based on those policies
- Usage: Typically deployed on a given technology tier (eg: within Java, or on a Bus, or in a DB), but should be capable of calling to other event systems (eg: Java event triggers SOA triggers DB)
- Variations: EDA (Event Driven Architecture), CDC (Change Data Capture), CEP (Complex Event Processor), Java Event Listeners...
- References: See Oracle Application Server, Oracle Data Integrator, Oracle Event Driven Architecture Suite, Tibco CEP. BEA Event Server
- Caveats: data event services are a powerful but new technical capability – as of yet, there are no common policy definition standards, or standard frameworks for event detection at any given software tier.

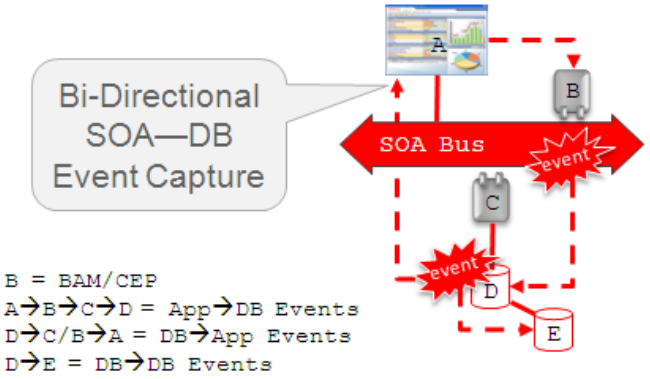


Figure 11: Data Event Service Pattern

By no means are these the only functional categories for data services, and actual data service instances will have further specialization beyond what is described here. The collection of aforementioned data service profiles are meant to give guidance to an architect when planning a multi-year SOA rollout strategy that might include a range of different data services for different kinds of use cases.

But given all the types of data services and complexity for rolling them out, where should the typical SOA start?

PART 4: PRACTICAL ADVICE FOR SIMPLE DATA SERVICES

In the aforementioned sections we have primarily examined a vision. The ideal state of Data Services within a Service-Oriented Architecture is a nice thought but leaves many wanting more on the practical side of implementing Data Services today.

Here are four quick tips for starting on Data Services today:

- Find the low-hanging fruit for your project
- Don't assume everything has to be XML data
- Be aware of J2EE and SOA-based Data Service tradeoffs

- Always remember, hybrid architectures are a fact of life
  (aka: don't be afraid of the two-tier architecture!)

First, find the low-hanging fruit on your project. The easiest ideas may be the "boring but important" ones. For example, find the most repetitively used data functions of a composite application, and manage those as part of a unified Data Service. These repetitively used data functions might be business-focused or technically-oriented but they should always be very general. For example:

- Business Data Service Examples
  - GetCustomer.wsdl (context, filters…)
  - UpdateBusinessEntity.wsdl (entityName, newEntity)
  - CalculateSalesTax.wsdl (item, geography, promotions…)

- Technical Data Service Examples
  - GetChangedData.wsdl (entityName, filters…)
  - AddAttribute.wsdl (canonicalFormat, newAttribute)
  - InvokeETLJob.wsdl (packageName)

These generic types of services may be boring, but will assuredly be some of the most widely used, and widely overloaded, within an Enterprise SOA. A big part of the Data Service challenge is to provide a controlled, but flexible infrastructure that will allow different organizations to build, modify and publish their own services within a shared framework.

Low-hanging fruit may also be found by looking at places to optimize Data Services. Instead of arbitrarily assuming that every piece of data must be converted to XML at some point – that assumption could quadruple the size of your payloads and decimate performance levels – instead, be willing to work on the data in its source formats.

For example:

- If a technical requirement is for a large (>20MB) supplier data feed to be posted into a database and the existing feed is just flat text, avoid an up-conversion to XML and put it directly into the database using an optimized data service.
- If a technical requirement is to transform a large (>20MB) XML document and put it on a JMS queue, an ETL engine (as an alternative to XSLT scripts) may speed the transformation and improve the business Service Level Agreement.
- If a technical requirement is to replicate part of a database as part of a BPEL process flow, delegate the work to a Replication Service but keep the control points, monitoring and SLA commitments at the SOA tier.
- If a technical requirement is to load a Business Intelligence cube as part of a composite SCA business service, use a slave process (where SOA is the master process) that is pre-configured to work efficiently with mult-dimensional models.

It sounds trite, but the simple advice for Data Services is to always use the right tool for the job. Too many SOA fans see XML as the solution to every problem when in fact there are hosts of tools far better optimized for the non-XML data formats that are pervasive within typical large businesses. Service-Oriented Architecture is best conceived of as a framework for common control points and re-configuration – not as a universal data layer.

Thus, the low-hanging fruit for Data Services may be boring Web Services with simple data actions, or thin SOA façades for wrapping conventional data technology. But these starting points are the perhaps the most useful and common-sense ways to start a multi-year Data Services plan that truly serves the Enterprise.

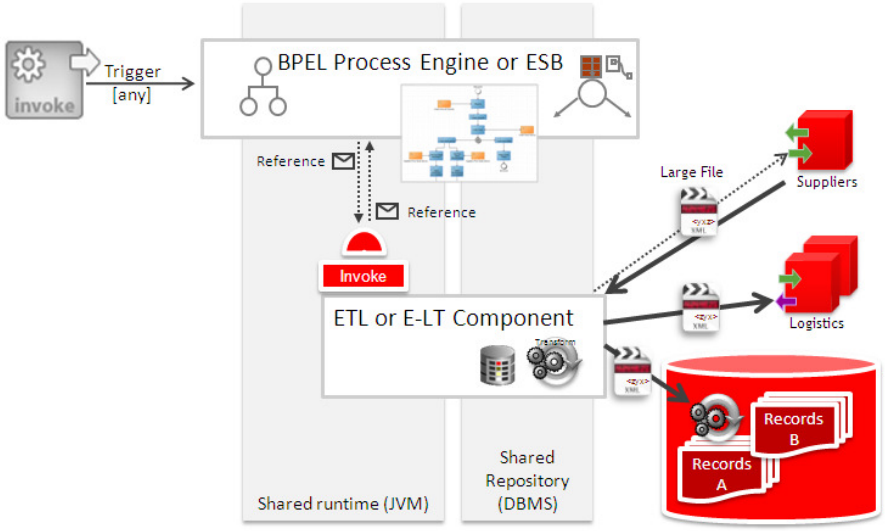


Figure 12: ETL Delegate to SOA, Basic Pattern

Building a rational plan for Enterprise Data Services can be confounding for the average technologist who hears a lot of noise about J2EE frameworks and new XQuery tools. Indeed, while SDO (Service Data Objects, a recently popular J2EE framework for data services) and XQuery engines (frequently promoted by some vendors as data services) are exceptionally useful for many greenfield SOA applications, they can also be a tremendous bottleneck in SOA applications that require access to large amounts of legacy date.

By definition, both the SDO and XQuery engine patterns replicate portions of the core legacy data, either in metadata or in data values themselves. This is desirable when the benefits of the new-found abstractions (as either SDO components or XML documents) are important for a consuming application. But the requisite impedance mismatch (between the new and legacy data shapes) and data replication (using various caching schemes dependent on the vendor) can significantly reduce performance of your data. In cases where performance is a second priority to the benefits your abstraction layer provides, this detriment may not matter.

The Data Services architect must remain acutely aware of the application performance requirements and the additional latency that SDO and XQuery approaches cause in the Data Services layer. The point here being that neither

SDO nor XQuery are required to actually deploy Data Services. In fact, non-SDO and non-XQuery Data Services may well be the most performant Data Services in a given SOA.

The bottom line is that a mature Data Services infrastructure will exhibit a range of architectures, functional services, and delivery styles. To summarize:

Architecture Patterns for Data Services – *where the service runs*

- Basic WSDL/XML Façade – simple WSDL façade to a data source
- Java SDO Proxy – Java abstraction for diverse data sources
- XQuery/XML Proxy – XML layer abstraction for diverse data sources
- Data Service Façade – a pass-by-reference API for conventional data services (replication, migration, integration, transformation, master data…)

High Level Functional Data Services – *what the service does*

- Master Data Service – lifecycle maintenance of golden records
- Batch Data Services – optimized bulk movement & transformation
- Data Access Service – fetching and changing regular business data
- Data Grid Services – optimized caching and clustering of data objects
- Data Quality Services – automated cleansing, matching and de-duplication
- Data Transformation Services – centralized transformation components
- Data Event Services – monitoring for data state, changes and rules

Data Distribution Styles for Data Services – *how to get the data*

- RPC-style Delivery – remote invocation using regular request-reply
- Event-based Delivery – publish/subscribe via queuing type system
- Process-based Delivery – transactions via BPEL or other long-lived XA
- Object Delivery – via marshaled objects in the application language
- Bulk-style Delivery – low level, direct to/from source persistence layer

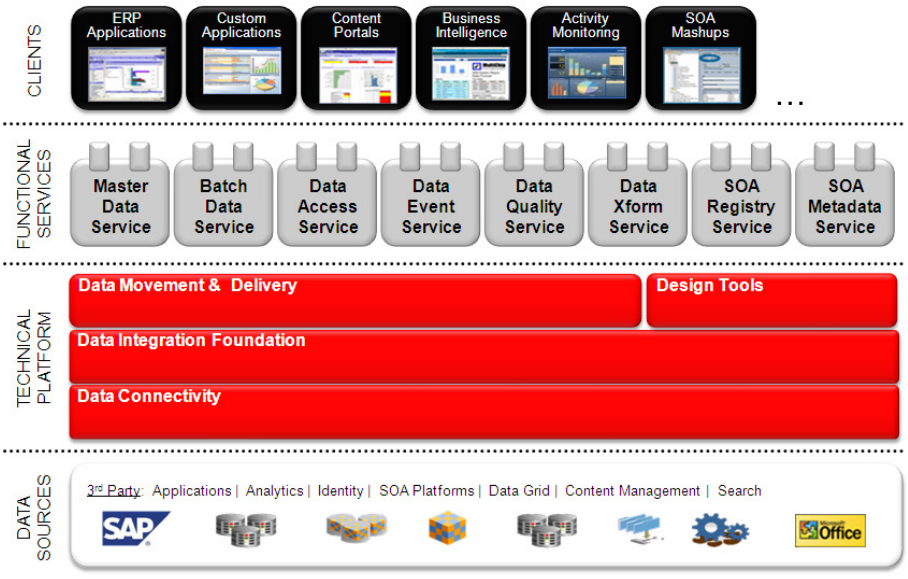


Figure 13: Enterprise Data Services Logical View

No single approach is the best for all possible Enterprise Data Services. And no single functional capability can fulfill all enterprise data needs. In the future of SOA enabled architectures, a hybrid approach for Data Services will dominate.

Business needs and Data Service architects will demand a diverse range of Service Level Agreements that sometimes favor flexibility, sometimes can be isolated in Greenfield systems without legacy data, and sometimes require extreme performance and scalability levels. Enabling software architects to choose the best architecture, functional pattern, and delivery formats are essential for a rational long-term Data Services strategy.

Even allowing for different options than those presented here, we can still be sure that Data Services will be a critical component of any Enterprise scale SOA and that no single technical approach to Data Services can solve all Enterprise data problems. The best guidance for adopting Data Services is to start with the quick project wins, technical low-hanging fruit, and stick with the proven data management patterns leveraged in a SOA context.

PART 5: ORACLE DATA INTEGRATION SUITE FOR DATA SERVICES

Oracle Data Integration Suite is a bundle of best-of-breed products from Oracle, which is specifically helpful in enterprise data integration and SOA Data Service situations. This product Suite aims to improve business operations by decreasing the costs and complexity of data integration at an enterprise scale. For the first time, businesses can unify their conventional data infrastructure with modern, loosely-coupled component-based architectures.

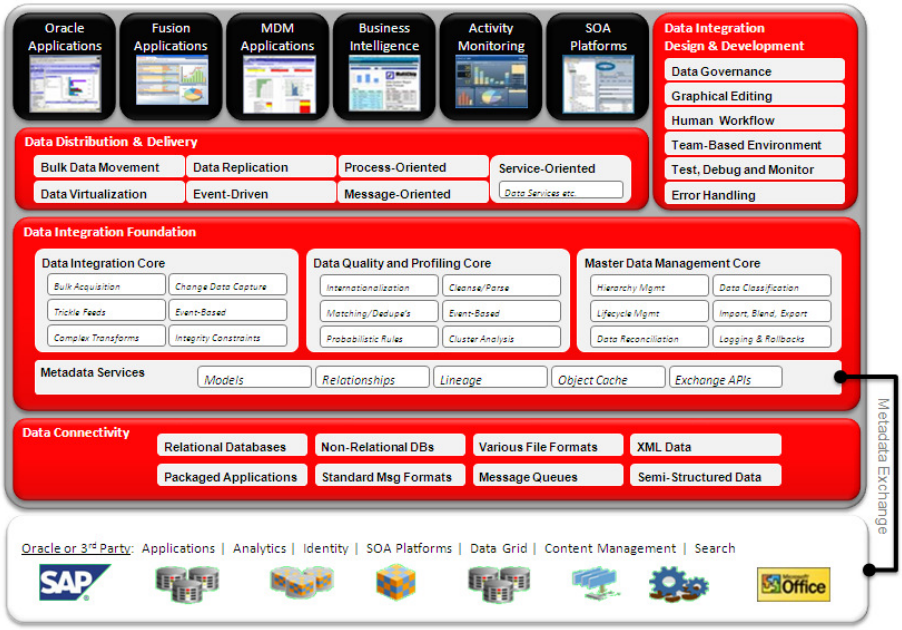


Figure 14: Oracle Data Integration Suite Technical Platform

ODI Suite provides comprehensive technical platform capabilities for data distribution, design tools, a data integration foundation and broad data connectivity. The purpose of these technical capabilities is as follows:

- Data Distribution – provides the high-level access points for all data integration and data services. Data services may be published as SOA-ready Web service end-points, Java APIs, BPEL Process Models, Cached Java objects, or via bulk delivery protocols and formats. This layer

provides a common data distribution framework regardless of the particular client application requirements.

- Design Tools – provide the tooling for people to manage the data integration and data services operations. For enterprise scale operations, there will be multiple roles supported here, including Data Stewards, Enterprise Architects, Process Modelers, and Data Architects. This layer is the administrative and development console for the framework.

- Data Integration Foundation – provides the core technical capabilities for data integration. The common capabilities include data transformation using ETL style techniques, data quality functions for data of all types, and master data services for managing the lifecycle of data records. This layer is foundation for delivering highly-optimized data integration within any enterprise context.

- Data Connectivity – provides access to data in any location, in any format, and over any protocol. Sometimes data integration is best achieve using application APIs and frequently it is best achieved going to the database layer directly, this layer provides access to any point in a source or target software application/system.

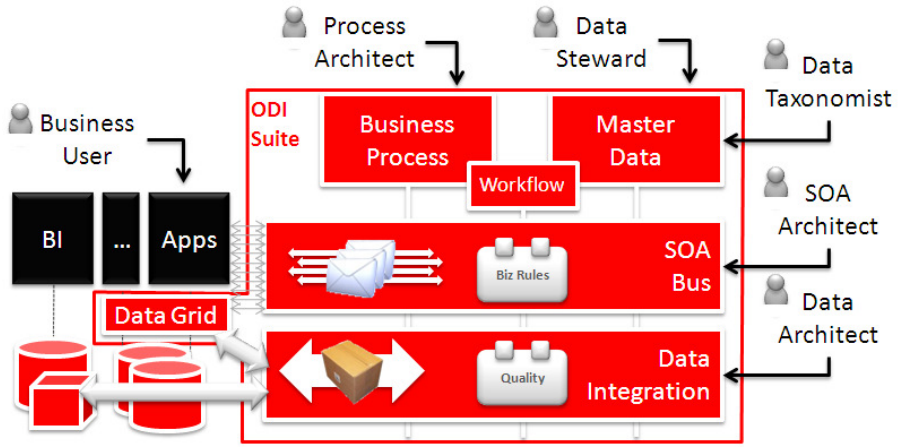


Figure 15: Key Users of the ODI Suite

Functionally, the key users of the Oracle Data Integration Suite are a cross-section of integration and data architects, as well as an emerging practice area called data stewardship. These architects, stewards, and officer types of roles are very important parts of a holistic integration strategy. The following section provides some insight into a few of the typical work roles that might take part in ODI Suite data interactions.

Business Users / Subject Matter Experts (SMEs) / Analysts

Who are they?

- Non-technical functional experts and end-users
- Typically interacting with a computer on a limited basis
- Primary applications will be ERP systems and Office applications
- Sometimes may include line workers and/or other blue collar roles

- They may sometimes use Business Intelligence dashboards, view-only

How will they interact with ODI Suite?

- They may never know that an ODI Suite system exists
- They will only know if their application data is good or bad
- For example, they will be working with Customer records, Supplier records, Asset tracking systems, Product portfolios, etc. Their knowledge of data integration will be limited to how often they have to contend with poor records which they must manually reconcile

Process Architects

Who are they?

- This is a proxy role between the pure business-oriented process modeler and the SOA enterprise architect responsible for the service bus
- Understands business process requirements, and can translate them to technical specifications encoded within BPEL
- Primary application will be BPEL Process Manager

How will they interact with ODI Suite?

- As a core user of ODI Suite, the Process Architects will use the BPEL Process Manager for the full lifecycle of process management
- They will be experts in importing native Business Process Models from other tools, such as Aris/BPA Suite and with optimizing business process flows for high-performance SOA environments
- They will interact with Data Services as end-points in various processes

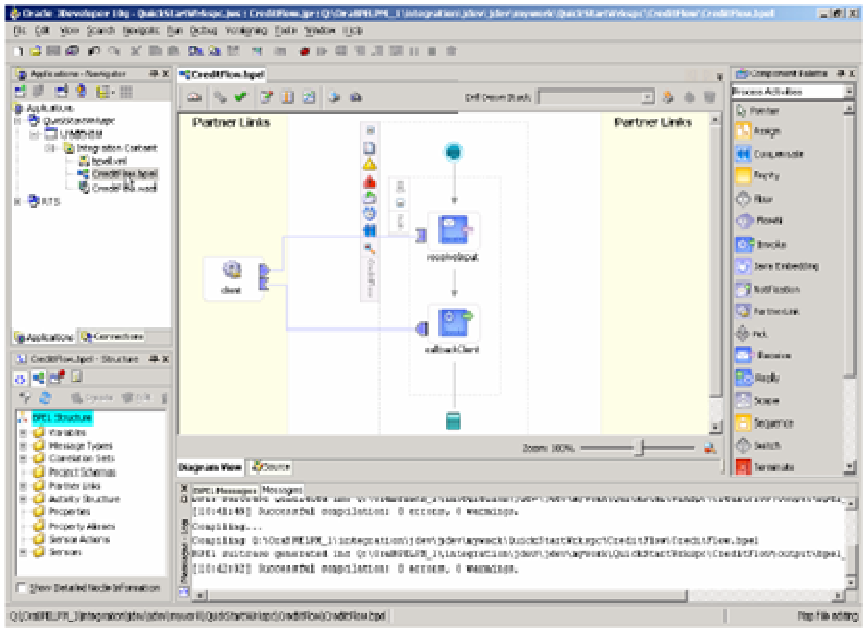


Figure 16: BPEL Process Manager Process Model

Data Stewards

Who are they?

- This is the shepherd / steward / maintenance role: taking care of data

- Understands business requirements and IT objectives – defines and executes the low-level plans to fix the data itself
- Primary applications will be Oracle | Hyperion Data Relationship Manager (DRM) and MDM Hub Applications (the core of the Stewardship function lives within the MDM framework), but also includes some access to ERP systems and MDM Foundation Interfaces

How will they interact with ODI Suite?

- As a core user of Oracle | Hyperion DRM they manage reference data
- They will be experts in finding and navigating the data within DRM and any other MDM applications, they will know which data can be changed, by whom, and how to do it
- They will interact with workflow systems, as a team of Stewards, to respond to tasks that have been set by SMEs and Business Analysts
- They will ensure good data

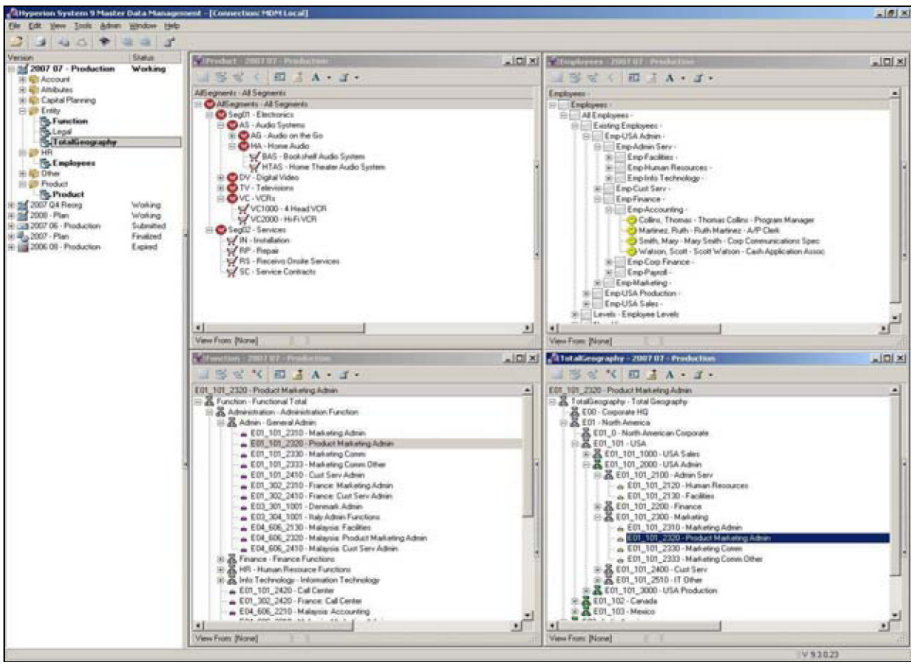


Figure 17: Oracle | Hyperion DRM for Master Data Management

Data Taxonomist

Who are they?

- This is a definitional role: defining categories, entities, and groupings
- Understand the business requirements, IT objectives, and upstream uses of the corporate information
- Models hierarchies, ontologies, tag sets and some data models
- Primary applications will be MDM Applications and Foundation Interface

How will they interact with ODI Suite?

- As a user of DRM and other MDM Applications (eg: hierarchy management, classification, effectivity dating etc)

- They will create and maintain the classification systems (manual and automated) used to organize structured, semi-structured and unstructured content – these may be applied to MDM Applications or exported for use in other systems, such as content management systems, SOA messaging, ETL processes and other runtime tools that use reference data
- They will respond to business users and Stewards requirements by improving the "findability" of corporate data

Enterprise Integration Architect

Who are they?

- This is the blueprints role: designing the systems, schemas and flow
- Understand the business requirements, IT objectives, data formats and design limitations of various technologies
- AKA: Software Architect, Database Architect, Systems Architect

How will they interact with ODI Suite?

- As a user of SOA Suite foundation interfaces (eg: modeling etc)
- They will be experts in the IT systems that feed and are fed by the data integration processes, they will make decisions about latency requirements, scheduling of systems updates, and ensure end-to-end dependability of MDM data and systems resources
- They will respond to requirement set by Analysts and Stewards for new systems participating in the ODI Suite ecosystem of data
- They will set requirements and objectives to Developers and DBAs for implementation design and construction
- They will setup and configure the integration services within the MDM environment, properly leveraging the back-end services provided by the raw middleware function points

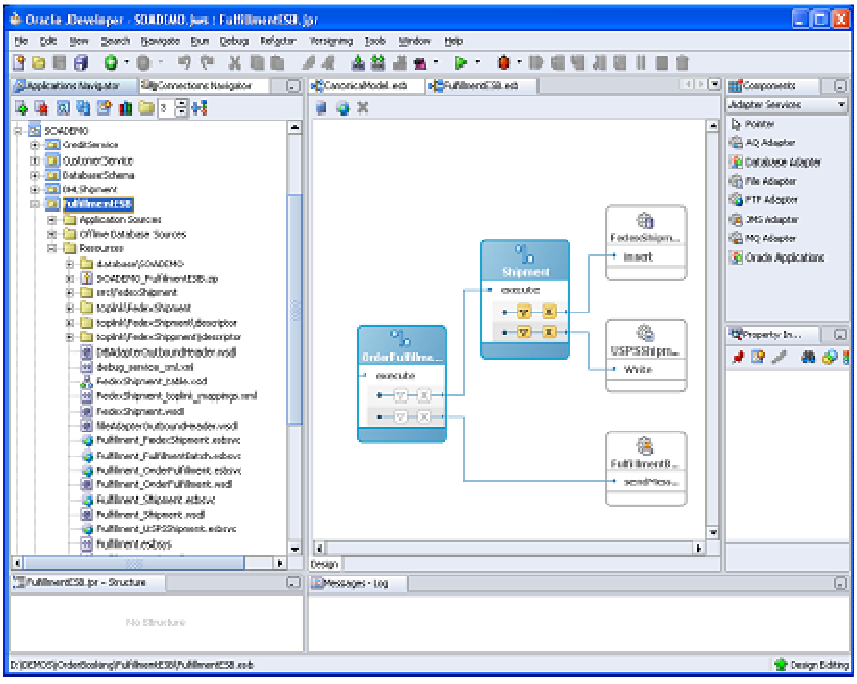


Figure 18: Oracle Enterprise Service Bus Component Configuration

Data Architect and DBAs

Who are they?

- This is the production role: produce new capabilities in IT
- Understand the IT objectives and execute to a plan
- AKA: Software Engineer, Database Administrator, Developer

How will they interact with ODI Suite?

- As a user of ODI Suite foundation data stores (eg: internal workings)
- They will be experts implementing code, mappings, integrations, and configuring the ODI Suite platform itself, and its interfaces to other applications within the overall IT environment
- They will implement data controls, schemas, and ETL interface mappings
- They will respond to requirements set by Architects and Analysts
- They will understand the technical limitations and interface requirements for enterprise data sources, and know how to access data from the low-level bindings and APIs
- They will tune and optimize schema, taxonomy, queries etc

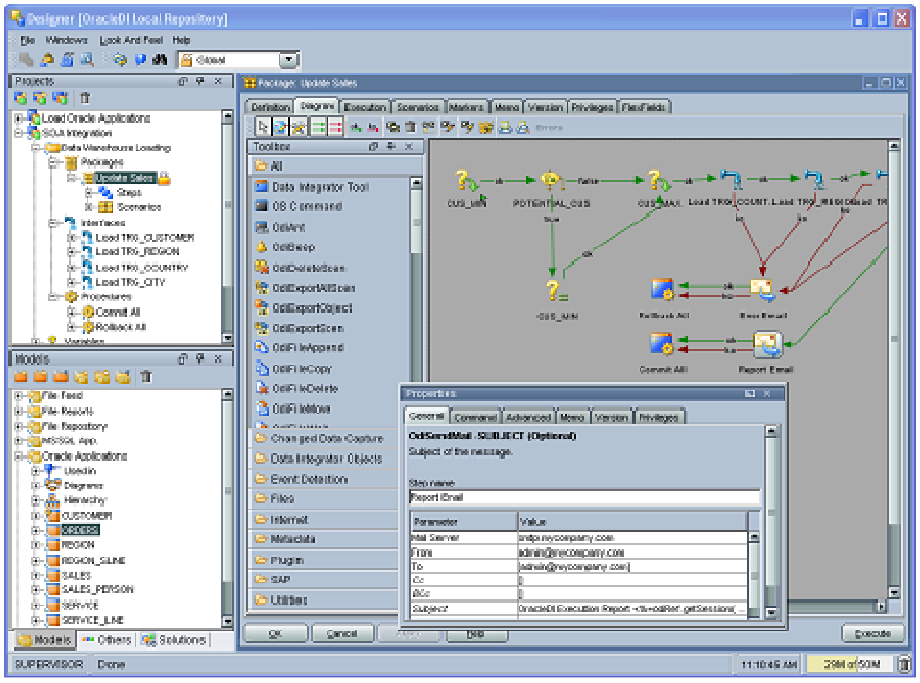


Figure 19: Oracle Data Integrator Batch Integration Processing

The many kinds of end-user roles that the Oracle Data Integration Suite supports may seem intimidating, but is an accurate reflection of the complexity that underlies the average enterprise scale data integration effort. Multiple data access points, managed reference/master data, and conventional ETL batch jobs are all part of a regular enterprise data integration scope. ODI Suite easily handles this complexity in one comprehensive platform.

One way that ODI Suite simplifies this complexity is by using a shared Java runtime for many of the ODI Suite subsystems, this ensures that there's a single

control point, using open and standard Java runtime components, where the various aspects of the ODI Suite components can be managed together. Another way that ODI Suite simplifies the data integration platform, is to provide a common human workflow sub-system across all the ODI Suite components, allowing the various end-users to stay on the same page by reporting and responding to system events within the same workflow.

Despite the incredible breadth of functionality and users the Oracle Data Integration Suite can support across the enterprise, it can be surprisingly easy to setup and configure.

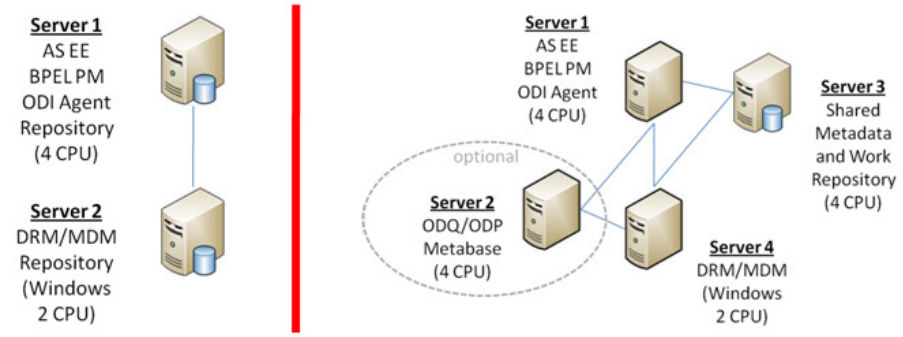


Figure 20: A Simple and a Typical Deployment for ODI Suite

With as few as two servers, the ODI Suite can be configured with all of its base set of included components. A more typical setup would likely include a dedicated database server and possibly add a dedicated server for the optional Oracle Data Quality component.

In this remarkably small package, the Oracle Data Integration Suite will provide a single unified control point for three foundational integration patterns:

1. **Process-centric Integration** – with an emphasis on the business view, long-lived and complex multi-step transactions are grounded within a closely managed business process flow
2. **Message-based Integration** – application layer integration ensures business logic is respected by the middleware, and a SOA approach places priority on flexible, loosely-coupled binding points
3. **Data-based Integration** – the efficiency of point-to-point data interchange is enabled by a SOA-controlled sub-process for executing data integration directly to/from the data tier, and with exceptionally high performance

These three integration patterns are essential parts of a well-rounded enterprise data integration strategy for enterprise systems. Oracle's Data Integration Suite starts with three key components to fulfill best-of-breed functionality for each of the three key integration styles, they are:

1. **Oracle BPEL Process Manager** – a powerful and standards-based process control point for transactional systems of all types, it includes bi-directional interaction with business process management platforms for business user consumption

2. **Oracle Enterprise Service Bus (ESB)** – a high-performance messaging system that handles all publish/subscribe, mediation, XML document transformation and message routing capabilities
3. **Oracle Data Integrator** – an exceptionally fast extract, transform and load (ETL) platform for handling large data payloads of any type, and loading any database or business intelligence system from the SOA tier

But the ODI Suite goes beyond these three integration patterns to supply Master Data Management capabilities that are suitable for managing reference data of all kinds and financial data in particular. The Oracle | Hyperion Data Relationship Manager (DRM) is formerly the master data system for Hyperion's popular financial planning and management applications, as well as a master data dimension management system for the Essbase business intelligence cube. The DRM component is a critical enabler for keeping business reference data aligned throughout the business.

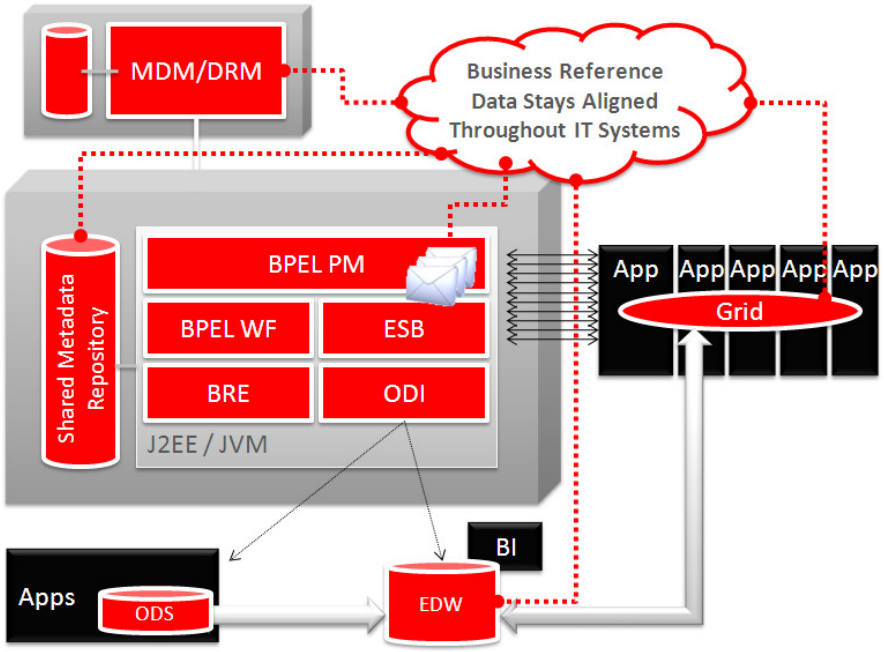


Figure 21: Keeping Reference Data Aligned, from Grids to Warehouses

The optionally available Oracle Coherence Data Grid and Oracle Data Quality may provide inline capabilities for improving overall data quality and pushing data to business applications for extremely low-latency data access. For example, the Coherence Data Grid can expose a near-cache subsystem to any Java, .Net, or C++ application such that data is accessible in millisecond level transactions. This kind of reliable sub-second speed at very high data rates is only achievable with data grid technology. Using the Oracle Coherence Data Grid as part of the ODI Suite means that high value master data can be intelligently distributed directly to this shared data object pool, for application consumption in the most demanding performance situations.

Oracle Data Quality and Data Profiling can cleanse, parse, standardize and de-duplicate data as it flow anywhere in the ODI Suite infrastructure. Typically, this process is used to clean up bad data before it arrives in an Enterprise Data

Warehouse (EDW), but can also be used to scrub the data before loading the data grid, operational data stores (ODS), or any component in the ODI Suite.

A full enumeration of the standard and optional components of the ODI Suite is as follows:

Included Features	Optional Features
• Data Integration Designer • Data Connectivity Foundation • Metadata Management Services • Data Integration Core Services • Master Data Management Services • Data Movement & Delivery Services • Scheduling & Workflow Services • Standard Data Governance & Integrity • Standard Adapters & Connectivity	• Advanced Data Quality • Advanced Data Profiling & Audit • Data Grid / Object Cache • Unstructured Text Adapters • EDI Adapters • ebXML Adapters • RosettaNet Adapters • Healthcare Adapters • Application Adapters

Base Components:	Oracle BPEL Process Manager, Oracle Application Server Enterprise Edition, Oracle Enterprise Service Bus, Oracle Business Rule Engine, Oracle Data Integrator, Oracle | Hyperion Data Relationship Manager
Optional Components:	Oracle Data Quality, Oracle Data Profiling, Oracle Coherence Data Grid, Oracle Data Adapters (specialty)

To get a more practical understanding of the Oracle Data Integration Suite, consider a realistic business scenario. A global financial institution offers thousands of unique financial products, that are available in different geographies and regulatory environments, but needs to maintain centralized visibility and operational consistency with the identification codes of these thousands of product codes. Further complicating matters, there are different account systems, general ledger, and reporting environments throughout the various front, mid, and back-office systems within this multi-national organization.

How can Data Services within ODI Suite help?

Take for example a large financial institution that must simultaneously support high-demand, high-availability transactional applications, messaging integrations for thousands of application instances, and thousands of operational data stores and data warehouse grids. Traditionally these architectures would have each required fundamentally different infrastructure, from different vendors and with few overlapping solutions. But there is, and always has been one significant commonality among those diverse infrastructures – the data. Core business data types like Customer, Product, Order and others are connected across systems despite the relative isolation of different enterprise infrastructure patterns. But why should they be?

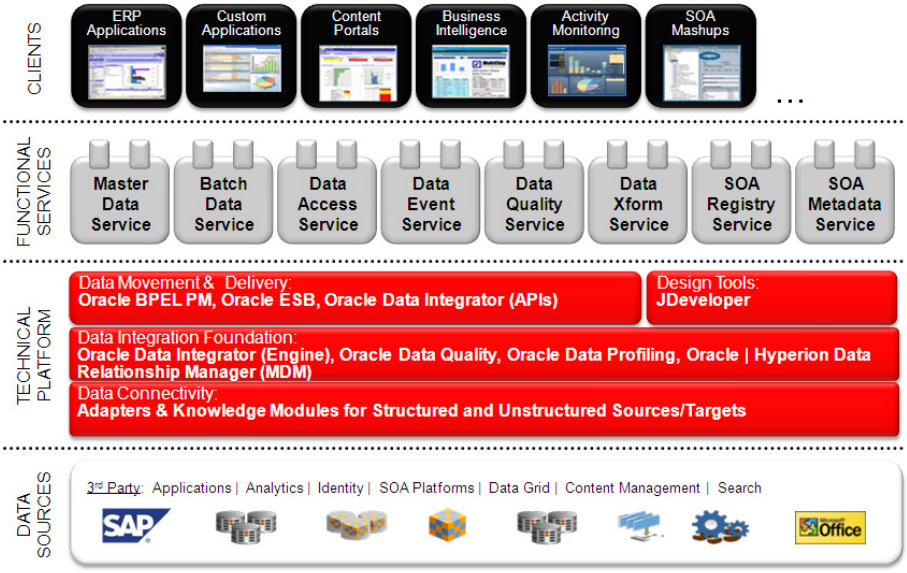
A modern Data Service architecture should support synchronizing data grids with master data, publishing high quality canonical data within a messaging infrastructure, and expose control points for commanding business intelligence and data warehouse systems as loosely-coupled services.

Put even more simply, a smart Data Services infrastructure will be capable of sharing business reference data across systems, regardless if any of those systems

were of different types. Typical enterprise software infrastructure systems that would benefit from transparent business reference data include:

- Messaging Systems (ESB, JMS, EAI, EDI…)
- Data Integration Systems (Replication, Migration, ETL…)
- Data Warehouse Systems (ODS, EDW, Appliances…)
- Master Data Systems (System of Record, Master File, Hubs…)
- Business Applications (Application Data Grids, Verticals, ERP…)

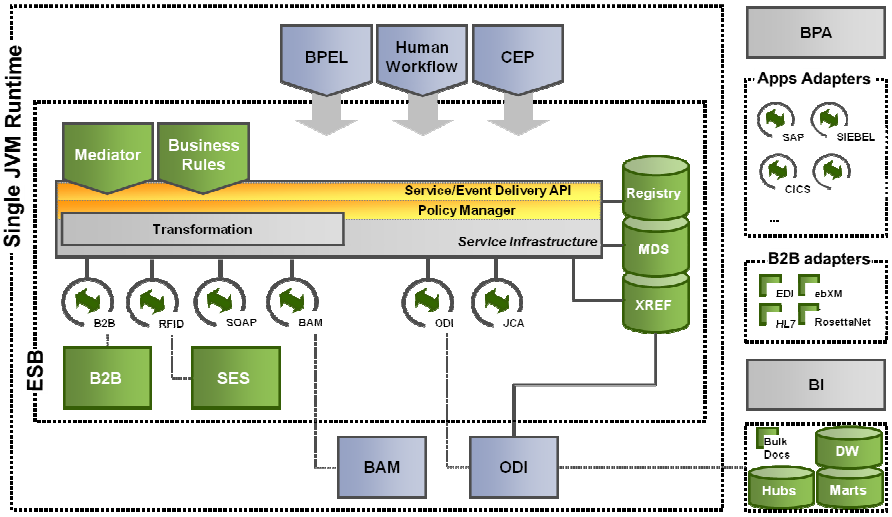
This vision is not so much a dream, as it is a requirement for modern information-centric businesses that hope to use information technology as a competitive edge within their industries. Yet regardless of how grand the IT strategy might be, a good Data Services plan will first solve fundamental tactical issues that simplify the use of data throughout all enterprise architectures.



Oracle Data Integration Suite 10g is a comprehensive set of enterprise software to address these tactical integration and reference data management challenges. Oracle achieves this with uncompromising modern integration points among the various integration components. For example, Oracle is the only enterprise software vendor that can provide a single-runtime solution for:

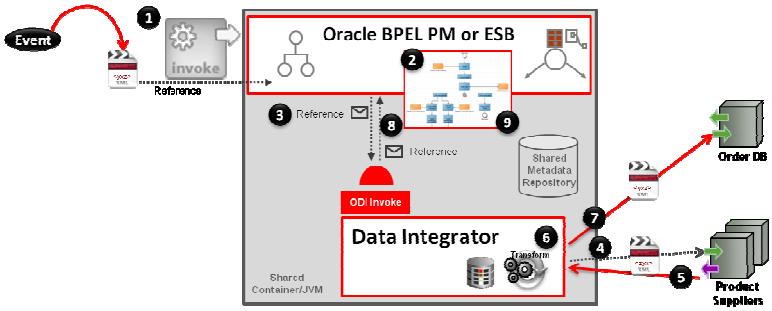
- Business Process-based Data Delivery
- High-Performance Message Bus for Data Delivery
- High-Performance Data Integration/ETL

Single runtime capabilities simplify the deployment of enterprise-scale data integration while also providing tighter integration among the components. For example, execution within the same Java Virtual Machine (JVM) means that it is possible to make native invocations among each of the BPEL, ESB, and ETL components using far more optimized bindings. Also, it becomes much simpler to use monitoring and management software for watching the status of events and system overhead when the components share the same runtime.



Additionally, there are several possible and pre-built integrations among the ODI Suite components which include the following:

- BPEL PM to ODI Web Service Invocation – it is an out-of-box capability for any BPEL process to invoke any ODI job as part of the BPEL Partner Link services, use cases may include:
  - Large Document Transformation for SOA
  - DB to DB Replication for SOA
  - DB Loading / Business Intelligence Refresh for SOA
  - CDC Data Event Propagation for SOA



- ODI to Data Quality Package Tools – the deployment of any ODI job, or any transaction which calls an ODI job, can easily embed a data quality function for cleansing, parsing, standardizing and de-duplicating data as part of that transaction

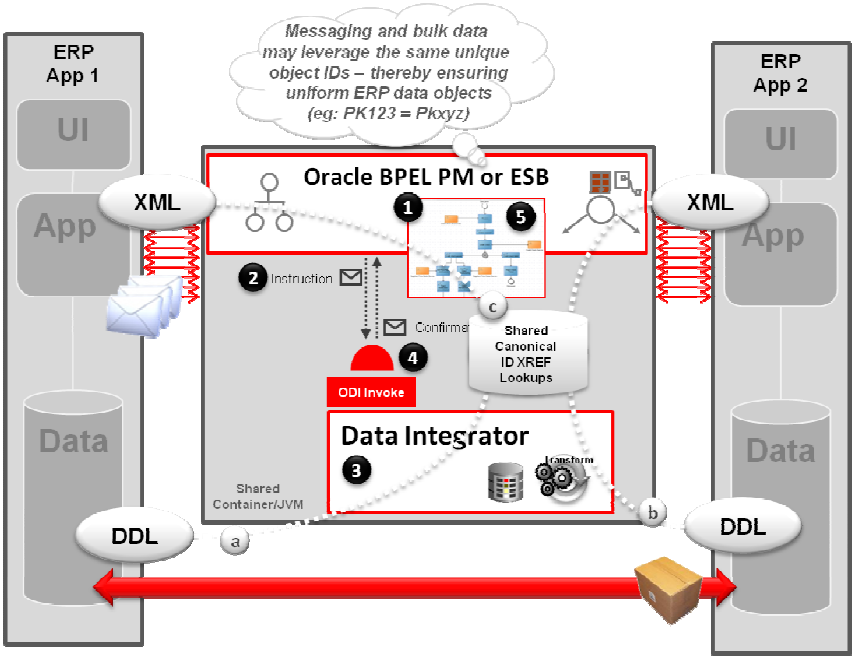
- DRM to BPEL Human Workflow – for the use of a human workflow process during the maintenance and management of master data functions, including multi-step approval processes

- ODI to BPEL Human Workflow – an Error Hospital capability that enables Data Stewards to track and repair data records that fail during batch data integration jobs, thereby simplifying the recovery and recycling

processes so that non-technical users can repair data

- DRM to ODI for Reference Data Lookup – an Import/Export Profile capability within the Oracle | Hyperion DRM system allows specific hierarchies and reference data to be used as part of a batch process, typically for "lookup table" style functionality

- ODI to ESB Common Data Object ID XREF – both ODI and ESB may consistently use the same common, globally unique IDs for referencing canonical application business objects across XML service bus and ETL transactions



- DRM to BPEL/ESB for Reference Data Lookup – a realtime API enables DRM to respond to messaging system requests for hierarchy lookups, improving the quality of on-the-wire messages

- BPEL/ESB and Business Rule Engine – leveraging production business rules within any BPEL or ESB process for a more declarative and rule-based business workflow

- ODI Populating Data Grid – data movement and transformation can serve many different kinds of target technologies, including the Java-based Data Grids – ODI may write data to Grid APIs in sequence or in parallel to writing to data warehouses/data stores

- BPEL Dehydration using Data Grid – long-lived data delivery processes for data integration can cache themselves in-memory using Grid features, thereby accelerating performance and reliability when transactions resume

These are just some of the more interesting interoperability points among ODI Suite components. Regardless of the integration points which are technically interesting today, the bigger value lies in the exceptional flexibility of the core infrastructure to be reconfigured in new ways – with minimal overhead and effort in the infrastructure tier. This exceptional re-configurability is a central feature of a Data Services approach, and the basis of any successful long-term strategy for enterprise data management.

Business requirements and data architects will always demand a diverse range of Service Level Agreements (SLAs) that sometimes favor flexibility over speed, sometimes can operate in relative isolation, and sometimes require extreme availability, performance and scalability levels. Choosing the best mix of architecture, functional patterns, and delivery formats is essential for a rational, business-driven long-term Data Services strategy.

Finding a single platform that can deliver this kind comprehensive flexibility should be on the short list of to-do items for any architect who is seriously exploring their Data Service alternatives.



The Case for Enterprise Data Services in SOA
January 2008

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2007, Oracle. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change without notice.
This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.
Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.