IBM Global Services

Service Oriented Modeling and Architecture (SOMA)

Version 3.1 – Information Discipline

White Paper

Version 1 – August 31, 2007

Authors:
Abdul Allam (allam@us.ibm.com)
John Kling (jkling@us.ibm.com)
David McCarty (davidmccarty@fr.ibm.com)
Guenter Sauter (gsauter@us.ibm.com)
Peter Worcester (pworcest@us.ibm.com)

Abstract and Purpose

The purpose of this document is to provide practitioners who are applying SOMA an understanding of the role of information architecture in the SOMA phases and how the information discipline enhances SOA engagements. Specifically, this paper focuses on the following objectives:

- It briefly defines the relationship – and actually the overlap – of information architecture (Information On Demand[1]) and service oriented architecture, known as Information as a Service. It is important to understand the architecture aspects such as the alignment of SOA and Information On Demand first since the SOMA method leverages the underlying architecture.
- It outlines the tasks, techniques and deliverables for the information specific architecture activities such that an architect can understand how these tasks contribute to the overall solution design and can perform these tasks[2]. The activities of the information discipline – specifically in the identification and specification phase – are generic in the sense that they do not assume a specific solution that requires information services.
- It defines information services and gives practitioners guidance and indicators to appropriately select and specify them.
- It gives examples to help the practitioner understand issues of scope, depth and granularity.

The SOMA method itself is not explained here and prior SOMA familiarity is assumed for the reader (see Sect. 8.1 Reference [1]).

Acknowledgements

We would like to acknowledge the contributions from Brian Farish and Liang-Jie Zhang to this paper and thank our reviewers Emily Plachy and Ali Arsanjani.

[1] Information on Demand is an IBM initiative that helps customers to better leverage their information so that it become a strategic asset; see Section 2.2 for more details.

[2] We leverage existing work products and best practices from various traditional information architecture domains.

Contents

1. Introduction

IBM has developed the Service Oriented Modeling and Architecture (SOMA) method in order to assist practitioners who are engaged in Service Oriented Architecture (SOA) projects. In late 2005, the version 2.4 Technique Paper (see Reference [1]) was published with input from many experts in this domain. IBM practitioners have been trained on the method and applied it in many engagements. SOMA 2.4 was received extremely positively not only by the IBM practitioners but also by our clients. Due to the significant experience in client engagements with SOMA, a lot of lessons have been learned and requirements for future enhancements were collected. One area that has been significantly enhanced in SOMA 3.0 is the information perspective of SOA. In addition, SOMA 3.0 extends SOMA 2.4 to cover micro design, build, test, and deploy activities.

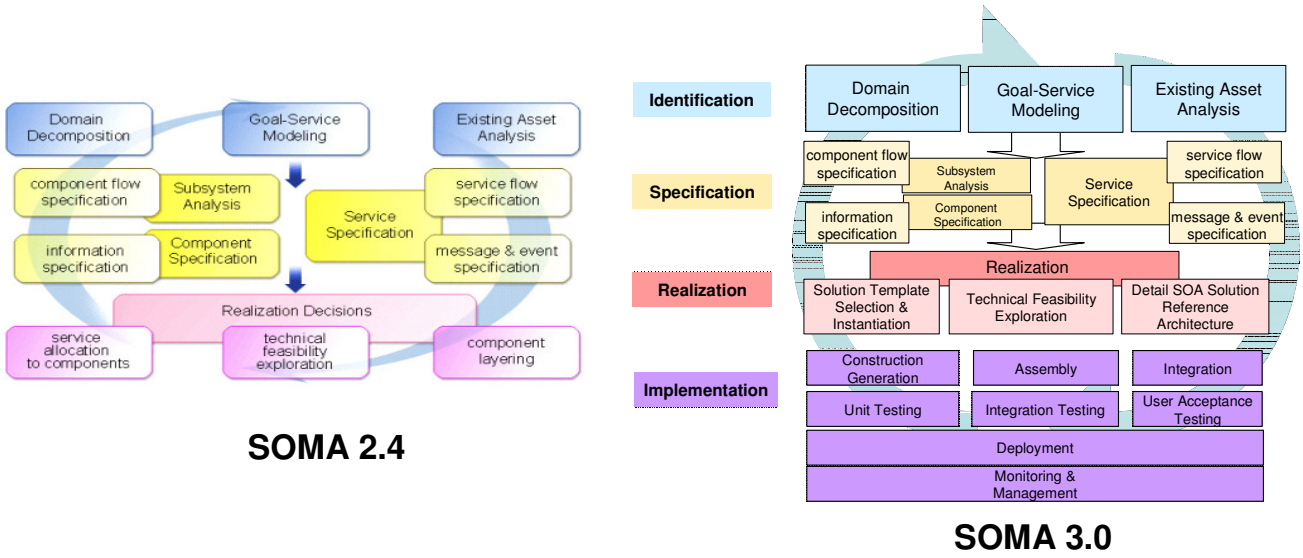

Figure 1: SOMA 3.0 Extends SOMA 2.4

1.1 The Information Discipline

In SOA we construct business solutions using a building block approach across multiple architectural layers and functional domains. SOMA brings a methodical approach to the analysis and design of these building blocks, and especially focuses on the processes, the services and the service components. Each of these architectural elements encapsulates data and behavior behind a formalized interface so that business solutions can be assembled from a known set of reusable parts.

For this approach to be successful the elements must be carefully designed to ensure they provide clearly defined capabilities with high qualities of service and broad reuse. The SOMA information discipline brings a set of structured techniques to address the data aspects of SOA design. The goal is that by understanding what business information exists in the solution, informed decisions can be made to ensure that information is leveraged in ways that best support the technical and business objectives of the SOA solution:

- That services are reusable across the entire enterprise
- That the business data exposed to consumers is accurate, complete and timely.
- That data shared across business domains and technology layers has a commonly understood structure and meaning for all parties.
- That the core data entities linking together the business domains of an enterprise are consistent and trusted across all lines of business.
- That an enterprise gains maximum business value from its data and data systems.

These objectives are valid for all parts of an SOA solution regardless of technology and implementation choices. Exposing an existing application API as a service for example requires an understanding of the data being exposed: is it reliable and accurate? How does it relate to other data in the enterprise? Is it being presented in an understandable format for the consumers? Applying a structured approach to data analysis, modeling and design in an SOA project leads to a solution implementation that is better at meeting existing business requirements as well as being better prepared to adapt to new ones.

Another significant benefit of performing data architecture in the SOA solution design is that it can help identify cases where information technology and specifically Information On Demand (see Sect. 2.2) provides the best implementation choice for a specific SOA components. This paper describes the identification, specification and design of so called information services which leverage the capabilities of information systems to provide the realization of SOA service and service components in the solution.

The SOMA 3.1 information discipline outlines a comprehensive approach to data architecture in an SOA project. Most of the activities discussed apply to any service and not just information services. However, not all of the activities are equally important for all engagements. When we describe the activities in more detail in the main part of this paper, we position their role in SOA, when they provide value and when they are not mandatory. We also summarize the activities at the end of the paper and give an overview of their relevancy (see Sect. 7). Before this, we introduce some of the fundamental data architecture themes that underpin the rest of the paper.

1.2 General SOA Design Activities

From the beginning of an SOA/SOMA engagement throughout its duration there is often no **common business glossary that defines the terms** related to processes, services, and data. A common vocabulary which controls the common definition of terms is required to correctly implement SOA: without an agreement what we mean by a customer, member, etc, we cannot correctly implement services related to those terms. Some clients struggle internally with a clear definition of business terms related to a project, such as what do we really mean by a customer or a member. Often, IBM practitioners discover those inconsistencies while trying to learn the accepted business language and abbreviations. It is critical to have a common understanding between business analysts and the technical community on the terminology used in processes and services, and also the underlying data. If we differ in the meaning of the input / output of a service, it is unlikely that a service implementation can be successful.

Consistency related to the terminology is a good starting point when designing services, but is not necessarily sufficient. In some SOA projects, architects have designed services without considering guidelines from the information architecture community on how to best model entities and relationships between them. The input

and output parameters of services (i.e. the messages) are often far more complex than single data types. They represent complex definitions of entities and the relationships between them. The development time and quality of SOA projects can be greatly improved if SOA architects leverage data models when designing the input / output formats of service models. The resulting **alignment of service and data models** accelerates the design, leverages normative guidance for data models and avoids unnecessary transformations.

Practitioners who have considered the concepts described above can deliver service designs with a high degree of consistency across models and metadata artifacts. However, this is no guarantee that the quality of the data that is being returned by services will be acceptable. Data which meets the rules and constraints of its original repository and application may not satisfy requirements on an enterprise level. For example, an identifier might be unique within a single system but is it really unique across multiple systems? Quality issues may not become apparent within the original single application but may cause significant problems when exposed more broadly through an SOA on an enterprise level. For example, missing values, redundant entries, and inconsistent data formats are sometimes hidden within the original scope of the application and become problematic when exposed to new consumers in an SOA. It is therefore important to **conduct a data quality assessment** – sometimes called **data profiling** – when planning to realize services, so that appropriate actions can be taken to resolve data value inconsistencies.

The issues and concepts described so far are generic in the sense that they apply to any service in an SOA. As motivated above, data modeling and data profiling can provide value to the consistency of services and its output data irregardless of the type of service.

1.3 Information Service Specific Activities

There is a category of services for which the realization depends on capabilities from the information discipline, or Information On Demand and/or where a separation of information from applications and processes provides benefits. These services are called **Information Services**.

Most SOA projects do not start on a green field but are based on an existing IT environment. Some of the challenges are unique to SOA but more often than not well known problems in traditional data architecture fall within the scope of SOA as well. The condition of a typical organization's data environment is often not where it needs to be before the organization can begin an effective SOA transformation. From an enterprise perspective, there's often a lack of authoritative sources offering a complete and accurate view of the organization's core information. Instead, we find a wide array of locations and technologies used for storing and processing data. Many large IT organizations have their core enterprise information spread out and replicated across multiple vertical systems, each maintaining information within its specific context rather than the context of the enterprise. This leads to inconsistencies within the business processes. Information On Demand capabilities– in particular content, information integration, and master data services – can be leveraged to realize information services that provide accurate, consistent, integrated information in the right context.

Consider the lack of an authoritative trusted source or single system of record as an illustrative example. Suppose that in an organization's supply chain systems' portfolio there are five systems that hold supplier information internally. Each of these can be considered a legitimate source of supplier data within the owning department. When building a service to share supplier data, what should be the source of supplier data?

- Is it one of the five current systems that have their own copy of the supplier data? If so, which one?

- Is it a new database that's created for this specific purpose? How does this data source relate to the existing sources?
- Does data have to come concurrently from all of the five databases? If so is it the responsibility of the data architect, the service designer, the business process designer, or the business analyst to understand the rules for combining and transforming the data to a format required by the consumer?

Reusable, strategic enterprise information should be viewed as sets of business entities, standardized for re-use across the entire organization, and complying with industry standard structures, semantics and service contracts. The goal is to create a set of information services that become the authoritative, unique, and consistent way to access the enterprise information. To enforce accessing any information only through an application will limit the scope of the information to the context of the application rather than that of the enterprise as required in an SOA. In this target service-oriented environment, an organization's business functionality and data can be leveraged as enterprise assets that are reusable across multiple departments and lines of business. This enables the following characteristics of the enterprise's data environment:

- Single logical business entities from which to get a consistent and complete view of information through service interfaces.
- Increased awareness of the profile and characteristics of the data in an enterprise context
- Improved data quality across the enterprise
- Enforced data standards
- Data that's clearly visible and readily accessible
- Reduced reliance on custom interfaces and proprietary formats
- Clearly identified authoritative data sources that are effectively used throughout the enterprise
- Security that's "baked into" the solution, and not an afterthought
- Data that's easily discoverable by potential consumers across the organization

Information as a Service is about leveraging information architecture concepts and capabilities – as defined through Information On Demand – in the context of SOA. There are important capabilities and concepts in SOA that are not related to Information On Demand and vice versa. But there is also a substantial overlap between them – such as leveraging content, information integration, and master data services –which significantly improve the delivery of an SOA project.

1.4 Structure of the Paper

This technique paper outlines a structured approach to addressing architecture concepts from Information On Demand in an SOA project. The tasks, techniques, deliverables, normative guidance, and examples are part of the SOMA method.

Since the SOMA method is based on the underlying architecture, we will first describe in Section 2 the alignment between the information architecture – defined through Information On Demand – and SOA. We will then give in Section 3 an overview of SOMA, focusing on the tasks that are related to the information discipline and their position in the overall approach. Section 4 – 6 describe then in more detail the new tasks according to the various phases of SOMA.

2. Alignment of SOA with Information Architecture

To understand the information discipline of SOMA, we need to understand the SOA solution stack (Section 2.1), and Information On Demand (IOD, Section 2.2) and their relationship. This is the basis for Information as a Service (Section 2.3).

2.1 The SOA Solution Stack

The SOA solution stack (Reference [3]) includes a logical reference architecture that describes SOA. This is shown in Figure 2. It provides a high-level abstraction of SOA factored into layers, each addressing specific value propositions within SOA.

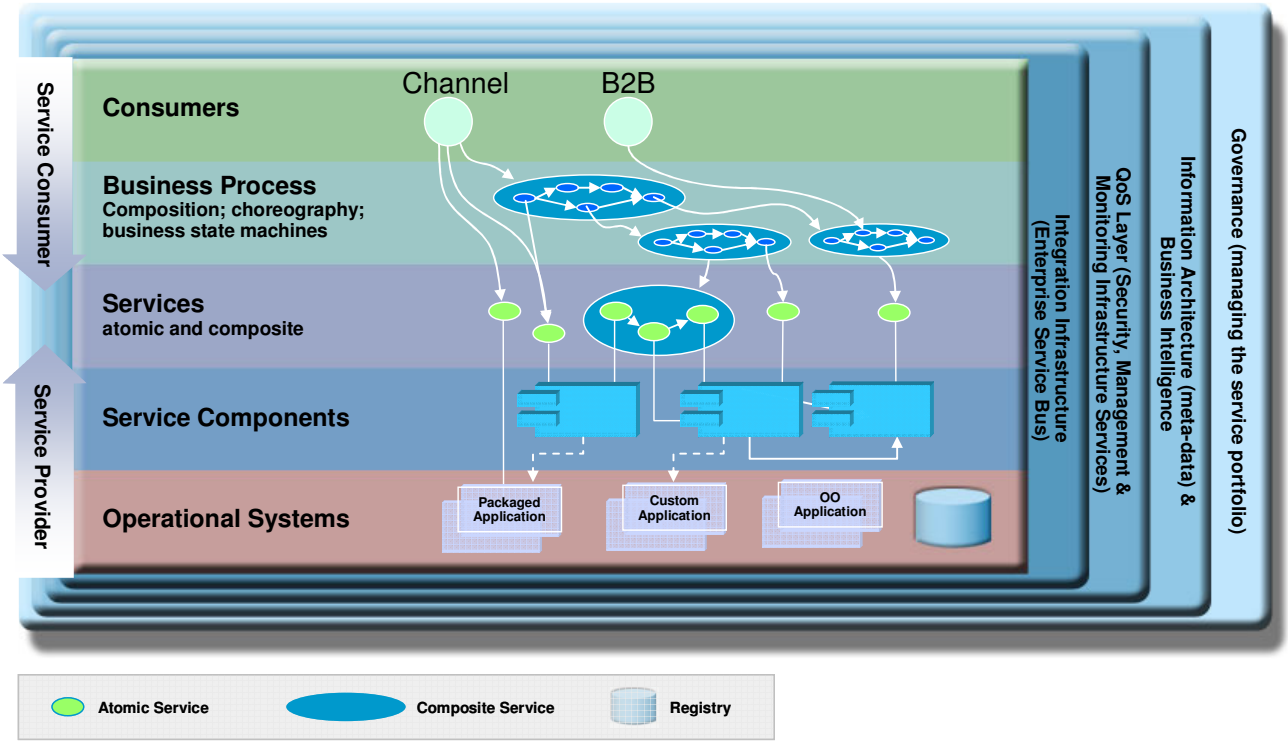

Figure 2: SOA Solution Stack Reference Architecture

An SOA solution stack is a loosely coupled architecture in that a layer is not strictly hidden from the layers above it. For example, a consumer may choose to access a service through either the business process layer or the service layer. A service may also choose to leverage two styles of service implementation: namely, service components like EJBs or .NET components or packaged applications such as SAP or Siebel or other SOA-enabled legacy applications.

The logical view of an SOA depicts it as consisting of a set of layers, architectural building blocks, architectural and design decisions (see Reference [3]). This helps in the understanding and implementation of separation of concerns and assists in the process of creating an SOA in conjunction with methods such as SOMA. The SOA solution stack defines a blueprint that can be used to design an SOA.

A service consumer may access a service through the service layer directly or through the business process layer. A service provider will provide an implementation of a service in a service component or through the wrapping of an existing system or packaged application. Further, the service provider will ensure quality of service through the quality monitoring and management supplied by the quality of service layer. The communication between a service and its service component implementation or operational system will occur via the integration layer. If there is a point-to-point connection (discouraged), it will come through this layer; if there is an enterprise service bus, it will also reside in the integration layer.

The information architecture layer in the SOA solution stack provides data, metadata, content, information integration, master data and analytic services that are relevant within SOA. The following section introduces first the general concepts of Information On Demand before we focus in Section 2.3 on Information as a Service.

2.2 Information On Demand Reference (IOD) Architecture

The Information On Demand reference architecture defines IBM's view on the expanded set of capabilities and concepts to address the need of information architecture. Historically, data architecture focused on storage and retrieval of structured data. More recently, many customers need to address more complex business challenges due to globalization, mergers & acquisitions, risk & compliance, supply chain optimizations, customer loyalty, etc. The fact that information today resides in silos – without a common definition or format, isolated, incomplete, inaccurate, and often not in the context that the business needs – aggravates this problem significantly. CEOs recognize that they need to do a better job leveraging information to respond to those requirements (see Reference [4]). IOD provides capabilities that allow organizations to turn information into a strategic asset and to manage it accordingly.

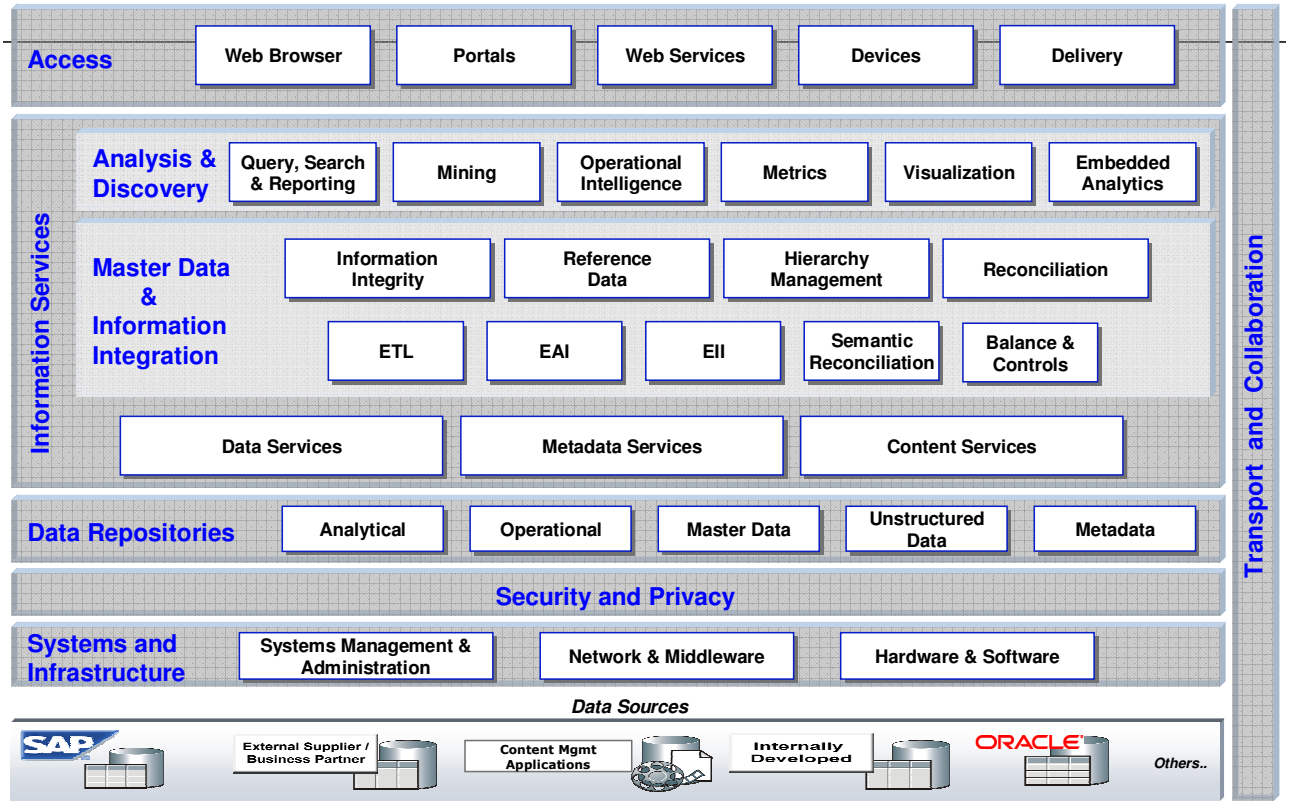

Figure 3: Information On Demand (IOD) Logical Architecture

The IOD logical architecture includes a set of major capabilities that we will briefly introduce. The architecture elements are embedded into an existing environment shown in the "data sources" layer of Figure 3. General system and infrastructure components as well as security and privacy concepts are the foundation of the architecture. IOD considers many types of data and data repositories such analytical data, operational data, master data, unstructured data, and metadata. The core capabilities of IOD are categorized as "information services":

- **Data Services**

  This component manages access to data stored in any type of data storage. The data storage may be a reference data store, operational data store, data warehouse or data mart. The data can be stored in any technology including relational databases. Data services can be divided into two categories: core data operation services include create, read, update, and delete (CRUD) operations. Data management services include functionality such as logging, subscription, and distribution services.

- **Metadata Services**

  Various user roles perform (CRUD) operations on metadata-related artifacts (e.g. data models), data (analysis) assessments, data integration / data flow specifications etc. through various tools. In order to ensure the consistency and improved collaboration, metadata services help to manage the artifacts

- **Content Services**

  This component manages unstructured information such as documents, media, files, etc. The scope of this component includes basic CRUD operations as well as more advanced access mechanisms

(content federation and search) and content-oriented capabilities such as imaging, archiving, records management and content-specific workflow support.

- **Information Integration Services**

  Data is often not in the format, shape and level of quality that an information consumer (reporting tool, application, portal, etc.) needs. Information integration services provide capabilities to understand and assess the data as it is currently stored in one or many legacy systems. It supports the cleansing, transformation, integration, and delivery of the data into the format and context that the consumer needs.

- **Master data Services**

  Master data are the entities that are most critical to the success of an organization such as customer, product, etc. Even though the success of an organization relies on them, the data itself is often inconsistent, inaccurate, incomplete and distributed across application silos. Master data management provides the single version of truth for master data in a defined (logically) centralized repository and synchronizes the master data across the existing legacy environment.

- **Analysis and Discovery Services**

  This component takes raw data – structured and/or unstructured – as an input to aggregate, enrich and derive additional insight that can then be stored in specialized repositories such as data warehouses, data marts, etc.

The transport and collaboration layer provides interoperability between IOD components and its user community with other domains within and beyond the enterprise. The access layer provides various channels for users to access the information and capabilities from IOD.

The traditional context where we apply IOD focuses on virtualized data, unconstrained access to structured and unstructured information, single version of the truth, metadata management, deep analytics, advanced search, optimized data movement & placement, data quality, and data governance. More recently, the emergence of SOA has made it important to consider the role of IOD in the SOA context, defined by Information as a Service.

2.3 Information as a Service

IOD focuses on information-intensive processing to store and retrieve structured and unstructured information. The main interfaces are traditionally the SQL language and content-specific APIs. They allow the consumer to retrieve information in almost any format through generic interfaces. Many information consumers – in particular in the business intelligence domain – rely on that flexible and generic interface.

In contrast, service interfaces defined in SOA expose a limited set of relatively specific services. The goal is to clearly define the semantics of each specific service and to register and govern it in a registry so that it can be reused by various consumers across and beyond the enterprise. Therefore, SOA introduces another type of interface for information access through services which is more controlled and more reusable than the traditional interface.

2.3.1 Classification of Information Services

IOD provides valuable capabilities for SOA, specifically to deliver trusted and integrated information – structured and unstructured – as services in an environment of diverse information of uncertain quality. Throughout the remainder of the paper, we will discuss those capabilities in much more detail (see Sect. 6.3). Below is a summary according to the structure of the IOD logical architecture

- **Data Services**
  Data modeling, holistic management of service-related XML data and enterprise (relational) data, and exposure of queries as services.

- **Metadata Services**
  Unified management of metadata related to IOD (data models, data assessment, business glossary, data integration, etc.) and metadata related to SOA (service definition incl. message models, process models, etc.).

- **Content Services**
  Expose (federated) content – leveraging imaging, archiving, records management and other concepts – as a service and provide content centric workflows.

- **Information Integration Services**
  Include business glossary and data analysis functionality. Expose cleansed, transformed, consolidated and/or federated data as services.

- **Master data Services**
  Managing and exposing trusted master data as services.

- **Analysis and Discovery Services**
  Exposing analytic data and insight as services.

2.3.2 The Role of Information Services in SOA

Information services, like all services in SOA, provide reusable units of capability in the broader IT environment. Designing for reuse is an architectural principle that is applied across all levels of IT architecture and the term “service” is often used to express this concept of reusability. But how do “information services” relate to SOA services?

In SOA the term “service” has a very specific context. A Service-Oriented Architecture is an enterprise-scale IT architecture for linking resources on demand. These resources are represented as business-aligned services which can participate and be composed in a value-net, enterprise, or line of business to fulfil business needs. The primary structuring element for SOA applications is a service as distinct from subsystems, systems, or components. An SOA separates out the concerns of the Service Consumers and Service Providers (and Brokers). A Service is a discoverable software resource which has a service description. The service description is available for searching, binding and invocation by a service consumer. The service description implementation is realized through a service provider who delivers quality of service requirements for the service consumer. Services can be governed by declarative policies. These concepts are reflected in the SOA Solution Stack (see Figure 2).

To understand the relationship of information services to the SOA services shown in the SOA solution stack it is important to understand clearly the relationship of services to service components.

Services provide the formal contracts between the consumer layer and the service provider layer. From the top-down the service layer provides the mapping from the business process to the service implementation. The service layer is responsible for

- identifying the appropriate service provider for the service request from the consumer
- locating the service implementation accumulating other requirements such including access control
- binding to the service implementation and invoking the requested service operation

Service Components provide the implementation layer for services. From the bottom-up the service layer exposes interfaces from the service component layer. There is a many-to-many relationship between service interfaces and service components as shown in Figure 4.

- one service component may be exposed into different formats by different service interfaces (see Figure 4, ❶)
- multiple service components can be combined in the component layer and exposed in a single service interface (Figure 4, ❷)
- composite services can be built in the service layer to combine multiple existing published services into a higher value service. (Figure 4, ❸)
- service components can be clustered into subsystems e.g. Accounting components vs OrderManagement components
- service components can be decomposed further into reusable composite parts making up the component implementation (see Figure 5)

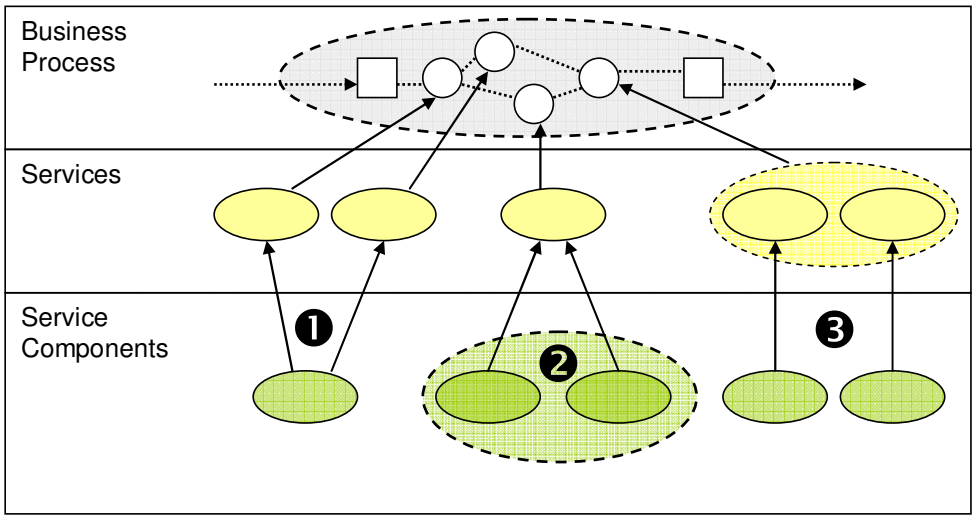

Figure 4: Services and Service Components

There are many implementation choices available for realizing service components. While a service component can be developed entirely as new code, it is also very common to use the service component layer as a wrapper/connector to existing components in operational systems. Figure 5 illustrates this approach and some of the typical combinations relating the services and service components from Figure 4 to the underlying application and information services.

- **Information services implement and are equivalent to SOA services** (see Figure 5, ❶)

  An information service can provide the full implementation of a service component which in turn can provide the full implementation of the service. For this case the information service is equivalent to

the SOA service e.g. the retrieveFullCustomerDetail information service from an MDM system can be exposed directly as a business service at the SOA service layer.

- **Information services partially implement a service** (Figure 5, ❷)

  An information service can provide the full implementation of a service component which is then wrapped with other service components to implement the service. For this case the information service is providing the implementation of only a part of the service.

- **Information services partially implement a service component** (Figure 5, ❸)

  An information service can provide a partial implementation of the service component. For example, when a business process invokes the submitOrder service the implementation could be a collaboration between an application service inserting the new order into the Orders system and an information service providing the geospatial analysis required to allocate order fulfillment to the distribution center nearest the delivery address for the order.

- **Reuse of information services by multiple components** (Figure 5, ❹)

  An information service can be reused in more than one service component, and consequently, in more than one service.

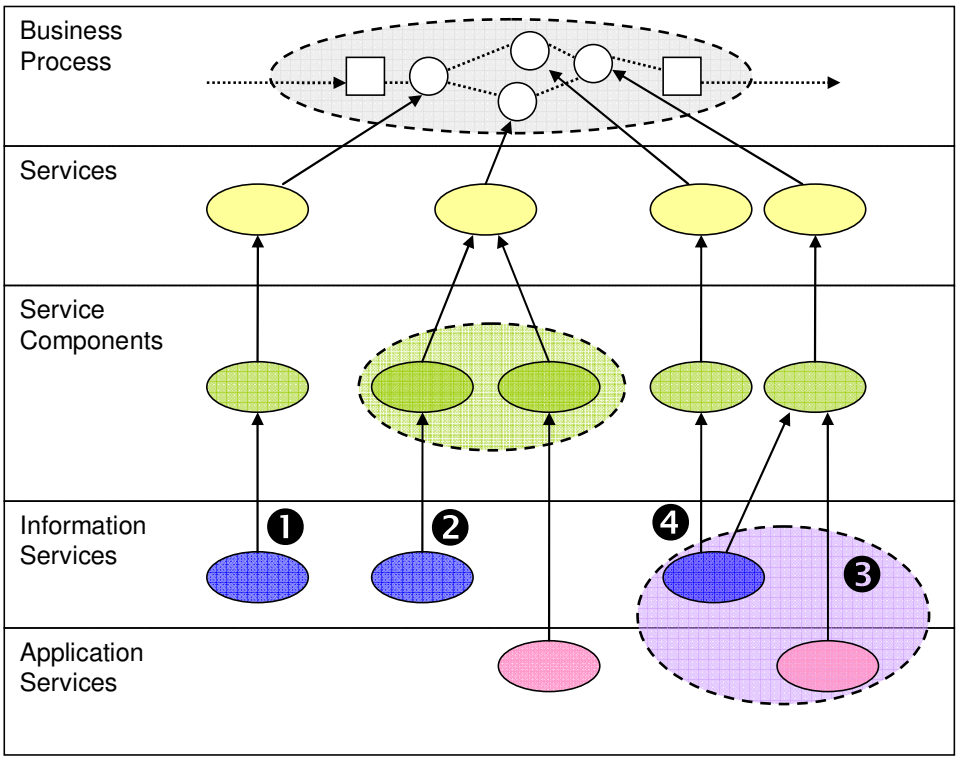

Figure 5: Positioning Information Services in the SOA Solution stack

Each layer shown in Figure 5 has the capability of implementing rules and “business” logic. A clear understanding of the placement of both technical and business rules into the appropriate architectural layers is critical to achieving an understandable and extensible SOA solution. While the classification of rules and logic into layers can at times be subjective, the following principles provide a reference point.

- **Business Process Logic** should be limited to the workflow and decision points guiding an activity through the process. Defining these rules is the responsibility of the business analyst who will

typically be using a business modeling toolset. For example, a rule which says “If a customer is placing an order with value higher than $1000, and the customer has a credit history score lower than 90%, then inventory may be allocated but payment processing must be complete before the order is approved for provisioning”. If services have been well designed, the business process should have no visibility or understanding of the underlying operational systems.

- **Service Layer Logic** should be limited to discovery and allocation of the service request to the appropriate service component(s). These are mostly technical rules such as appending access control tokens, logging, error handling, and making routing and binding decisions.
- **Service Component Logic** is predominantly the rules needed to integrate operational systems, applications and databases i.e. understanding how requests are mapped to operational systems. This includes the technical mapping and binding rules, but equally important are the dependency rules for combining underlying operational capabilities into understandable service interfaces e.g. “To modify the customer residential address first call the address standardization and validation service in the Logistics System, then pass the standardized address to the changeAddress API on the Customer Care system which will update to the new address and notify all dependent systems of the change”.
- **Information Service Logic** should be limited to data structure and validation rules. For example, “All specified currency types on a funds transfer must be defined in the CURRENCY table”, or “Order history for a customer can be found by joining orders < 90 days old from the Order system with orders > 90 days old from the data warehouse using the customer reference number as the shared key”, or “By comparing the names, birthdates, address histories and biometric records of two people, the statistical probability that they are the same person is x %”.
- **Application Service Logic** includes all the business and technical rules already encapsulated into the existing application. These are constrained by the application context boundaries i.e. an insurance claims processing application understands the rules for approving/denying a vehicle accident claim based on the outcome of a related legal case against the claimant for speeding, but it does not understand the rules and dependencies that govern the process for the legal case.

3. SOMA Overview

SOMA is an end to end methodology for SOA. It includes phases for service identification, specification, realization, implementation and deployment. Figure 6 shows the various phases and relationship to GS method and Rational unified process (see Reference [1] for further details on SOMA).

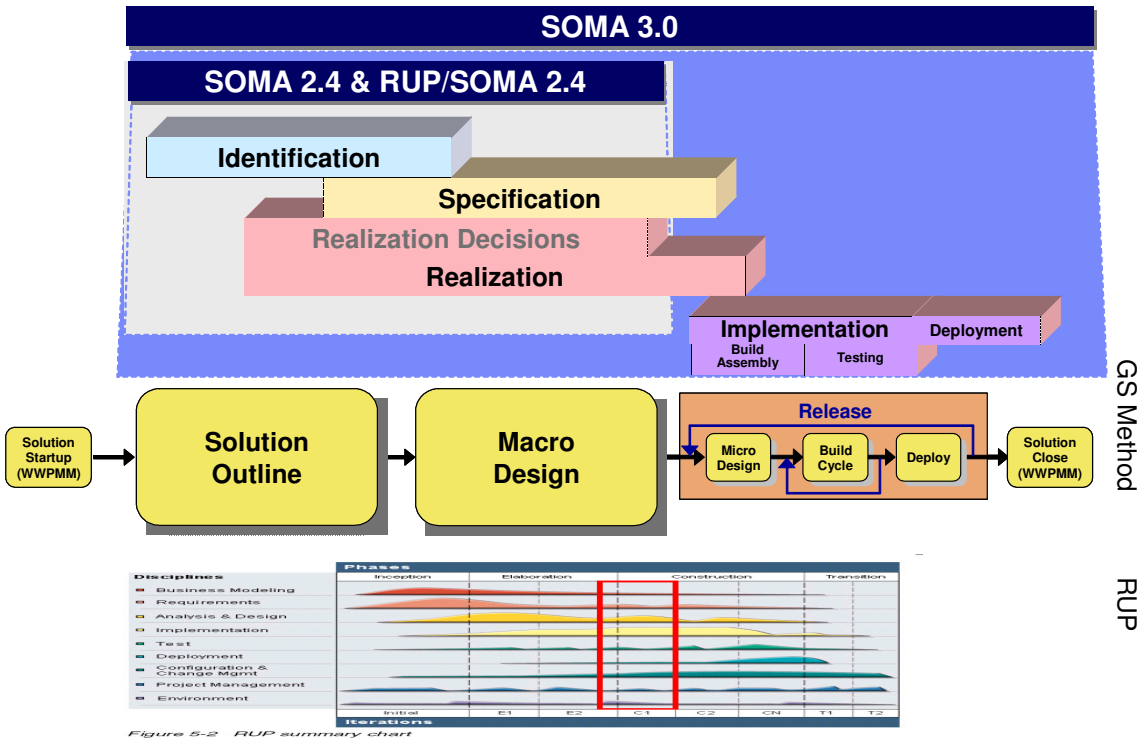

Figure 6: SOMA an end to end SOA Method

The key “building blocks” for SOMA method are role, task and work product and are shown in Figure 7

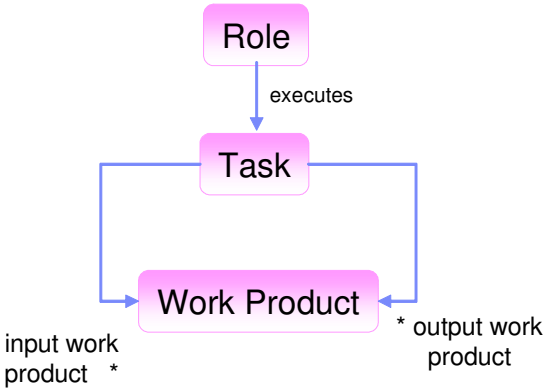

Figure 7: SOMA 3.0 Method Building Blocks

A **role** defines a set of related skills, competencies, and responsibilities of an individual or individuals. Examples for a role are an analyst (and more specifically a business analyst, business designer, business modeler, etc.), a developer (and more specifically an architect, designer, QA analyst, etc.).

A **task** is performed by one or more role(s). To perform a task one or more work products are needed. A task generates one or more output work products.

A **work product** is the generated output of each task and optionally also the input. For example, this technique paper defines that a conceptual data model specification is a work product for the corresponding task.

Related tasks are categorized by **discipline**. There are eight disciplines in SOMA 3.0 as shown in the following Figure 8.

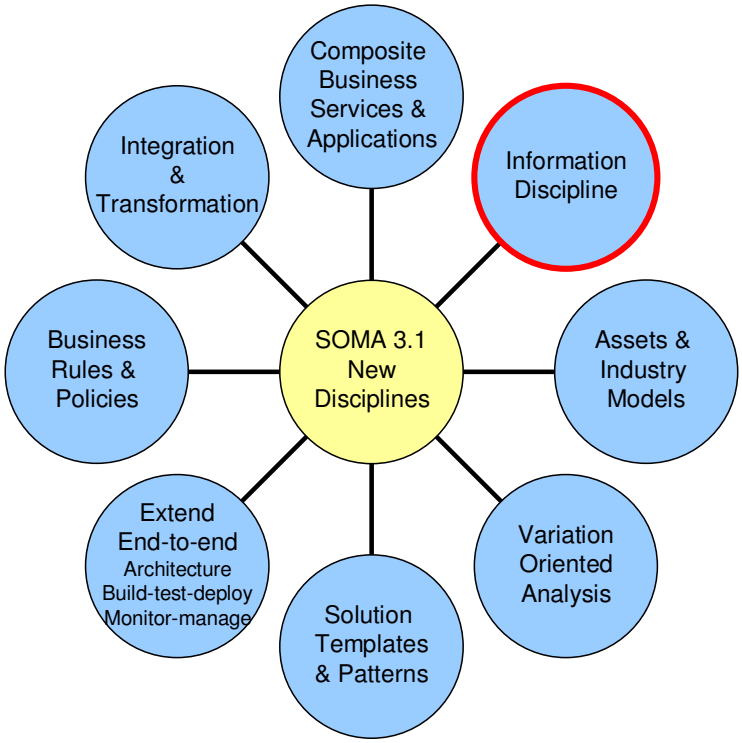

Figure 8: SOMA 3.0 Disciplines

Each discipline is a collection of tasks that are related to a major area of concern as shown in Figure 9.

Separating these tasks into separate disciplines makes the tasks easier to comprehend and to better understand which tasks are related to a certain area of interest.

A **capability pattern** is a special process that describes a reusable cluster of tasks in common process areas. Capability patterns express and communicate process knowledge for a key area of interest such as a discipline and can be directly used by a process practitioner to guide his/her work. They are used as building blocks to assemble delivery processes. Larger capability patterns ensure optimal reuse and application of the key practices they express. Each phase is composed of capability patterns.

A **phase** has multiple activities and tasks. A phase could be extended to support a hybrid SOA solution by selecting necessary solution templates during method adoption workshop and instantiating the solution templates during each phase but only including tasks from the solution template element defined for that phase for a given solution template.

An **activity** has multiple tasks.

A **solution template** is a mechanism for extending SOMA to incorporate and combine multiple strategies and approaches for building an SOA solution using well-established methods, techniques and mechanisms.

A **solution template element** is a mechanism to group tasks in a solution template by SOMA phases. All tasks and capability patterns in a solution template for a SOMA phase will be grouped under a solution template element. The notion of a solution template element is implemented using capability pattern to facilitate dragging and dropping of tasks by phase during method adoption workshop**.**

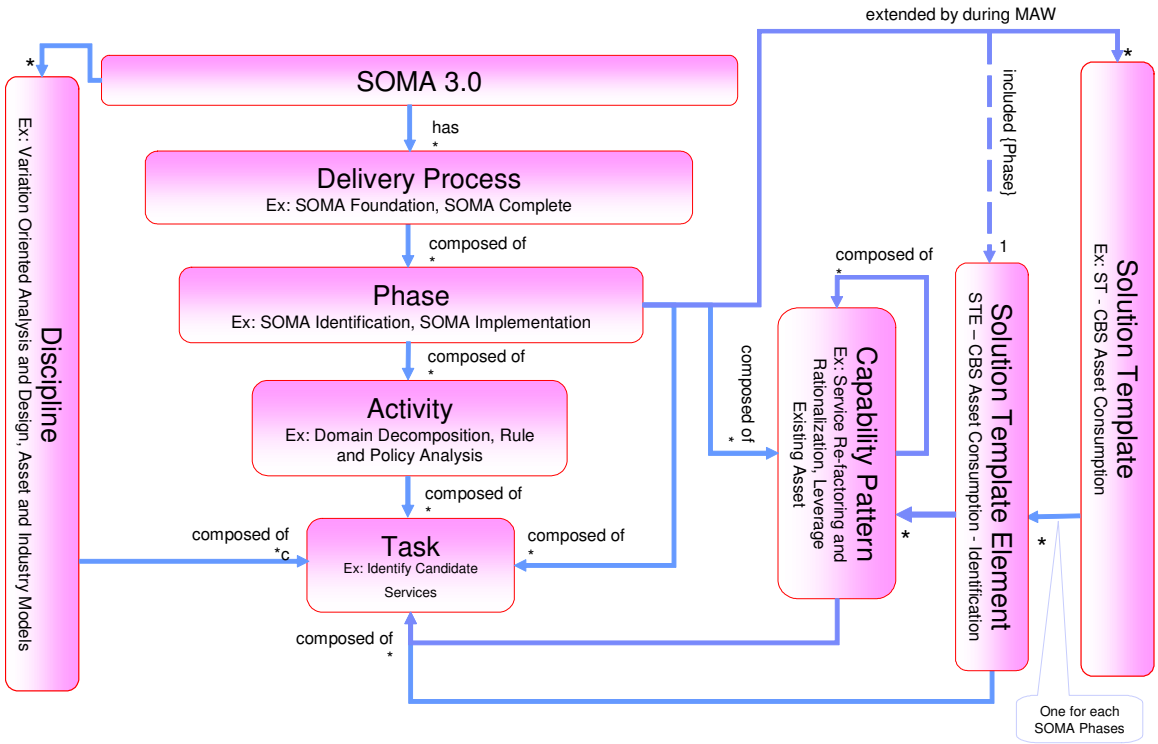

Figure 9: SOMA 3 Method Structure Concepts

The focus of this paper is on the Information Discipline. The information discipline tasks, techniques, and deliverables outlined in this paper are not separate or distinct from the other activities comprising the various phases of SOMA. They are tightly integrated and interdependent. This paper outlines each information architecture activity and discusses how it is achieved and how it integrates with related SOMA activities. Like any comprehensive activity, not all steps will be followed in every project. By gaining a broad and deep understanding of all the activities and their relationships in SOMA, the project team can determine which parts of the method will be adopted for a specific project.

4. SOMA Identification Phase

The SOMA identification phase is used to develop a list of potential services for the SOA project. Three complementary approaches are used:

- Domain decomposition, a top-down business-driven view, analyzes key business aspects to identify services.
- Goal service modeling establishes alignment between services and business goals.
- Existing asset analysis identifies functionality that can be exposed as services using multiple asset sources such as existing systems and industry models.
- Service litmus test to select services to be exposed from a list of candidate services.

During SOMA identification the emphasis is on identifying process flows, potential services and components for the solution. Requirements during this phase are driven from the business goals and pain points with emphasis on business scenarios and business process analysis. The goal of the existing asset analysis is to identify services that have potential to be used by various business processes. In addition, it helps to identify existing functionality available to support activities within the business process. A detailed analysis and a detailed design is applied later in the specification and realization phases of SOMA. Goal service modeling helps to focus on the business processes which support the business goals, as well as to identify services which would have been missed in a top down or bottom-up analysis. Since the next SOMA specification phase is a time consuming activity, it is essential that only those services that meet certain criteria such as business alignment, re-usability, etc are selected. The service litmus test criteria are used for services to be considered for service specification from a list of candidate services.

During the SOMA identification phase, existing IBM assets (e.g., IBM Industry Models and in particular IAA for the insurance industry and IFW for banking industry) and client assets are leveraged. For instance, if existing business process models of these assets are similar to the required / to-be business process under study, then we can benefit from all the process analysis done with respect to events, tasks descriptions, etc. In effect, these assets act as accelerators. There are several information architecture techniques which can be applied before and during the service identification phase to improve the quality, consistency and efficiency of the other tasks in this phase.

The information discipline activities for the identification phase are the following:

- **Create a Business Glossary (Section 4.1)**

  A business glossary defines the language of the business and by extension the language of the project. Capturing and documenting an agreed business vocabulary reduces confusion and misunderstandings across the different actors and organizations working on the SOA project. The business glossary is a reference against which all project documentation should comply. The business glossary is an enterprise architecture asset for the organization. It is continually refined by the SOA project and other IT projects in the organization.

- **Create a Conceptual Data Model (Section 4.2)**

  Where the business glossary defines the individual terms describing and organization's information, the conceptual data model brings structure and relationships to those terms. Providing a clear reference for how the business organizes its information, and then following that structure throughout the analysis and design process is essential to achieving successful integration of processes and services in the SOA project. The conceptual data model is scoped in alignment with the SOA project and can be based on existing industry standard models where appropriate for the organization.

- **Create a System Context Diagram (Section 4.3)**

  Context diagrams give a clear and simple representation of the key data and function included in a new system, and how that data and function interacts with external systems. Creating such a diagram early in the SOA project establishes a clear understanding of the system boundaries for all actors working on the analysis and design of the SOA solution. The diagram is refined throughout the project and is especially useful for resolving issues of functional scope during the project.

- **Create a Catalog of Data Sources (Section 4.4)**

  The catalog of data sources provides an understanding of the physical data systems and their characteristics. It provides a list of existing assets which can potentially provide service implementations directly, as well as providing reference information used later in the project to make suitable implementation choices for all services. The catalog of data sources is initially created in the SOMA identification phase but is refined throughout the project.

- **Identify Information Services (Section 4.5)**

  The overall SOMA 3.1 methodology includes the categorization of services as part of the SOMA identification phase. Typically, we need early in the SOMA engagement a basis for planning scope and investment through the remainder of the project. Although it is premature at this point to make final decisions on how to realize a service, we can start to categorize service and to identify the information services. This decision is then revisted in the future SOMA phases.

- **Identify Information Criteria for the Service Litmus Test (Section 4.5)**

  The service litmus test governs service exposure decisions. Candidate services are evaluated against criteria such as business alignment and composability to determine if they should be implemented and published for general use by service consumers. This is an iterative process, started in the SOMA identification phase and continued through the life of the project. Verifying that a candidate service is aligned with the conceptual data model for the business, and that if implemented it can deliver accurate and reliable results are essential criteria against which all candidate services must be evaluated before exposure decisions are taken.

While the items listed above define the formal information discipline activities undertaken during the SOMA identification phase they are by no means the only activities that a data architect should be involved in during this part of the project. SOA projects are widely variable and iterative by nature. The SOA architect needs to take a flexible and opportunistic approach to harvesting and/or creating services that will support the business processes currently in scope for his project as well as establishing a foundation for future projects. The work of the data architect parallels this approach. Part of the work is formalized top-down creation of data models, documentation and designs while other parts involve harvesting existing assets wherever they may be applicable to the SOA project. The data architect also plays a consulting and mentoring role for other practitioners needing to work with business information structures and meaning. This role begins during project definition and requirements gathering and continues through analysis, design, test and deployment of the solution. Involving the data architect in project definition, business requirements and use case definition discussions is beneficial to both the data architect (who gains early insight into the data requirements for the project) and to the other participants who will identify and drive to resolution any contradictions, inconsistencies and lack of clarity around information definitions during business analysis. This may lead the data architect to drill down into technical analysis and detailed data modeling earlier than expected for cases where there are known issues around specific areas of business information.

4.1 Create a Business Glossary

4.1.1 Definition

A **business glossary** (BG), sometimes referred to as a data dictionary, is the artifact that defines the terms and data associated with an initiative. Depending on the extent and type of engagement the business glossary will either define the context of a term within a silo, an information domain, or (preferably) the enterprise wide definition of a term.

It is common for different departments or lines of business (LOB) to have different semantic contexts for what would seem to be the same term. For example, take the element “address”: to distribution department this is likely the “ship to” address, to accounting it is most likely a “bill to” address, to sales and marketing this will likely be a “call on” or “contact” address. This is a very simplified example and is easily dealt with using a name prefix or having three different address fields. Nevertheless, there needs to be a way to document and identify which “type” of address we are dealing with and what each one means.

The business glossary defines the language of the business and by extension the language of the project. Therefore, care needs to be exercised that the terms defined in the business glossary are fully qualified and that specific descriptive definitions are provided. To the extent possible, a definition that applies enterprise-wide should be crafted. Where departments use a term differently, those definitions should be captured and associated with their appropriate contexts (department).

When an organization builds an enterprise-wide business glossary, it may include both semantics and representational definitions for terms. The semantic components focus on creating precise meaning of the terms. Representation definitions include how terms are represented in an IT system such as an integer, string or date format (see data type). Business glossaries are one step along a pathway of creating precise semantic definitions for an organization.

In the SOMA 3 identification phase the business glossary captures terms that surface during any of the SOMA activities in process decomposition, goal service modeling or existing asset analysis. Terms can be related to process activities, goals or can simply consist only of the definitions of the identified individual source data bases. The major point being that there needs to be a formally agreed definition for each element in the business glossary even if initially it only applies to individually identified sources.

4.1.2 Objective

The business glossary is part of the formal contract between the producers and consumers of information across the enterprise. It is the artifact or reference that will allow anyone to determine the meaning, type and context of any term and in particular any business data element used in an initiative. Too often we see significant lack of formal definition around data entities even within a single system. Different interpretations of the same term increase the risks against successful project delivery. There are often embedded business rules that make data inaccurate out of the context in which it is originally used. There can be rules embedded in the programs that use the data or additional reference data needed to give the entity context. These are some of the potential issues that need to be identified.

The lack of such an artifact can contribute to consumers of services and the information they expose saying that they do not trust their information. There can be confusion as to the exact meaning of a data element, or an element is used out of context because the provider and consumer do not agree on exactly what the element or entity really represents, or fails to use it without the necessary supporting semantics.

In the case where the scope of the engagement is an enterprise wide master data management engagement then this business glossary becomes even more important because it will be the artifact that aligns a definition of a term across the enterprise.

The business glossary is a “living document” that will evolve as the initiative progresses through the SOMA identification, specification and realization phases. It will mature and be added to as the initiative progresses.

Data architects develop a business glossary to represent the data elements of an existing data store or a broader data integration project scope. However, it is also valuable to start creating a business glossary as soon as data terms and entities are being discussed in the project. In domain decomposition for example, as soon as we start looking at business processes we will start to talk about business terms applicable and should be capturing or updating these terms into the business glossary.

On the flip side of the above is creating or updating a business glossary for existing data stores. In that context, if a business glossary exists for any of the existing identified sources then it should be leveraged as a starting point. It should be scanned for thoroughness and the definitions reviewed with the appropriate stewards of that data. Typically a business glossary is developed for each and every data source that is involved in a data integration project and is an important artifact in such an undertaking.

4.1.3 Value

Unfortunately, it is all too common that the same term is interpreted in different ways between business and IT experts as well as between various lines of business. The success or failure of a project depends on a commonly agreed understanding of shared terms such as member, customer, supplier, etc. Without a formal and shared definition, this common understanding is not possible.

It is not uncommon for businesses to have redundant copies of seemingly similar or identical data elements, whose context is only valid within the data store in which it resides. Often when this element is exposed to outside consumers, the information loses its context, or worse is becomes misleading or invalid.

Therein lays the fundamental value of a business glossary. It is the contract that aligns the definition of data elements so that its context is meaningful across all consumers of this data.

This artifact should contain not only the agreed upon definition for a data element, but any variations or dependencies associated with that element. This will eventually help drive the conceptual and logical data models. Ultimately this will enable the components and participants in an SOA solution to arrive at an aligned and consistent definition of business information in context.

The business glossary can also benefit the design of operational systems. For example, several tables in an operational system may hold telephone numbers; in developing a business glossary the format of this telephone number field will be consistent.

In some scenarios where a service requires information from several sources, there may be redundant data elements with the same “name” or implied definition when in fact the definitions or contexts are quite different and have meaning only within their particular data store, possibly only when retrieved by their associated program. For these cases the business glossary helps identify and resolve such conflicts by providing one consistent and common business definition and dependencies for each entity across the enterprise. When users from different segments talk about “Customer” or “Revenue” everyone will know exactly what they are referring to.

Possible impacts of not having this work product are:

- Risk of missing fundamental information requirements of the business
- Cost in moving forward with projects where the information requirements are not well understood
- Cost in time expended reworking unclear and/or misunderstood requirements
- Cost in credibility, business error recovery, system rework up to lost business when delivered systems do not match the needs of the business

Possible reasons for not having this work product are:

- The only time this work product is not needed is when the work being done is purely technical in nature, e.g. simply re-architecting a middleware layer

4.1.4 Approach

4.1.4.1 When is a Business Glossary Created?

We have to remember that a business glossary cannot be started early enough and it is not just an exercise initiated during an existing asset analysis. A business glossary can and should be considered as early as possible: even during requirements gathering and project definition. A business glossary is not limited to existing data stores or databases; it also contains the definitions for all business terms used to describe business processes and services in SOA. The earlier the glossary is being developed, the sooner and the more likely the foundation for consistency of terminology throughout the project and the enterprise is accomplished.

There may be an existing business glossary for one or more existing systems or defined by an industry standard. These glossaries will be included during the existing asset analysis. If such a glossary exists, then it merely needs to be reviewed and included for integration into the enterprise business glossary.

The business glossary will be established and developed during the SOMA identification phase. Domain decomposition, existing asset analysis, and goal service modeling are all contributors to, and consumers of, the business glossary. As the project progresses through the specification and realization phases the business glossary will be continue to be updated and refined.

Ultimately the business glossary will be input into the conceptual and logical data modeling activities.

4.1.4.2 Who creates a Business Glossary?

This leads to the discussion of roles necessary to create a proper business glossary. In some organizations there are existing business analysts who understand the business definition of the data in question. In other organizations there are informal “experts” who are the historical, informal stewards of data. In many cases there is lack of formal definitions and dependencies associated with most of a company’s data.

This/the owner(s) of the business glossary will vary from organization to organization and even within silos in the same organization. Information domains and their ownership should be defined within an enterprise data/IT governance structure, and these same information domains and ownership hierarchies can be applied to the business glossary. If such governance structures are not already in place then it is likely that the data architect will play this stewardship role for the duration of the SOA project with a view to identifying the long term ownership strategy by project end. It is highly recommended that any project contain or utilize a

governance process. If such a process does not already exist within a company then the project should include implementing such a governance process.

Typically, there will be a business analyst and/or data steward identified for each information domain or perhaps at an even more granular level for each operational data source, domain, or entity involved in the solution. In some instances that could be the same person while in other cases it may be individuals from the LOB or segment that "owns" the data involved. It is even possible that any one source may have multiple "stewards" for the data, each with expertise in particular subject matter. . It is possible that there will be more than one steward for any one term, take for example "customer type". There may be customers from marketing, finance etc and set of customer related data may have a steward from that department or functional area. In the example above, "Address" may require adding some a qualifier or extending the data structure to identify the category of "address". The data architect will be helpful in suggesting ways to logically associate and identify the type of "address" being asked for. This is just one more reason to have a mixture of skill sets helping to drive these definitions. If left solely to a business analyst then the resulting view may not meet the needs of all the dependent downstream SOMA activities in the specification and realization phases

If this is not the case then an appropriate subject matter expert (SME) must be identified and assigned the role of "steward" for any particular source or portion of a source. This should be someone who understands the business use of all the business terms identified as candidates for inclusion in the project. This may or may not be a single individual depending on the amount of identified source data and the individual's knowledge of that data. They should also know or learn the dependencies and relationships between all these entities.

It is also often very helpful to have a data architect available and involved in this process as they typically understand the physical constraints and structural aspects of the data sources. They are also helpful in helping to determine ways to structure the relationships and dependencies.

In the case where the organization is starting and no business glossary exists for any of the identified information this is the more difficult exercise. The notion of the role of SME, business analyst and steward are all valid, however it means that these individuals have the arduous task of seeking out the business "experts" who have the actual business definition of the terms and getting agreement from everyone involved on the final definition of that term. This is not a task to be taken lightly and often will take a tremendous amount of time and effort to achieve. However, as mentioned, this is the contractual agreement, across the enterprise, of the "real" definition of that element or term. This will establish the common language upon which every consumer of data will rely. The mechanics of achieving this will vary from company to company and project to project.

4.1.4.3 How is a Business Glossary created?

The following points will be covered, in detail, in the practical aspects of creating a business glossary.

- Gathering information sources
- Extracting business terms
- Building a glossary
- Classifying terms

If an existing business glossary exists for any identified data source involved in the SOA project then the project team will have an easier time reviewing and preparing the proper business glossary for the scope of that information domain.

There are typically two cases to consider. If a business glossary does not exist, how does the project team go about creating one? The same question applies when a business glossary must be created and individual ones, in some form, exist for their respective sources or scope.

The most valuable aspect to the business glossary is the hardest to obtain, that is an agreed upon universal definition. Typically that is done in any number of ways. It can be done by interviewing subject matter experts or through facilitated sessions or questionnaires. Depending on how far the scope of the term reaches often correlates to the difficulty in getting agreement on the definition. If a term is used by a single LOB then an SME or business analyst may already have the "agreed" upon definition. In the case where a term is going to be used by the entire enterprise then reaching agreement on the definition can become measurably harder and might require first gathering the individual definitions and then have the individual SME meet and try and come to a common understanding and definition. It might be that what was initially a single term becomes one or more terms in the enterprise to satisfy all contexts.

One simple aspect of the business glossary that is often overlooked and the easiest information to obtain is the physical aspects of terms (i.e. data type definitions and constraints etc). It is worth noting that depending on the term, the physical structure may not be a mandatory aspect to collect and may be defined later or in the specification phase as data models are developed. However it is worth maintaining that information in the business glossary even if it is added at a later date. This can be done manually or using some of the automated information analysis tools available on the market, which will actually help discover the physical characteristics of terms and can be very helpful in assessing the existing quality of the information. This will be discussed in greater detail in the section on creating a catalogue of data sources (see Sect. 5.3).

It is no small or simple task to achieve agreement among parties that have their own history, ideas and contexts of what an element means to them. Take for example the rather simple notion of a "Customer". That definition will have a different context depending on the department or LOB using the element. For marketing a customer may be an individual or business they contact to send material to. For sales it may mean the individual they call upon to sell to. To distribution it may mean the entity that they ship to and so on for the different functional areas. It is also quite likely that to some a "Customer" is an individual and others it is a corporation or business. Often times, there may not be one single definition of a "Customer" but it may be extended and given a contextual definition with the notion of a "Customer Type".

The bottom line is that a consensus will have to be reached that will deliver one agreed upon definition that will be valid and used across the enterprise or at least for the scope of the SOA project.

How does the steward or steward team go about achieving this agreed upon definition? There is no one simple answer for this question. All parties must reach an agreed upon definition for every term and entity in question. It may be through facilitated meetings and negotiations or subject matter experts proposing definitions and socializing them. What is important is that at the end of the day there is ONE consistent definition for every term, or that complex terms are able to be defined and given context

.The final aspect of the process of creating a business glossary is the verification and validation aspect.

It firmly grounds the complete business model and resulting solutions in the business domain. Some of the linkages and traceabilities that build on business language analysis include:

Completion Criteria: A key strength of the business glossary is that it is relatively deterministic. There are tests that can be applied to determine that the business glossary terms list is complete and accurate for the business domain. These tests include:

All key client personnel have validated completeness of the sources used to build business language definitions.

Terminology is converging – fewer and fewer new business terms are discovered in either written sources or conversations with domain experts or as new sources are introduced.

Terms have been completely classified by business concepts.

New concepts have been incorporated into the classification scheme.

Quality, a business person should be able to validate that the terms are meaningful to the business and are properly classified and have the proper context.

4.1.5 Deliverable

The artifact produced when creating a business glossary can take many forms. It can vary from a structured written document, to a database, to a specialized commercially available tool. The form is not as important as the actual content.

In its simplest form it is a set of metadata that contains definitions and representations of business terms. A business glossary should, at a minimum contain the following information.

- An agreed upon definition of the business terms
- Ownership identification of the term
- Business rules necessary to give the data context
- Special constraints or annotations
- Hierarchical relationships between terms, domains and/or information domains

Optionally and depending on the term and context some of the following attributes should be captured as well. If not initially in this phase then it would be helpful to keep the business glossary up to date as the project progresses into later phases.

- Physical characteristics of the terms
- Integrity constraints
- General element structure
- Schema objects
- Triggers or event mechanisms

The GS method work product related to this task is APP 301 (Classified Business Terms).

4.1.6 Dependencies

As stated before, this task does not rely on any input from any other task. However, the development of a business glossary can be significantly accelerated if existing glossaries exist or industry standards are used.

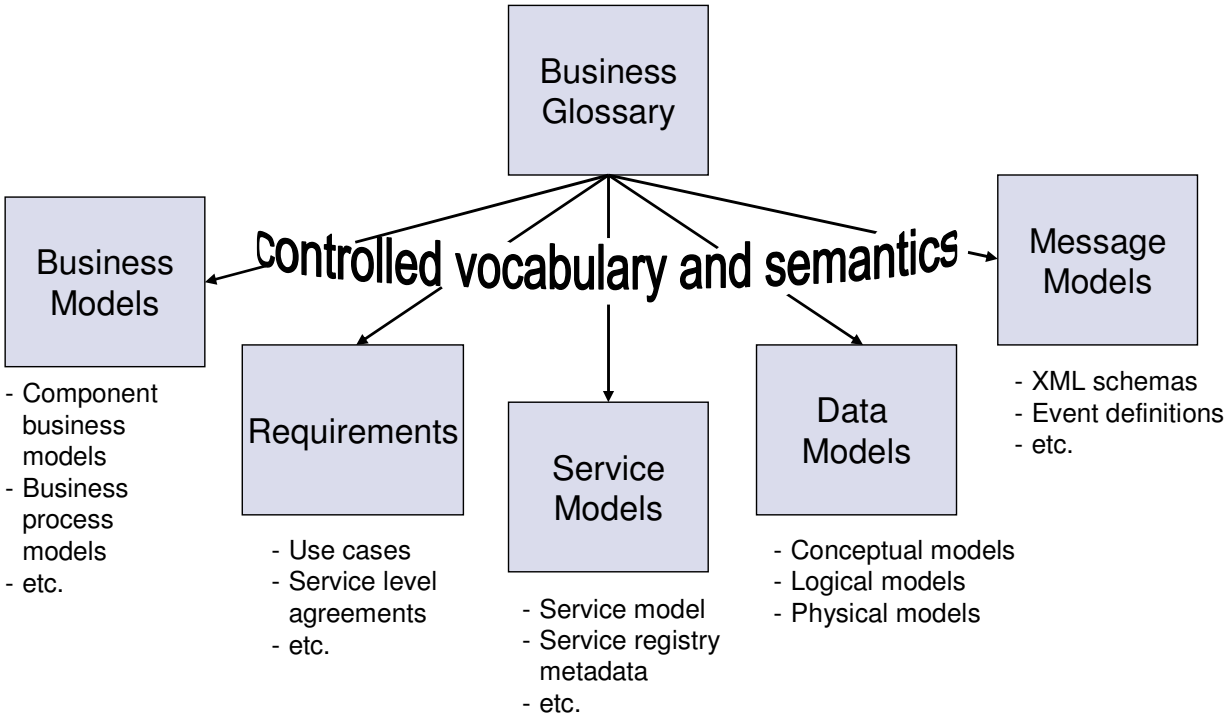

Figure 10: Dependencies of the Business Glossary

The major purpose of the business glossary is to capture and consistently define the vocabulary and its taxonomy of the project which can start during the requirement analysis.

The main output of this task is the business glossary specification which is valuable to almost all other tasks in SOMA to guarantee a consistent use of the terms and the context in which they are used.

4.1.7 Example

An example of one possible artifact from a well defined business glossary around the concept of an account opening procedure might look like the following. The hierarchy may be region, Account, Account Type ('Money Market', Checking etc). Depending on the maturity of the existing definitions, at this point in the identification phase there may or may not be full definitions and values for all terms. This will mature as the project matures through the SOMA phases. As more is learned about the business terms more information should be updated in the Business Glossary.

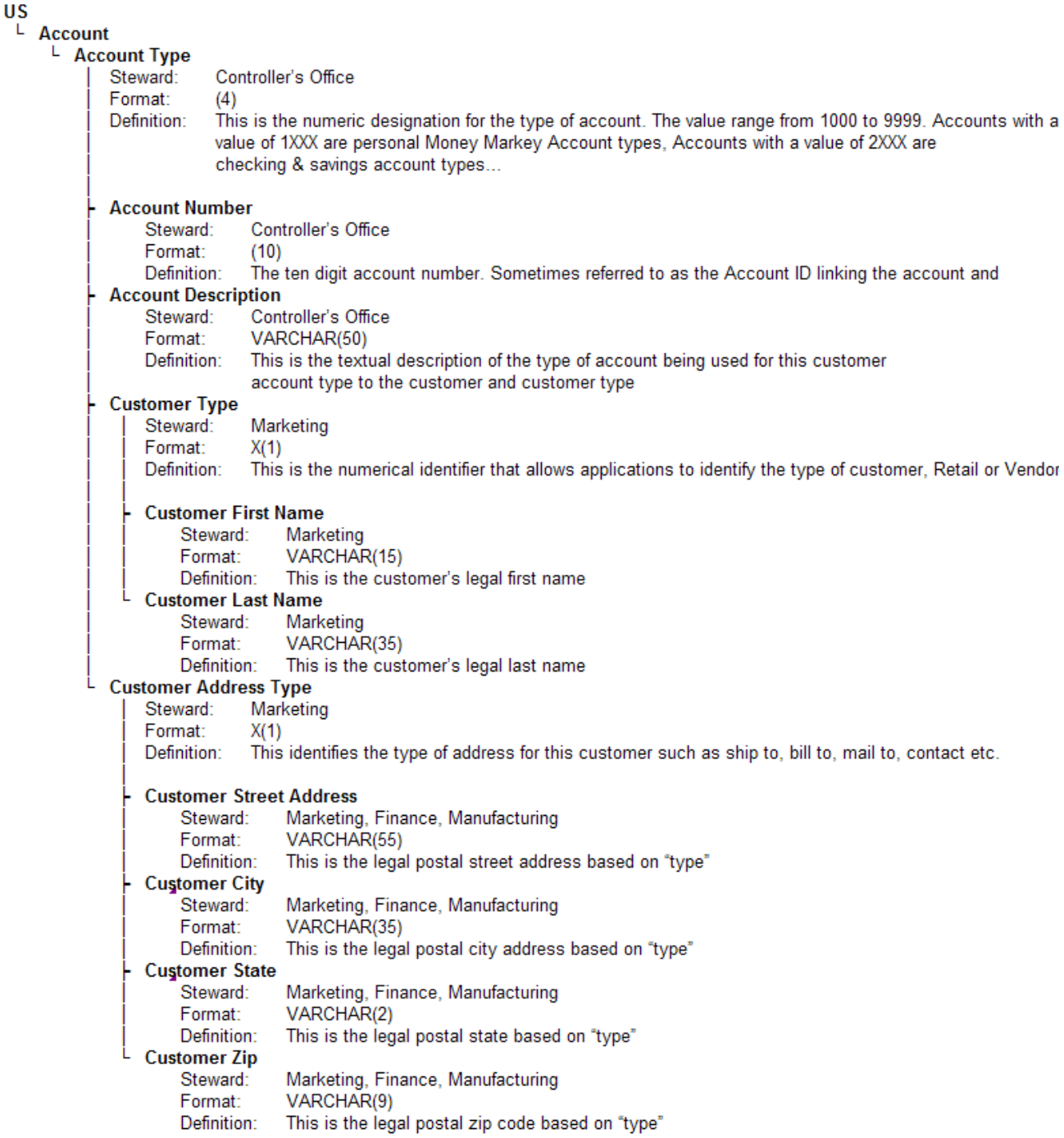

Figure 11: Example of a Business Glossary Definition

The above figure shows some possible business terms used in an "Account" process.

4.1.8 Guidance

An architect should consider the following normative guidance when developing a business glossary:

- Start to document the definition of the vocabulary (terms) and the related taxonomy as early as possible and continue throughout the project to refine.
- Leverage where possible metadata software solutions – such as IBM's WebSphere Business Glossary (see Reference [5]) – that help to align the vocabulary from the business glossary with other artifacts such as the data model.

4.2 Create a Conceptual Data Model

4.2.1 Definition

The **conceptual data model (CDM)** is a canonical data model at the highest level of abstraction. A canonical model is one that is based on standardized or generic entities and can exist at both the conceptual and logical levels. This work product represents the strategic information requirements of the enterprise within the scope of the SOA project. Ideally, it is a subset of a full canonical conceptual data model at the conceptual level. However, if nothing already exists in the organization, it should be built as a step towards one.

Entities are the fundamental building block of the conceptual data model. An entity represents a grouping of "attributes" around a single, key unifying information concept. Each attribute represents a single information concept at the most granular level. If an attribute is an atom then an entity is a molecule. Both attributes and entities are based on and aligned with aligned with the business glossary regarding terminology and semantic definitions. The conceptual data model is created during the SOMA identification phase and includes only entities.

While in the conceptual data model, a few major attributes may be revealed for illustration purposes, most or even all of the attributes that comprise each entity are added later as part of logical data modeling during the specification phase.

The entities included in the conceptual data model are those which are needed by the candidate services (and implicitly which may be specified in the system context diagram, see Sect. 4.3). It doesn't matter whether an entity is needed by only one candidate service or by several candidate services. If the entity is needed by any candidate service, it is included in the conceptual data model. This is because at this stage, issues of data ownership and sharing have not been resolved. If any candidate service needs an entity for whatever reason, it must be accounted for in the canonical model.

In the conceptual data model, an entity will have "relationships" to other entities based on the business relationships implicit in the entity's information content, i.e. in its attributes. Even though at the conceptual data model level, attributes are not modeled, the nature of an entity's information is implicit in its definition and business purpose. For example, an Organization entity is expected to have information about the nature of the business, the legal aspects of the business, key personnel, etc. An address entity has information about its street location, the nature of the address, etc. A customer can have many addresses for different

purposes but can a customer be at more than one address? For a household of individual customers, the answer may be yes, but for a business location, the answer may be no. If a customer can have many addresses, and an address can have many customers, it is a “many-to-many” relationship. If a customer can have many addresses but that address can have only one customer, it is a “one-to-many” relationship.

For SOA projects spanning multiple business areas in moderate to large enterprises the conceptual data model can be subdivided into information domains – which are sometimes called subject areas. Information domains are often based on entity affinities. This means those entities that have a high number of relationships. For example a customer information domain might include: customer, address, orders, materials, shipments, invoices, and payments. A materials information domain might include: materials, bills of materials, plants, warehouses, deliveries, shipments and inventory. These information domains have some entities in common. However, they are not necessarily the same as the business domains identified in business domain decomposition which may have been initially identified in component business modeling.

4.2.2 Objective

The development of a conceptual data model has several purposes within the SOA project. The main reasons for developing a conceptual data model in the SOA project are:

- To provide a strategic overview and understanding of the major high-level groups of information needed to meet the information needs of the candidate services and the related business within the scope of the SOA project.
- To starting point and boundaries for subsequent information analysis, data modeling, and information design activities. It sets the baseline information context in which all services will operate for the project across all participating lines of business.
- To provide high-level planning constructs with respect to the information needs of the SOA project and to align those with enterprise information architecture strategy.
- To enable and support a proper alignment between the key information requirements of the business and its partners, with the goals and objectives of the enterprise.
- To enable business process and IT architects to explore the information-based constraints on the business and the opportunities for, and the implications of, change.
- To provide a useful vehicle for information governance regarding responsibilities for information creation, update, deletion, distribution and use.

The conceptual data model typically will be coarse-grained and is intended to show the broad set of entities and relationships under consideration. It is not intended to be granular enough in perspective for service specification, but rather to provide a high level view of the information universe under consideration.

4.2.3 Value

This work product enables the SOA team to demonstrate an understanding of the structure and content of the customer’s information requirements. This will provide:

- A basis for defining potential opportunities for information sharing between candidate services in support of asset reusability across lines of business in the enterprise.

- A means for reducing the risk of missing important information needs or failing to align the SOA management of information with the wider information needs and strategy for the enterprise and assuring that business goals and objectives will be properly supported.
- A starting point for representing data entities in the process models, use cases and logical data models needed to develop the SOA solution.
- A starting point for analyzing information sharing requirements and information quality analysis, across the business units within the enterprise including the impacts of future changes on the SOA solution. This means that the accuracy of the estimation of subsequent projects is less likely to be affected and there is less potential for delay in follow-on projects.

This work product may not be needed for cases where:

- An conceptual data model covering the functional domains for the SOA project already exists.
- The scope of the engagement is limited such that detailed analysis of service interfaces will not be performed. This might be the case when the focus of the SOA project is deployment of infrastructure or integration of pre-existing or outsourced services.

4.2.4 Approach

The conceptual data model can be constructed using the following activities which to a degree and be performed iteratively and in parallel:

Step 1 - Identify Potential Information Domain

Step 2 - Identify Existing Metadata

Step 3 - Identify Key Entities and their Relationships

Step 4 - Conduct Workshops with the Client

4.2.4.1 Step 1 - Identify Potential Information Domain

The identification of information domains within the conceptual data model can mean segmenting the conceptual data model into logical entity groupings that are meaningful to the business. In an SOA project using SOMA, a first cut at these information domains can be derived from the business domains used in the functional area analysis. The data architect will collaborate with the SOA business analyst and the SOA business process architect to define the domains and functional areas for both the data model and process model in parallel. This approach is more efficient and more accurate than approaching data and process modeling separately.

Functional area analysis uses the partitioning of the business, such as CBM model, as a starting point or input. Business domains are identified (e.g. from CBM competencies) and further decomposed into sub-domains and ultimately into functional areas (e.g. using CBM components as input). SOMA uses functional areas to categorize candidate services in the service portfolio into a service hierarchy, making further analysis of candidate services more manageable. Functional areas also suggest potential subsystem boundaries used for service component identification.

The information domains correspond to the business domains found in functional area analysis. The conceptual data model is developed as a set of related and overlapping information domains. Each information domain contains its entities, entities shared with other information domains and the main relationships linking them. A shared entity can be shown in more than one information domain.

A high level information domain showing only the major entities across all information domains or just listing the information domains with their relationships may be developed to provide a single “big picture” view. Normally two levels of detail should be sufficient in the conceptual data model. However, it may be necessary to further subdivide an information domain in order to make unusually complex information domains more understandable.

Tool selection for developing the conceptual data model will be dependent on each customer. If possible, use tools that maintain links from the business glossary into the conceptual data model and eventually into other models (e.g. business process models, service model, logical data model, common message model, etc.).

4.2.4.2 Step 2 - Identify Existing Metadata

SOA projects are very often tasked with integrating existing systems in new ways. There are many possible sources of metadata that can greatly reduce the time spent building the conceptual data model and improving its quality. These include:

- **Business Glossary**

  The business glossary should always be treated as an integral contributor to the conceptual data model. The business glossary and the conceptual data model are interdependent and each provides input and clarification to the other.

- **Enterprise Data Model**

  An enterprise data model may already exist. However, it is not likely to be in a state that renders the conceptual data model redundant. If this is the case, the conceptual data model becomes a building block for developing or refining an enterprise data model. In either case, the context under which an enterprise data model was developed must be understood and must not be just blindly followed. In this phase the focus for the conceptual data model developed in SOA is accounting for all of candidate services’ information requirements. In the next phase as the logical data model and canonical message model are developed, the focus will shift to information ownership and integration, that is, providing a shared cross-domain view of core data entities that will be created, updated, exposed and shared through SOA services.

- **Application Data Models**

  Existing application system documentation including the conceptual, logical and physical models used to build and maintain the applications data stores. The conceptual data model should be built based on a business view of the data and not be heavily influenced by the existing operational data systems. However, the operational systems do contain enormous amounts of detail regarding how business data is defined and managed in the existing environment and having access to this information can greatly reduce the effort of defining the SOA models and also provide early validation feedback about decisions being made.

- **Existing Data Interchange Formats**

  The enterprise may have existing data interchange formats for sending and receiving files and messages with internal application systems and external partners. These formats can provide valuable reference information when building the conceptual data model as they often present the data structures and relationships in the way best understood by the target partner in the exchange.

- **Existing Data Integration Systems**

  If the organization has systems and processes for managing data cleansing, data replication, synchronization, data transformation through extract-transform-load (ETL), data consolidation, and data federation to support data warehouses, operational data stores, distributed data sets, master data systems, and other analytical data systems, then much of the analysis and design work done in

building these systems should be reusable in the SOA project. This is true at the conceptual level needed to build the conceptual data model as well as at the more detailed data modeling tasks in later phases later.

- **Industry Standards and Models**

  Industry standard data models, if available, for the customer's industry, can provide a strong possible baseline for the conceptual data model. SOA is fundamentally about easing integration and adherence to a common set of widely accepted, standard services. This is a practical way to assist this. The customer may have good reasons why his business should deviate from an industry standard model but in each case these reasons should be challenged and documented during the development of the customer's own conceptual data model and ensuing data models developed later in the project. A caveat regarding industry standard models is these models are often developed by committees of data modelers from several leading companies in the industry (e.g. ARTS for the retail industry). Such models, in an effort to incorporate many points of view can become very complex. Using them can become an exercise in deciding what is not relevant for a specific customer. Such models can be used in tandem with IBM's Cross Industry Data Model (see Reference [8]). This model includes entities that are common to many industries. As such it provides a top-down framework for navigating the much more detailed industry model.

4.2.4.3 Step 3 - Identify Key Entities and their Relationships

An entity is a concept that represents a real world business object that can be either tangible or intangible. An attribute is concept that represents real world business objects at their most granular level. An entity is a collection of those attributes that make business sense when grouped with the entity. Entities are made up of attributes. In the Customer entity, Customer Name, Customer Industry and Relationship Status are examples of attributes.

The identification of key entities involves the following tasks:

- Identify candidate entities needed to support the functionality embodied in the candidate services. For example, sales order processing likely needs the customer, product and inventory entities as inputs and creates the sales order as an output.
- Compare these candidate entities to the metadata identified in step 2 (see Sect. 4.2.4.2). If an existing application or standard industry model contains entities, not included in the list of candidate entities, identify the underlying business reason. Are they most likely related to business components for which candidate services have not been yet identified? Are they related to business components that are not pertinent to the client's business process model? Likewise, if there are candidate entities that do not appear in the existing metadata, identify the underlying business reason. Do these candidate entities relate to candidate services that are new to the industry? In summary it is important that the business reasons for gaps between the candidate services and the metadata be analyzed.
- For each of the candidate entities, identify that entity's relationships to the other candidate entities. This refers the structural business relationships. This is best done by describing the nature of a potential relationship to see if it makes sense. For example, it makes sense to say "a customer can have many sales orders" but not, "a sales order can have many customers". Likewise, "a single sales order can include many materials" and "a material can be included in many sales orders" It does not, however, make sense to say "a customer can have many materials and a material can have many customers". Customers and materials are related through sales orders but not directly to each other.
- Place entities into information domains. In Step 1, information domains were identified. Candidate entities are related to candidate services. These services are related to business areas. Business areas are the basis for creating information domains. When complete, candidate entities and their information domains will be aligned with the candidate services and their corresponding business

areas. To the extent these are not aligned, it will be necessary to determine underlying reasons and make whatever adjustments are needed.

4.2.4.4 Step 4 - Conduct Workshops with the Client

The conceptual data model provides a business view of the core entities in the organization. To achieve this it is absolutely critical that the model is developed and validated by business subject matter experts. The data architect or modeler acts as a facilitator and advisor in these workshops but must resist the temptation to let technical data modeling issues undermine the business views of how they see information in the business context.

The following are several key questions that can be used to guide these workshops:

- Is the model is comprehensive and does it cover all areas of the SOA?
- As far as possible, has business terminology been used?
- Are the boundaries between the information domains clear and with minimal ambiguity?
- Are there clear, unambiguous definitions of each of the entities, relationships and information domains?
- Is there a clear alignment of entities, candidate services, information domains and business areas?

4.2.5 Deliverable

The conceptual data model typically will be coarse-grained and is intended to show the broad interactions that occur between the candidate entities. It is not intended to be granular enough in perspective for actual development, but rather to provide a high level view of the information universe under consideration. It depicts, in both graphical and textual form, the structure and content of the information domains of information that will be exposed and shared across the enterprise as the SOA evolves.

The recommended notation for this work product is entity-relationship (ER) modeling supplemented by discussions of the reasoning behind the ER models. There are several "accepted" variations of this notation, all recognizing the same basic constructs. The most common is adopting the information engineering approach but choice may be influenced by tools available on the project.

The names used in the conceptual data model are defined in the business glossary. Additional annotation should be provided wherever possible to add clarity and assist understanding of the diagram

The conceptual data model will usually comprise several separate, but related, components:

- A high level entity relationship diagram (ERD) showing the major entities and their relationships
- Information domain level ERDs that show the entities in each information domain and the entities shared between information domains. For example, Product is shared between the Customer information domain and the Material Management information domain
- Entity, relationship and information domain, definitions
- A matrix of candidate entities to candidate services

4.2.6 Dependencies

The primary inputs to the creation of the conceptual data model are:

- The business glossary
- Candidate services
- Existing metadata
- Functional business area and process analysis

All of these need not be complete to start developing the conceptual data model. The model can be started and developed as these other deliverables become available. However, the model cannot be considered complete until all of the dependent work products are completed and reflected in the conceptual data model.

4.2.7 Example

The following model contains the conceptual entities needed for the JKE account open service. There are several points worth noting.

- A customer can have many addresses and address can have many customers as when several customers reside at the same household. In the ER notation the “crow’s foot” represents the many side of the relationship. This is known as a “many-to-many relationship”.
- A customer can have many accounts and an account can have many customers as when there are multiple signers on an account. This is known as a “one-to-many relationship”.
- A customer can have several sets of contact preferences. An example is where a customer has several residences based on the time of the year. But the contact preferences relate to one customer.
- A customer can be composed of other customers. A corporate customer can be comprised of many subsidiaries. Both can be customers depending on the product and the nature of the sales effort. This is known as a “recursive relationship”

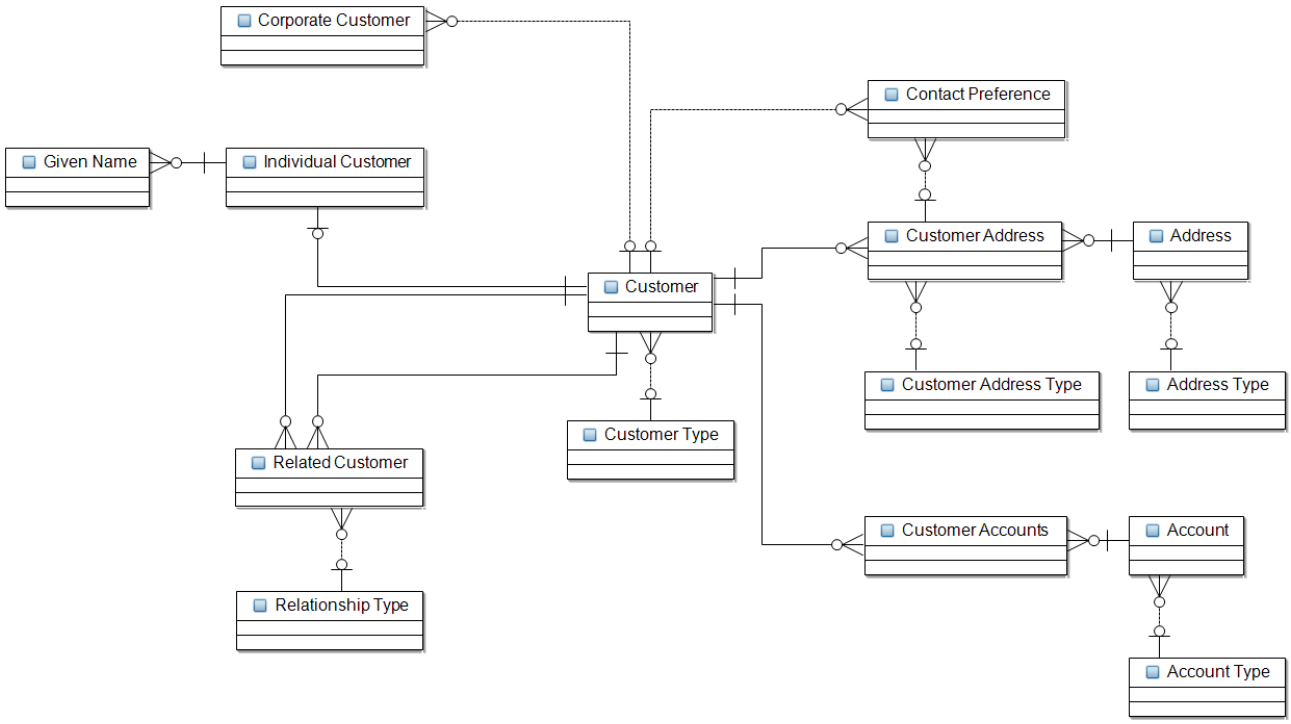

Figure 12: Conceptual Data Model

4.2.8 Guidance

An architect should consider the following normative guidance when developing a conceptual data model:

- The conceptual data model has many similarities to a logical data model. Both are interested in identifying the information / data that the business manages and both will use a similar graphical notation accompanied with text to represent this. The conceptual data model, however, will be at a significantly higher level of abstraction and is more concerned with gaining an understanding of the scope and scale of the information that is managed by the enterprise. Rather than defining the detailed specifics of any one-information domain, the conceptual data model is more focused on knowing that a class of information exists and that it is important.
- The conceptual data model, once developed, is highly reusable and will provide the architectural basis for all subsequent analysis and requirements projects. The development of logical data models will use the conceptual data model as input.
- There is a very strong relationship (or "synergy") between the development of the conceptual data model and the business process definition. Many of the activities in the process definition exist primarily to manage the business information and each piece of information owes its existence to an activity that creates and manages it. Each of these models can be used as input into the development of the other and in most cases they will be developed in parallel and by the same people. More detail on how these are developed in parallel is provided in the "Enterprise Information Modeling" Technique Paper (see Reference [10]).
- When related to the conceptual framework for enterprise information architectures developed by John Zachman, (see Reference [9]), the conceptual data model represents the "Ballpark" & "Owners" view of the data architecture.

- While an conceptual data model should cover the full scope of the enterprise, in some engagements this may not be possible or feasible, given time constraints, etc. If this is the case a 'limited scope' conceptual data model can be constructed and used as a starting point for further modeling work. (i.e., while the enterprise model can be built in pieces, this must be done by extension of a single model and not by the uncoordinated development of several separate models).
- The conceptual data model can be used in the early stages of software package selection. The high-level information domains can be mapped to the data model of the potential packages to determine, at a high level, the "fit" of the package to the enterprise information requirements.

4.3 Create a System Context Diagram

SOA solutions in general are, by definition, highly flexible. This makes defining system boundaries for an SOA solution somewhat meaningless as we expect that the boundaries can change easily and often. However, building such systems can only realistically be achieved if the task is broken down into incremental projects, each being limited to a tightly defined set of business function and data. Like any IT project, maintaining control of the project scope is a critical success factor. Defining the scope begins in the pre-modeling phase undertaken before starting SOMA activity.

Preparing a system context diagram during pre modeling or very early in SOMA analysis helps to limit what information domains need to be investigated and provides some scoping boundaries for the information and data efforts.

It is very important that as well as listing and defining what will be IN the scope of the effort-at-hand, that a list of what will be specifically OUT of the scope of the effort-at-hand be created. Of particular importance are those functions or areas of information that fall “just outside” scope. These “close” functions are typically where issues and misunderstandings arise.

4.3.1 Definition

The system context captures the data and functions passing across the system boundary reflecting the to-be or future state. In doing so it provides a reference point for project scope issues as well as providing a high level input to the key activities in the SOMA identification phase: domain decomposition, goal service modeling, and existing asset analysis.

4.3.2 Objective

The system context work product initially represents the entire system as a single object or process and identifies the interfaces between the system and external entities. Usually shown as a diagram, this representation defines the system and identifies the information and control flows that cross the system boundary.

The system context highlights several important characteristics of the system: users, external systems, batch inputs and outputs, and external devices.

- External events to which the system must respond
- Events that the system generates that affect external entities
- Data that the system receives from the outside world and that must be processed in some way
- Data produced by the system and sent to the outside world

Because SOA solutions leverage existing system data and function, the system context can play two roles in SOMA:

1. A system context diagram for the SOA system as a whole defines the boundaries and scope of the SOA project being implemented.
2. For each existing system in the SOA, a system context can provide insight into distribution of function and data. This is valuable input into realization decisions later in SOMA where composite services, information services, or micro-flows might be required to implement a specified service interface.

Note that the system context may limit the breadth of its coverage to emphasize just one class of external interfaces, for example, only data placement. Additionally, the details required at lower levels of elaboration will depend upon what interfaces are to be subsequently implemented.

4.3.3 Value

The value of this work product is:

- To clarify and confirm the environment in which the system has to operate. Once agreed to by the client and the development team, the system context becomes very useful for maintaining focus on the development effort.
- To identify cases where key data or function for candidate services is distributed across multiple existing systems.
- To verify that the information flows between the solution to be installed and external entities are in agreement with any business process or context diagrams.

The impact of not having this work product is:

- In the early stages of a project, without an agreed-on context for the system, it is difficult to define the boundaries of the project effort.
- There is risk of either expanding the development effort into areas that are not part of the system or of overlooking areas that should be developed.

The system context need not be developed in the following circumstances:

- The system is not complex and has no external interface or data conversion requirements.
- The client or another third party vendor developed this work product and the content has been shared with the current project team.

4.3.4 Approach

Several notations can be used to diagram the component of system context. Most often they are represented as a level-0 process model, although static object models or functional models can represent them. Diagrams are generally better than text for this kind of overview material. The template below (Figure 13) shows information flow into and out of the system. Note that in an actual diagram instance, each information flow would be labeled with a descriptive name to help the reader understand the purpose of the interface. Each flow then should be described in more detail in narrative form to supplement the diagram. At a minimum, the following information should be provided for each flow:

- Owner of the external entity
- Number of users / transactions represented by the flow
- Frequency of the transactions for each flow
- Approximate volume of data contained in each transaction
- Security Controls required

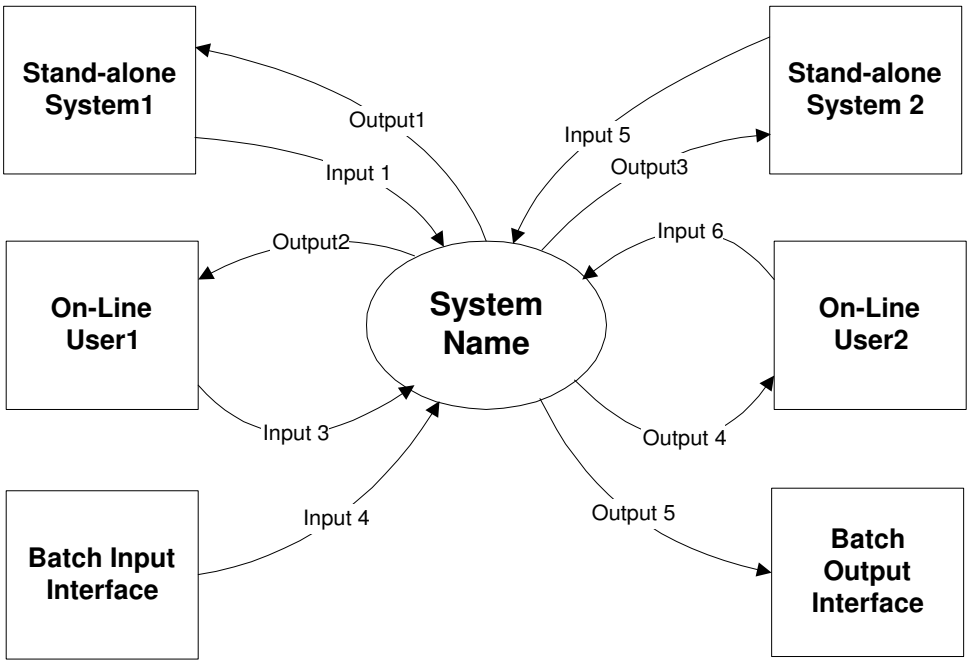

Figure 13: Example of a System Context Diagram

4.3.5 Deliverable

The deliverable of this task is a system context specification as defined in APP011 (System Context).

4.3.6 Dependencies

This task relies on the following input

- It is recommended (see guidance in Sect. 4.3.7) to leverage the business process model specification when developing the system context diagram.
- Data flows in the system context diagram can be expressed using the entities and information domains defined in the conceptual data model (see Sect. 4.2).

The main output of this task is the system context diagram which is the input for the following tasks:

- Business process modeling
- Use case modeling
- Service modeling
- Data modeling

4.3.7 Guidance

An architect should consider the following normative guidance when developing a system context diagram:

- Identify all external entities and users that need to interface with the system. This information can often be obtained from business process models and definitions that are either pre-existing or currently under development. Entity definitions should align with the glossary and conceptual model used for the SOA system.
- Create the initial context diagram by drawing an object (box or circle) and labeling it with the name of the system to be developed. Surround this initial object with other boxes labeled with the names of the external objects.
- Ask the customer, user, or business area expert what the associations are between external objects and the system and how often these occur and what security and privacy controls are required at this point. What is the trust level offered or needed by the connected systems. Also it needs to define the quality level of the information expected and supplied (i.e., what is the level of confidentiality, integrity, and availability that is expected and will be supplied from/to other systems). Some of this information may be indicated on business process models, candidate service model, or context diagrams.
- For existing systems being leveraged or replaced by the SOA solution, consider creating context diagrams if they do not already exist. This will assist in validating the new context information, by indicating existing external interfaces to other systems, equipment, databases, files, users, and communication links. It will also indicate cases where key entities exist in multiple systems and how those systems participate in the lifecycle of the entity. Such details can be valuable input into making the best realization decisions for implementing services acting on those entities, e.g. when micro-flows, composite services, or data aggregation services are the most suitable solution for a given service interface.
- Indicate what volumes of data and what kinds of transactions, with their associated feeds and speeds, must be supported by the system.
- Document issues, issue resolutions, and reasons for everything captured on the diagram.

4.4 Create a Catalog of Data Sources

Existing asset analysis determines whether and where we can leverage existing systems to realize service functionality. This activity helps to identify candidate services and flows that may have been missed by domain decomposition and goal-service modeling. Existing asset analysis also provides key input to service realization decisions. The scope or focus of analysis can be narrowed by using the goals that are analyzed and refined by goal-service modeling and the functional areas and processes defined during domain decomposition. If a system context diagram has been developed for the solution then this too can be used to limit the scope.

Existing asset analysis identifies and validates candidate services and flows. Technical constraints related to existing systems are evaluated as early as possible for risk management purposes; thus conducting technical feasibility of service realization decisions is often done as soon as possible after/during existing asset analysis.

Existing assets can include both data and functionality that is already in place, or can easily be put in place for the organization. Thus the analysis should include an inventory of existing services, existing application APIs, and existing data (both for structured and non-structured data).

The catalog of data sources identifies, describes and groups business data in terms of its characteristics. These data sources are an extrapolation of the business entities defined in a conceptual data model developed for the SOA solution, grouped to reflect data characteristics. The way in which data is grouped will guide many later service realization and implementation decisions, and is driven by business requirements (not technology), such as:

- "Improve customer service and retention"

  A business may have determined that customer dissatisfaction has resulted from the business having too many separate views and touch points from different parts of the organization to the same customer. A goal of implementing the SOA solution is to provide an integrated view of the customer across the entire organization and to establish flexible business processes for managing the key events in the customer lifecycle. To achieve this requires a clear understanding of the data sources where customer data is stored, what application functions operate on the data, how these stores are related to each other in structure, content and lifecycle, and what non-functional characteristics these sources may have that could impact the realization decisions for the SOA solution. With this knowledge, the service designer can identify data issues such as inadequate matching rules and data quality issues which require solving

- "Out of stock items should be fulfilled from the least expensive alternative location"

  For the business to determine the “least expensive alternative location” will require knowing all of the potential systems where the product could be sourced e.g. manufacturing systems, transportation systems, warehouse systems, retail locations. Each of these can potentially have different subsets of a company’s products with different technical representations, different system service level agreements, etc. In order to build a solution to this problem we will have to compose elements from each participating system. Some elements of the composition will be deemed as business rules and exposed to the business process, while others are purely technical issues which can be resolved below the service interface as micro-flows, information services, etc.

This catalog of data sources can be defined at different levels, dependent on the scope of an engagement and the time available for these activities:

- **Conceptual data sources level** - where the focus is on defining the main data stores and their overall characteristics. This level is used as input to the definition of enterprise-wide rules, polices and guidelines on data access, sharing, placement and management. During the identification phase of SOMA this is the level of detail captured. This information then becomes an input for fit-gap analysis and the service litmus test.

- **Logical data sources level** – during the service specification phase of SOMA it is necessary to increase the level of detail to include the full structures and attributes corresponding to the logical data model and the service model. Not all conceptual entities will need to be elaborated in the catalog of data sources. The service litmus test and system context diagram can be used to scope which services are most likely to have data issues and from these the project team can focus on those entities for which more detailed analysis is required to make good realization decisions.

The catalog of data sources, while containing information about the physical distribution of data in the organization, is focused on a logical grouping of data around the key business entities as represented in the conceptual data model. The reason is that it is this representation that is the target structure to be exposed on the service interfaces. By understanding the requirements for each entity (both functional and non-functional) alongside the operational data characteristics shows which service operations will need special care in implementation to address issues with availability, quality, performance, complex data integration, etc.

During the development of the catalog of data sources, any existing information services that are already implemented or partially implemented should also be captured and added to the service model alongside the other candidate services identified through existing asset analysis.

4.4.1 Objective

The catalog of data sources provides a framework for capturing the characteristics of the business data underlying the services and business processes for the SOA project. The work product is used to ensure:

- Distribution of business data, its ownership, and critical events in its lifecycle are clearly understood and taken into account when new services and business processes are designed.
- Potential data issues are identified and appropriate design decisions are taken to mitigate them.
- There is a clear understanding of business data characteristics - key to the successful implementation of SOA business solutions.

At the enterprise level, the work product is used to help define the rules, policies and guidelines on data access, sharing, placement and management.

At the system level, it is used to drive the shape and style of specific solutions; without it, users could end up with poor response times, unexpected business process conditions, or invalid business process results.

4.4.2 Value

The impact of not having this work product is that the current environment is not well enough understood before designing the solution. Without a clear knowledge of the total framework of data required by the business, including its characteristics, the ability to access and share common data is limited. The task of data management becomes very costly, ineffective and time consuming. SOA projects focus on enterprise wide business processes implemented across multiple operational systems. Without considering both the structural and semantic characteristics of these systems as well as their non-functional attributes, it is all too easy to implement business process that fail to meet business expectations of accuracy and timeliness.

Reasons for not needing this work product are if the data requirements are not considered to be challenging, or data access is 'unsophisticated'. For example, where a major portion of the data is 'read only' and stable - customer facing information systems such as railway timetables or on-line shopping catalogues.

4.4.3 Approach

The data systems of an enterprise include all the tools, data stores, and operational systems responsible for storing and managing data. These include relational databases, content management systems, data replication and synchronization systems, data warehouses, federated databases, business intelligence systems, data mining systems, data analytics systems, reporting systems, data quality systems, and integration systems, etc.

The catalog of data sources comprises two distinct but related sections representing the following two views of the enterprises data systems:

- The list of data systems is a catalog of all existing or planned data systems grouped for viewing by system name.
- The list of conceptual entities contains the same set of data systems but this time they are grouped by conceptual entity.

The second view is the most interesting for SOA design as it presents the physical data characteristics aligned to the view we expect to see in the service model.

There is no formal notation for this work product. A largely tabular format with supporting text is a good starting point. As analysis becomes more specific during detailed data profiling this view is likely to be supplemented by diagrams, graphs and reports generated from data profiling and data modeling tools. The results of detailed data analysis may be stored in this work product (for continuity) or documented separately and referenced from this document.

The granularity of the data entities being analyzed also changes during the project.

- During the **SOMA identification phase**, it is adequate to work with coarse grained entities as defined in the conceptual data model. Assessments during this phase are mostly subjective and working with conceptual entities is an appropriate and efficient approach to take in this phase.
- During the **SOMA specification phase**, detailed data analysis is applied to problematic entities. At this point it is necessary to move into the SOA logical data model and the physical representation, that is the canonical message model. The data issues being investigated are resolved by mapping these two views against the physical databases storing the corresponding data and then profiling the contained data values.
- During the **SOMA realization phase**, the analysis continues in even more specific detail during technical feasibility exploration. This may involve building prototypes and testing them against representative operational data.

4.4.3.1 Creating the List of Data Systems

Capture a reference list of the data systems in scope for the project. This includes existing operational systems and planned systems. The objective is to create a check list used for other parts of the work product and as overview information for educating members of the project team. A spreadsheet or table is adequate for representing the list and at a minimum it should include the following:

- System name
- System owner
- Brief functional overview
- Major entity groupings

- List of links to more detailed information (context diagrams, system documentation, data models, etc).

The following techniques can be used to assemble this information:

- Leverage existing enterprise architecture and enterprise data architecture documentation.
- Interviews with system owners, data architects and database administrators.

4.4.3.2 List of conceptual entities and their characteristics

This section of the catalog of data sources captures the major conceptual entities or entity groups for the project and summarizes how they exist in operational and planned data systems. This is critical information for determining the most appropriate implementation choices for realizing each service or business process.

The document should be built iteratively in parallel with other SOMA activities. Not all entities will be investigated to the same degree of detail. Each entity should only be investigated to the point where it is clear which implementation pattern is the most appropriate for the services operating on it.

The content for this section will vary depending on the entities, the project, and the available documentation. The following approach can be used to build this work product.

- Starting early in the project begin building a list of conceptual entities and entity groups from the conceptual data model and the service model. This can be captured in table form.
- For each entity or entity group use the list of data systems built in the previous section to create a high level cross reference of conceptual entities to operational databases.
- For each entity or entity group list the services that will operate upon it. This list should correspond to those services captured in the service model.
- Using the service model and business process model as inputs assign a simple/medium/complex classification for the known requirements against each entity.
  - **simple** means few different transaction types, predominantly read access, and low volumes.
  - **medium** implies more complex transactions, higher volumes
  - **complex** signifies that there is a wide range of different transactions, complex interdependencies, very large volumes.
- The classification at this stage is mostly subjective and is best decided in agreement with business analysts, system owners, administrators, or other subject matter experts.
- Include short comments to justify or clarify why a particular classification was chosen.
- For each entity, assign a classification for the capability of the underlying data systems to implement the known requirements. Again, at this point the classification is purely subjective rating of simple/medium/complex based on the knowledge of existing subject matter experts. Short comments should also be captured explaining why classifications have been reached.
  - **simple** meaning that a conceptual entity maps to a well managed, high quality system which is well established in the organization as the primary repository for that information.
  - **medium** implies that there are several physical systems with shared responsibility for storing the entity and managing different aspects of its lifecycle. However, these systems are high quality and the rules and boundaries for how the entity is shared across them are well defined and adhered to.
  - **complex** implies that a given entity is not well managed in the existing environment. Aspects of the entity exist across many systems with inadequate rules and discipline controlling how that

distribution is managed. Often these entities have a long history of ownership and integration issues in the organization.

4.4.4 Dependencies

This task relies on the following input

- Business glossary
- Business process model
- Conceptual data model
- System context diagram

The main output of this task is the catalog of data sources which is the input for the following tasks.

- fit-gap analysis and the service litmus test

4.4.5 Example

The following table shows an example of a table achieved during the identification phase.

Entity	Operational Systems	Services	Requirements	Implementation
Address	Customer Service, Marketing, Shipping, E-Commerce, Financials, Purchasing	Address.getBillingAddress(Customer) Address.getDeliveryAddress(Order) Address.getResidentialAddress (Customer) Address.getWorkAddress(Customer) Address.UpdateAddress(Address)	**Simple** - all address access is read only except for customer self care which allows updates.	**Medium** - all customer addresses maintained in customer service system - all supplier addresses in purchasing system - no standard address validation algorithm in place - many customers have incorrect, outdated or missing address records
Product	Purchasing, Customer Service, Inventory, Marketing, Order	Product.getProductDetails(SKU) Product.getProducts(Category) Product.getProducts (Category, PriceRange) Product.getProducts(Keyword) Product.getCrossSellProducts(Product) Product.getUpsellProducts(Product)	**Medium** - read only - need to support lists and nested lists - large number of search criteria	**Medium** - all products mastered in Inventory system - common SKU id used in all systems - Inventory database considered very accurate by business and users (automated monthly cleanup jobs) - limited product relationships exist for bundles, cross-sells,

Entity	Operational Systems	Services	Requirements	Implementation
		Product.getReplacementProducts (Product)		accessory sells, kits, etc
Order	Order, E-Commerce Financials, Customer Service, Shipping	Order.createOrder(OrderDetails) Order.getOrderStatus(OrderNumber) Order.getOrderHistory (Customer, DateRange) Order.updateOrder(OrderDetails) Order.cancelOrder(OrderNumber)	**Complex** - supply chain process allows automated creation of complex orders, splitting orders, modification of orders in process. Volumes up to 50,000 per day with 24x7. - complex business rules for approvals	**Medium** - all orders exist as master copy in Order system - common order primary key across all systems - business rules well defined
Payment	Financials	Payment.createPayment (Customer, PaymentDetails) Payment.createPayment (Invoice, PaymentDetails) Payment.getPaymentDetails (PaymentNumber) Payment.getPaymentHistory (Customer, DateRange)	**Simple** - only online credit card payments in scope	**Simple** - all payments processing already centralized in accounts receivable module of Financials system. Public API's already in place supporting E-Commerce application.
Invoice	Financials, Customer Service	Invoice.getInvoiceDetails (InvoiceNumber) Invoice.getInvoiceHistory (Customer, DateRange)	**Simple** - read only - can create payment or dispute from invoice	**Simple** - services already exist against Financials system for supporting E-Commerce application.
Customer	Customer Service, Marketing, Orders, E-Commerce, Financials, Purchasing	Customer.getCustomerDetails (CustomerNumber) Customer.getCustomers (CustomerSearch) Customer.update (CustomerDetails) Customer.create(CustomerDetails)	**Complex** - updates made by many parties within and outside organization on different aspects of existing customers - onboarding a new customer is complex business process with many paths and target systems to be updated	**Complex** - history of disputes over customer ownership has lead to inconsistent implementations across different operational systems - no common primary key - previous attempts to implement single customer view have failed
…				

Table 1: Example of a Catalog of Data Sources

The table is then used to drive decisions regarding further analysis and early realization decisions:

- cases where both the requirements and the mapping to implementation are simple should demand no further analysis. It should be sufficient to use an existing service, wrap a system API as a service, create a simple information service, etc.
- cases where either the requirements or the implementation details are considered complex will require additional analysis to determine the most appropriate implementation choices. This analysis will be carried out in the SOMA specification and realization phases and may include technical feasibility exploration. Failure to do so will almost certainly compromise the quality of the solution being implemented.
- cases falling between these two also call for further investigation but subjective decisions are needed to balance the risks and rewards. Because the requirements are simpler and the implementation choices are less complex, there is much less risk than in the previous cases.

In SOMA identification a high level and subjective assessment is made to determine which areas of business information are likely to be problematic for integration into the SOA solution. In SOMA specification and realization these areas are investigated further using data profiling techniques against representative operational data. The approach for this analysis is detailed in Sect. 5.3.

4.4.6 Guidance

The following are practical suggestions to complete this activity efficiently.

- The objective is to build a clear picture of the characteristics of the data underlying the services captured in the service model. Taken to its extreme this would take considerable effort so it is important to take a pragmatic approach. The following may be options for reducing the scope of the work required:
  - Wherever possible, gather, reference, and reuse existing system documentation. If models exist for previous data integration projects (e.g. data warehousing, operational data stores, master data systems etc.) in the organization then these can often answer many of the questions surrounding data in the SOA solution.
  - Use existing subject matter experts to qualify and prioritize those areas of the conceptual data model which are likely to be problematic.
  - Review early iterations of the service litmus test and service realization decisions to eliminate from further data analysis those areas of the conceptual data model for which a clear solution is already known.
- Before starting the assessment process make sure the participants clearly understand the nature of the data problems most likely to jeopardize the project. The simplest are pure data quality issues, i.e. missing and incorrect data. These can be easily identified and quantified with data profiling. Equally or more important are the structural and semantic alignment problems that exist across operational data systems that exist to serve different business functions. These are less easily found when taking a purely technical approach so any insight you can get from business information domain experts is very valuable. These experts can only help you if they understand the nature of SOA and how it uses data.
- It is very likely that the most problematic data areas will be already well known. These often have a long history of integration and ownership issues. Gather anecdotal experience and opinions on previous attempts to resolve issues and conflicts around these entities: what worked or didn't work, why, and how those involved would do things differently next time.

- The subjective classifications outlined in this paper show a representative approach taken during SOMA identification. Be prepared to define the most appropriate assessment criteria separately for each project or even for each domain within a project. Choose criteria that are meaningful and acceptable for the client. Remember that the objective is to classify the difficulty of the data problems relative to each other – there are no absolute values during this process.
- Be willing to accept that compromise designs and solutions might be the only practical way forward with some entities. This is especially true in cases where the politics around control of an entity is complex or controversial. In these cases it may be necessary to accept some level of duplication or de-normalization in the data model design to reach a workable solution. In these cases document the factors leading to the decision, lay out the potential risks of the chosen solution, and ensure the decision is formally signed off by those controlling the project.
- Resolving all issues of data quality within an organization is an open-ended effort that clearly falls outside the scope of implementing an SOA project. During the SOA project you should only address data issues that will clearly jeopardize the SOA delivery. Other issues should be passed to the appropriate data governance body within the organization so that they can decide if/how they will be addressed under separate projects. The SOA project may act as a catalyst for a broader ongoing data quality initiative in the organization but this should remain separate from the SOA project.

4.5 Identify Information Services

4.5.1 Definition

SOMA recognizes that implementing SOA solutions requires compromise to balance the objectives of developing a well thought out flexible and extensible solution against pragmatic considerations such as time to delivery and return on investment. As a consequence, it is often necessary take design decisions very early in the project based on experience, intuition and best judgment rather than waiting until detailed analysis and modeling is completed (or even started). This approach enables critical planning decisions such as budget and investment requirements, project schedules, staffing, and skills development to be taken earlier and with more certainty than would otherwise be the case.

In SOMA identification, one such set of decisions is taken during service type identification. This is the process of categorizing all candidate services by “service type” before the specification and realization phases have been started. Examples of “service type” are process services, rule services and (as will be explored here) information services.

While seemingly counterintuitive, this early categorization of services is a natural part of the architecture and design process. It is very common for architects and IT specialists to be subconsciously assessing solution options even during very early stages of requirements gathering. From a typical requirements discussion, and good knowledge of the existing IT environment, architects and IT specialists can identify the most suitable implementation choices available, subjectively assess the advantages and disadvantages, and form an opinion on the best solution. For many requirements, there is a high probability that the resulting decisions will be appropriate for the solution, and that by taking the decision so early there can be considerable savings in cost and implementation time. For some cases, where the requirement or the existing environment it maps

to are complex, the early decision can be a guide to reduce and prioritize the necessary analysis and modeling activities required to arrive at the best solution design.

The following two examples illustrate cases where early service categorization can accelerate SOA solution delivery without compromising the final solution quality.

1. An organization built up through acquisition has consolidated many legacy systems into a purchased ERP system. They have stated their strategic IT direction is to continue to consolidate and expand this system for many core business functions. For all SOA services needing data and functionality within the scope of the ERP system, service wrappers will be built around the public APIs for the ERP.
2. An organization has a long history of failed attempts to reach a shared definition and management of their "customer" data entity which is distributed across several of their core business systems. The organization has taken a decision to implement an MDM system as part of the objectives of their SOA project. This system will be responsible for maintaining the definitive version of customer data for the organization and also for synchronizing this across the existing systems. For the SOA project, all the SOA services needing to retrieve or maintain customer data will be provided by the MDM system.

Even though SOMA advocates taking early decisions for service categorization, these decisions are not binding for the duration of the project. As further analysis, modeling and design activities are undertaken, there will be some cases where it becomes clear that initial thinking was inadequate, insufficient information was available, incorrect assumptions were made, or project requirements have subsequently changed. Such cases are subject to the same change management process that applies to any project change.

4.5.2 Objective

The objective of this activity is to review all candidate services in the service model and identify which are information services. This categorization is done subjectively and where uncertainty exists this should be recorded with the decision and taken into consideration in downstream SOMA activities. The approach and criteria for identifying information services are outlined in 4.5.4

4.5.3 Value

Identifying information services during service identification has the following benefits:

- It allows project planning decisions such as schedules, budgets, staffing, skills development, purchasing etc to be taken earlier in the project and with more confidence.
- It guides and prioritizes the dependent activities for specifying and designing the services and thereby reduces the time and cost needed to complete the project.

The impact of not undertaking this activity is that service implementation choices are not known until after SOMA realization is complete. This complicates project planning and management which must contend with many uncertainties until this is complete.

Reasons why this activity might not be needed are:

- That the scope of the project is limited to business analysis and/or design but does not include implementation.

- That the goal of the project is to assess implementation options rather than to deploy a solution.
- That the goal of the project is to deploy infrastructure with business process and services only serving as test cases for the new platform.

4.5.4 Approach

Information Service Identification is performed before detailed analysis and specification has begun. It is a subjective process requiring the collective experience and judgment of those tasked with making the identification. The following steps can be used to complete this activity:

1. Identify service categorization criteria.
2. Assemble participants with required knowledge and skills
3. Categorize candidate services
4. Ongoing review of decisions

4.5.4.1 Identify Service Categorization Criteria

During this activity each candidate service in the service model will be categorized into a service type which will then be used to guide the downstream analysis and design activities for it. To do this, the available "service types" must first be defined for the project. This will depend to some extent on the planned infrastructure of the solution and what technical capabilities it supports (e.g. does it include a configurable business rules engine?) as well as an agreement of the level of detailed implied by the service type (e.g. do we have an "application service type" or do we have a "CICS transaction service type" and a "J2EE service type" and a "SAP BAPI service type" ?).

Only the definition of an "Information Service Type" will be addressed in this paper. The following table provides a set of questions that can be asked of each candidate service. If the answer to one or more of these questions is "yes" then the candidate service is very likely an information service.

Information Service Identification Criteria
1. Is the service responsible for **retrieving or maintaining business data without applying any additional business processing or behavior**?
2. Does the service **retrieve or maintain unstructured content**?
3. Does the service expose **data that is distributed across multiple operational systems and/or databases**?
4. Does the service apply **data standardization and cleaning rules**?
5. Does the service **retrieve or update data which already exists in an integrated data store** e.g. a data **warehouse**, an **MDM** system, an operational data store, etc.
6. Does the service **return the results of complex data analysis** such as statistical analysis, trends, hidden patterns and relationships, summarization, predictions, etc?
7. Does the service **retrieve or update data from a system for which no public API** exists?

A full treatment of information service types and definitions is given in 6.2

4.5.4.2 Assemble Participants with Required Knowledge and Skills

Because the service categorization activity is highly subjective, it is important that those making the judgments have adequate skills and experience. The following is a checklist of the roles required:

- Business analyst: responsible for resolving questions about the business requirements being addressed by the candidate service being examined.
- Data architect: responsible for understanding the data being exposed through the service interface and where it currently exists in operational systems and data stores.
- Information services specialist: understands the capabilities and limitations of the various information services patterns and platforms and how they can be applied.
- Application specialist: understands the capabilities and limitations of existing applications that are under consideration for implementing the candidate service.
- SOA solution architect: understands all the proposed solution components and the architectural decisions defining their roles and responsibilities. Responsible for taking final decisions.
- Project manager and/or sponsor understand the time and budget constraints for the project and how these might constrain implementation choices.
- Additional specialists: depending on the service classifications being used and the scope of the project there may also be need for business process designer, product specialists, systems management and operations staff, or others who have experience to leverage in the decision making process.

4.5.4.3 Categorize Candidate Services

One or more facilitated workshops with representatives from section 4.5.4.2 is a good way to categorize the candidate services. Using the established and agreed criteria (section 4.5.4.1 for information services), the attendees discuss and agree the most appropriate category for each candidate service type.

Record notes against each decision summarizing the main justifications and assumptions used to arrive at the decision.

Expect during this process that there will be disagreement and debate. Be prepared to leave some services unresolved or deferred until further information is available. The objective is to first find those services for which there is solid agreement and document them so that planning can begin to support their design and implementation.

For those services that remain in the undecided category the following options are available:

- SOA solution architect takes a decision, records the issues, risks and justification, and proceeds into specification and realization activities with caution.
- No decision is taken until further investigation is done. This could mean deferring a decision until well into realization phase where technical feasibility studies might be needed to confirm final implementation choices.

4.5.4.4 Ongoing Review of Decisions

Decisions made in 4.5.4.3 are used for primarily for project planning and guide the downstream SOMA activities for each service. It is hoped that a high proportion of these decisions will prove justified until project

completion but at this point they should not be considered final. There will be several project review points which determine if the decisions stand or are changed later in the project. These include:

- Immediately after service categorization is complete the results should be circulated to a wider technical and busiess audience for review of the assumptions and justifications used in the decision making process. Be prepared to revisit decisions if this review process uncovers new information contradicting the original decision.

- High level project planning including estimated schedules, costs, dependencies, etc. can begin immediatedly after service categorization. Feedback from both project management and sponsors may constrain the solution options and require changes.

- During project execution, detailed analysis, modeling and design may uncover additional information that invalidates intial decisions and requires changes.

4.6 Identify Information Criteria for Service Litmus Test

During SOMA identification, a list of candidate services (and other elements) is identified. These services are all potentially useful in the enterprise wide SOA solution. However, there are significant immediate project costs for deploying these services, as well as the ongoing costs for maintaining published services. It is therefore very important to carefully analyze and evaluate each service interface before choosing to expose it into the solution. A set of criteria in the form of the service litmus test can and should be used to filter the collections of candidate services. This metaphor is used to denote a set of tests, that when applied, will determine if a given service should be eligible for exposure using a service description. These tests are employed together and help answer questions such as – “From the list of candidate services, which ones should be exposed? And thus, which ones should we fund? Which ones have business value?”

Like many activities in SOMA, taking exposure decisions is a very iterative process during the project. Some services, particularly those based on existing assets, may be very well defined and understood early in the project allowing exposure decisions to be taken early with little risk. Others will need detailed specification and even realization activities (such as feasibility studies) to be undertaken before a final decision about exposure can be made. This document outlines two important data considerations for service exposure which are data quality and service reusability. To fully understand either requires completion of the SOMA specification phase. This does not preclude service exposure decisions being taken early for simple and well understood services. However, where doubts exist, the exposure decisions should be delayed (or at least validated) after service specification is complete.

Service litmus test criteria are defined based on the priorities, objectives and business environment for each SOA project. An example set of criteria is shown in Figure 14.

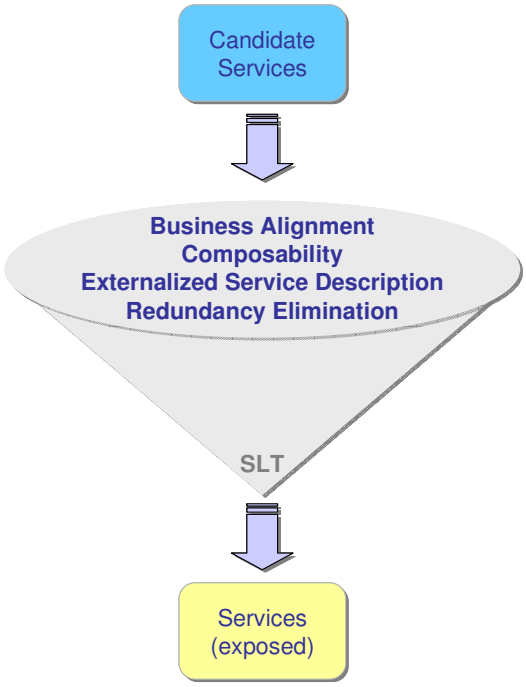

Figure 14: Service Litmus Test

These standard criteria (which can be adjusted in client engagements) are:

- **Business alignment**
  This refers to the traceability of a service to business goals, i.e., is the service of value to a business?

- **Composability**
  A service that is composable is self-contained and can participate in a composition or choreography. It is deployed independently but may cooperate with other services at run-time to execute business processes in support of business goals. There are no external dependencies involved that would disallow the service from participating in a composition.

- **Externalized Service Description**
  A service has an externalized service description, either generated through automated tools or created manually.

- **Redundancy Elimination**
  This refers to the functionality of a service being provided once and used in multiple business processes.

Candidate services that pass all four parts of the service litmus test should then be exposed as services in the SOA. There may be candidate services that did not pass the service litmus test but which are still implemented as services. The service litmus test is an aid to determine which services to expose; if a business chooses to expose candidate services that did not pass the service litmus test, the implication is that benefits associated with a SOA will not be realized.

Candidate services that do not meet the service litmus test will have to be implemented in some fashion as they are required to fulfill business needs. They may be implemented as methods on service components

and will not require the generation of WSDL or other forms of service definitions; or they may be used as non-exposable entities.

Information analysis plays a role in two areas of the service litmus test – composability and redundancy elimination. The data architect can bring both qualitative and quantitative input to the decision making process in these two areas.

4.6.1 Service Composability

For a service to be exposed it must be composable. One aspect of composability is that the service meets the required QoS attributes as defined in the composition's functional and non-functional requirements. From the information perspective the most significant aspect of this requirement is to understand and quantify the accuracy of the data that will be consumed and/or exposed by the surface.

It is very unlikely that the data being delivered by a service will be 100% accurate and complete. For simple services surfacing data from one operational system the data quality issues will focus directly on the accuracy and completeness of the underlying tables. For more complex services integrating data across multiple systems there will be additional issues around the strategies used for joining the data and resolving data conflicts across the participating systems. The data quality analysis activity (see Sect. 5.3) outlines how to investigate and record the severity of these problems for different business information. During the service litmus test this data quality measurement must be taken into account as a measure of service composability and if unacceptable then action must be taken.

Data quality issues may surface as either hard or soft errors. Hard errors result from missing or invalid data values which can be monitored and handled by the programs providing the service implementation or passed back to the service consumer as an error condition. Soft errors can be caused by a wide variety of data conditions and result in business data which is incorrect as it pertains to the business, but does not necessarily create a detectable exception that can be handled in the implementation code.

Examples of hard errors include finding character data in a field that should only contain numbers, finding an order detail record for which no order master record exists, or finding empty values for a mandatory field. The data architect can use simple data profiling and analyze the data directly to identify these types of errors, quantify how widespread they are, and report the results as input to the service litmus test.

Soft errors are more difficult to detect and normally require that the data architect works closely with business analysts who have a historical understanding of the business rules and external symptoms that indicate underlying data problems. Consider the following example. For a supply chain business process, a candidate business service has been proposed to return a list of all orders made against a given supplier. This will be used to estimate what level of discount can be expected against new orders and is therefore a factor in selecting which supplier the process will choose before making an order. However, 10% of the orders in the order system have supplier numbers which are missing, invalid, incorrectly assigned, or outdated. This means that the aggregated order amounts are often incorrect even though the service will produce no visible errors or exceptions. If this affects the discount calculation then there is a possibility that the least expensive supplier may not be chosen and so the company loses money by paying a higher price than was necessary.

It is the responsibility of the data architect to participate strongly in the evaluation of data quality issues that can impact on service exposure decisions. He should work proactively with business subject matter experts and data analysts to:

- Investigate the data underlying candidate services
- Quantify and present the results as input to the service litmus test. The results must be presented in both technical terms as well as business impact terms.

- Where issues are found the data architect should work with other team members to identify alternative solutions to mitigate the problems. Typically these alternatives could be:
  - Modify the service implementation design to address data issues and correct them during runtime.
  - Initiate external efforts to correct the underlying data so that the proposed service can meet acceptable QoS without changing its design.
  - Accept the business cost of ignoring the problems and do nothing.
  - Do not expose the service.

4.6.2 Redundancy Elimination

The second area of the service litmus test for which data architecture can play a significant role is redundancy elimination. Here the objective is to optimize the service portfolio of an organization such that the number of services exposed is kept to the minimum number possible while still covering the required areas of business functionality. This is a challenging task. SOA projects are almost always developed in phases with each phase strictly limited in scope. The paradox here is that each element developed for the solution must be designed for reuse in the wider context of a full enterprise SOA deployment. Failure to design this way prevents the enterprise from ever achieving the full benefits of SOA.

The service litmus test specifically demands “Can this service be used by the business stakeholder within all processes where its function is required?” To successfully answer this question requires a full domain analysis covering all domains needing that piece of functionality along with variation oriented analysis of the most likely change scenarios that could impact that service. In practice, this level of detail is rarely available and so some level of subjectivity is required based on the experience and judgments of the project team and business domain experts.

Applying information architecture techniques at this point can greatly enhance the objectivity of this part of the service litmus test by adding the question “Is the service interface aligned to the logical data model?” This is because the logical data model has been developed specifically to ensure that a commonly agreed, well structured representation of enterprise information that exists for the solution.

- The logical data model has been derived from the conceptual data model and business glossary for the project meaning it is aligned with the way the business understands the information.
- Where possible the logical data model is derived from industry agreed standards meaning that extending or integrating the SOA solution with new products or partners will be simpler.
- The logical model has been designed to meet the needs of the use cases and change scenarios defined for the project meaning it will support anticipated extensions with minimal disruption to services and processes that are already deployed.
- Aligning all service interfaces with the logical data model greatly enhances reusability and thereby eliminates redundancy. If services are exposed which conflict with the logical data model then it is very likely that it will fail to meet the requirements of future service consumers. This, in turn, leads to additional work by the consumer, or creation of new services which overlap the functionality of existing services.

5. SOMA Specification Phase

SOMA identification finds the candidate services, subsystem boundaries and flows/processes along with variation models. In SOMA specification we need to specify the details of these elements within the service model and the service component model respectively. Service components needed to realize the services will also be defined. This is then used as input to SOMA realization decisions and eventually the detailed design needed for implementation.

In SOMA specification, the three key elements of services, components and flows are elaborated through the following activities:

- Service specification - defines the dependencies, composition, exposure decisions, messages, quality service constraints and decisions regarding the management of state within a service.
- Subsystem analysis - identifies the service components (and related functional and technical components) and relationships that will be used to realize the services.
- Component specification - elaborates on the service components and creates the details of the service components required to realize services.

The information architecture activities for the specification phase are the following:

- **Develop Logical Data Model (see Sect. 5.1)**

  The logical data model defines the data structures and relationships for all the business information entities within the scope of the SOA project. The conceptual data model developed in SOMA identification defines the information domains and entities in scope. The logical model takes these high level entities and elaborates to include detailed definitions of their attributes and relationships. This model becomes the reference for detailed service modeling and business process modeling. Where possible, industry standard data models can be used directly or indirectly as references during the logical model development.

- **Updating the Business Glossary (see Sect.5.2)**

  When we introduced the concept of a business glossary (see Sect. 4.1), we mentioned that the business glossary has to be refined continuously. Often, the common understanding of a term and its taxonomy changes throughout the project, in particular when new experts are introduced to the team or when we start to design various artifacts in more detail.

- **Perform Data Quality Analysis (see Sect. 5.3)**

  In SOMA identification we perform a high level and subjective assessment of the operational data quality and how easily it can support the identified services. From this assessment we determine which services will face the most complexity and/or risk in meeting the required data service level agreements during implementation. During the SOMA specification phase we perform detailed technical analysis of these data areas to quantify the data quality risks, validate data matching rules, define detailed data integration mapping strategies, and begin evaluation of alternative service realization choices where providing acceptable data quality proves to be problematic. This may lead to technical feasibility studies in SOMA realization.

- **Define the Canonical Message Model (see Sect.5.4)**

  The service model is developed iteratively through all phases of SOMA. In SOMA identification, services are identified in high level business terms. During SOMA specification, we add detail such that the service interfaces, service components, and subsystems can be analyzed and eventually designed and implemented. The physical representation of data in this process is focused on the messages defined in the service interfaces. Each service contains one or more operations and each operation includes input and output message definitions. To maximize service reuse and service

interoperability all service messages should comply with a canonical message model. The canonical message model defines the physical structures, rules and semantics of the data being passed in service messages. It is normally implemented as an XML schema or set of schemas. Like the conceptual and logical data models from which it is derived, the canonical message model may be based directly or indirectly on industry standard models.

- **Begin Data Mapping Across SOA Architectural Layers (see Sect.5.5)**
  SOA solutions are designed and deployed as multiple architectural and operational layers. Data flows exist both horizontally across an architectural layer (e.g. through the activities of a business process) as well as vertically up and down through architectural layers (e.g. from a business process down through the service, service component, application and database layers). Establishing standardized and shared data representations throughout the solution helps minimize data transformations required for the solution to operate. However, SOA implementations invariably combine development of new elements with reuse of existing assets and integration with external process and service providers. The data mapping activity in SOMA defines the data transformation rules needed to integrate all solution elements to ensure data is shared correctly within and across architectural layers.

5.1 Create a Logical Data Model

5.1.1 Definition

The logical data model is based on the conceptual data model developed in the SOMA identification phase. The conceptual data model is at the highest level of abstraction. The logical data model is at a lower level of abstraction than conceptual data model because it provides significantly more detail. It and forms the link between the conceptual data model and the canonical message model (see Sect.5.4) or the physical data model. The latter reflects the actual structure of the data as it will be physically stored in the database. This model will be created as part of the realization phase.

The following define the most significant differences between the conceptual data model and logical data model:

- **Primary Keys**
  - Each entity in the logical data model is assigned a primary key. An entity's primary key is the attribute or set of attributes that distinguish one instance or row of the entity from another. Customer Number in the in Customer entity and Account Number in Account entity are candidate primary keys. The main characteristics of the primary key are that is unique and stable, i.e. it is unchanging over time. Customer Name is usually not a good primary key since names may change over time. Several attributes in the entity may be "concatenated", that is combined, to form a primary key. If Account Numbers are assigned across all types of accounts, Account Type may need to be used along with Account Number to form the primary key.

- Each entity in the logical data model must have a primary key. In the example below, the attributes that form each entity's primary key are shown "above the line". All non-primary key attributes are shown "below the line".
- Note that in the conceptual data model, when attributes are shown it is for illustration purposes only. Such attributes are always shown "below the line" in the entity symbol. One of the main characteristics of the conceptual data model is that no primary keys are identified. In the logical data model, primary keys are mandatory.

- **Attributes**
  - The attributes included in the logical data model must be sufficient to support all aspects of the services being specified. Unlike the conceptual data model where a few attributes may be shown for illustration purposes only, all of the attributes for each entity must be included in the logical data model.
  - In the both the conceptual data model and logical data model, attributes must use names that have business meaning. Attributes should use a business name that is consistent with the business glossary.
  - Attributes in the logical data model are not assigned with a physical data type. This occurs in the physical data model in the realization phase.

- **Relationships**

  Relationships in the logical data model are formed by linking entities using their primary keys. In the example (see Sect. 5.1.6):
  - In both the conceptual and logical data models, Customer is linked to Customer Preferences in a one-to-many relationship (i.e. one Customer can have many Customer Preferences). However, in the logical data model, this link is formed by "migrating" Customer Number, the primary key of Customer, to Customer Preferences as a "foreign key". This is denoted by "(FK)" after the attribute name in the model. Customer Number becomes part of the primary key of Customer Preferences and appears "above the line" in Customer Preferences. Because the primary key of Customer becomes part of the primary key of Customer Preferences, the link is a solid line. This is known as an "identifying relationship". The entire primary key is Customer Number and Date From. In the business, this allows the tracking of changes in customer preferences over time. This may be used to correlate preferences with life events.
  - Customer Type has a one-to-many relationship to Customer in that a single type of customer can apply to many individual customers. Customer Type is therefore migrated as a foreign key to Customer but not as a part of the latter's primary key. This is because Customer Number is unique across all customers regardless of their Customer Type. For this reason, Customer Type is shown "below the line" in Customer. Also for this reason, the relationship line in "dashed" instead of solid and denotes a "non-identifying" relationship.
  - In the conceptual data model, Customer and Address have a many-to-many relationship. This is denoted by "crow's feet" at both ends of the association. In the logical data model, this many-to-many relationship is replaced by an "associative entity", Customer Address. The primary key of Customer address is the union of the primary keys of Customer and Address. An associative entity associates two or more independent entities by combining the primary keys of both.
  - In the conceptual data model, Customer has a "recursive relationship" whereby a customer can be comprised of other customers. This is denoted in the conceptual data model by Customer having a many-to-many relationship to itself. In the logical data model, this is replaced by an associative entity containing two instances of Customer Number. Since an entity cannot contain two attributes with the same name, one of the Customer Numbers is assigned a "Rolename", Related Customer Number.
  - In the logical data model, an entity can have multiple "subtypes". These are 'child" entities to the "parent" or "supertype". In the example, Customer has two subtypes, Individual Customer and Corporate Customer. Customer Type is a "discriminator" in the parent entity. It is not part

of the primary key because the parent and child entities share a common primary key, Customer Number. Each "row" in the Customer entity has a unique Customer Number. The Customer Type determines whether the subtype will be an Individual Customer or a Corporate Customer. The relationship from parent to child is a one-to-one relationship; there are no "crow's feet" at the child end of the relationship.

5.1.2 Value

First and foremost, the logical data model enables the SOA team to understand the detailed information requirements to support all specified services. This includes the following:

- All attributes using definitions based on business definitions needed to implement the specified services
- A target governance model that describes the ownership of data among the specified services.
  - Ownership defines which services create, update and delete which attributes and which only use attributes maintained by other services
  - This becomes the basis for implementing the data sharing model regarding how data is shared between the specified services

Second, the logical data model is used as the basis for consistency across the following artifacts:

- the canonical message model,
- the business process model and more specifically the object definition in that model, and
- possibly the physical data model in the realization phase.

Finally, the logical data model is used to improve communication between the business users of the data and the application developers. Eventually, all of the work that a business does comes to rest in its data. Understanding the business nuances that are inherent in the data is one of the quickest and surest ways for an application developer to obtain an understanding of the business. In developing application services, the application developer needs this understanding to ensure that applications meet the information needs of the business users.

5.1.3 Approach

The logical data model can be constructed using the following activities which to a degree and be performed iteratively and in parallel:

- Identify the Information Requirements
- Identify Attributes
- Create Normalized Entities
- Verify the Normalized Entities
- Create the Target Governance Model

5.1.3.1 Step 1: Identify the Information Requirements

Identify the information requirements to support each service being specified. Each service as it is being specified must identify the data it needs to perform its operations. These are the services information requirements. At this point the focus is not on where the data is created or how it is stored only that it is required.

5.1.3.2 Step 2: Identify Attributes

Identify the attributes at the lowest possible level of granularity associated with the information requirements. Oftentimes information requirements are initially expressed in terms of higher level information constructs. For example, Customer Name may be broken down into multiple attributes such as: Family Name, Given Name (this may be recursive from zero to many), Nickname, Courtesy Titles Used (e.g. Mr., Mrs, Ms, Dr.) so that it becomes a grouping of attributes. Perform a gap analysis between the required attributes and the data elements in existing systems and the attributes in standard models. This may lead to the identification of services needed but not specified.

5.1.3.3 Step 3: Create Normalized Entities

Entities are groups of attributes. In a normalized model each attribute is entirely “dependent” on or “a function of” the primary key of the entity. In the example, Individual Customer is an entity which has Customer Number as its primary key. Family Name and SSN are dependent only on Customer Number since each customer has only one of each. While a customer can have only one family name, he or she can have multiple given names. One option would be to have a Given Name as a “repeating group” in Individual Customer. This would involve adding attributes such as Given Name 1, Given Name 2, Given Name 3 and so forth to Individual Customer. A more open ended approach is to move the Given Name to its own Entity with a primary key of Customer Number and Given Name. This allows any number of given names to be associated with the each Customer Number. As a practical matter, since there are rarely more than five given names, this may not make sense in a real application but it illustrates the concept in the context of our example.

5.1.3.4 Step 4: Verify the Normalized Entities

Perform a gap analysis between the normalized model created in Step 3 and the entities, attributes and relationships in existing application data structures and to standard industry models. Does the standard model contain attributes that are absent in the normalized model? If so, do they relate to services not supported by the normalized model? Conversely does the normalized model support services not contemplated by the standard industry model? Steps 1 through 3 provide deep insights into the information needed to support business services. In Step 4 existing assets are used to verify and refine these insights and in doing so improve the logical data model.

5.1.3.5 Step 5: Create the Target Governance Model

The governance model aligns the specified services with specific entities and attributes in terms of inserting new rows, updating attributes and deleting rows. If more than one service can perform these functions, consideration must be given to their coordination in order to avoid conflicts.

The governance model provides input into the canonical message model. The canonical message model must consider the data usage requirements for each service. This includes functional requirements, that is, the specific attributes needed. It also includes non-functional requirements. One of these is whether or not

data is needed near real time. Another is whether or not data is needed as of a specific point in time. These non-functional requirements affect decisions regarding of the means used to implement information integration in the realization phase.

5.1.4 Deliverable

The logical data model typically will be fine-grained and is intended to show the detailed relationships that occur between the specified entities. It is intended to be granular enough to account for all of the attributes needed to specify services at a detail level.

The recommended notation for this work product is entity relationship modeling supplemented by discussions of the reasoning behind the ER models. There are several "accepted" variations of this notation, all recognizing the same basic constructs. The most common is adopting the information engineering approach but choice may be influenced by tools available on the project. The names used in the logical data model are defined in the business glossary. Additional annotation should be provided wherever possible to add clarity and assist understanding of the diagram.

The conceptual data model will usually comprise several separate, but related, components:

- A detailed level entity relationship diagram (ERD) showing the major entities and their relationships
- Information domain level ERDs that show the entities in each information domain and the entities shared between information domains. For example, Product is shared between the Customer information domain and the Material Management information domain
- Entity, attribute, relationship and information domain definitions based on the business glossary
- A CRUD matrix of entities and attributes to services

5.1.5 Dependencies

The primary inputs to the creation of the logical data model are:

- The business glossary
- Specified services including functional and non-functional information requirements from the service model
- The conceptual data model
- Existing metadata including standard industry data models

All of these need not be complete to start developing the logical data model. The model can be started and developed as these other deliverables become available. However, the model cannot be considered complete until all of the dependent work products are completed and reflected in the logical data model.

5.1.6 Example

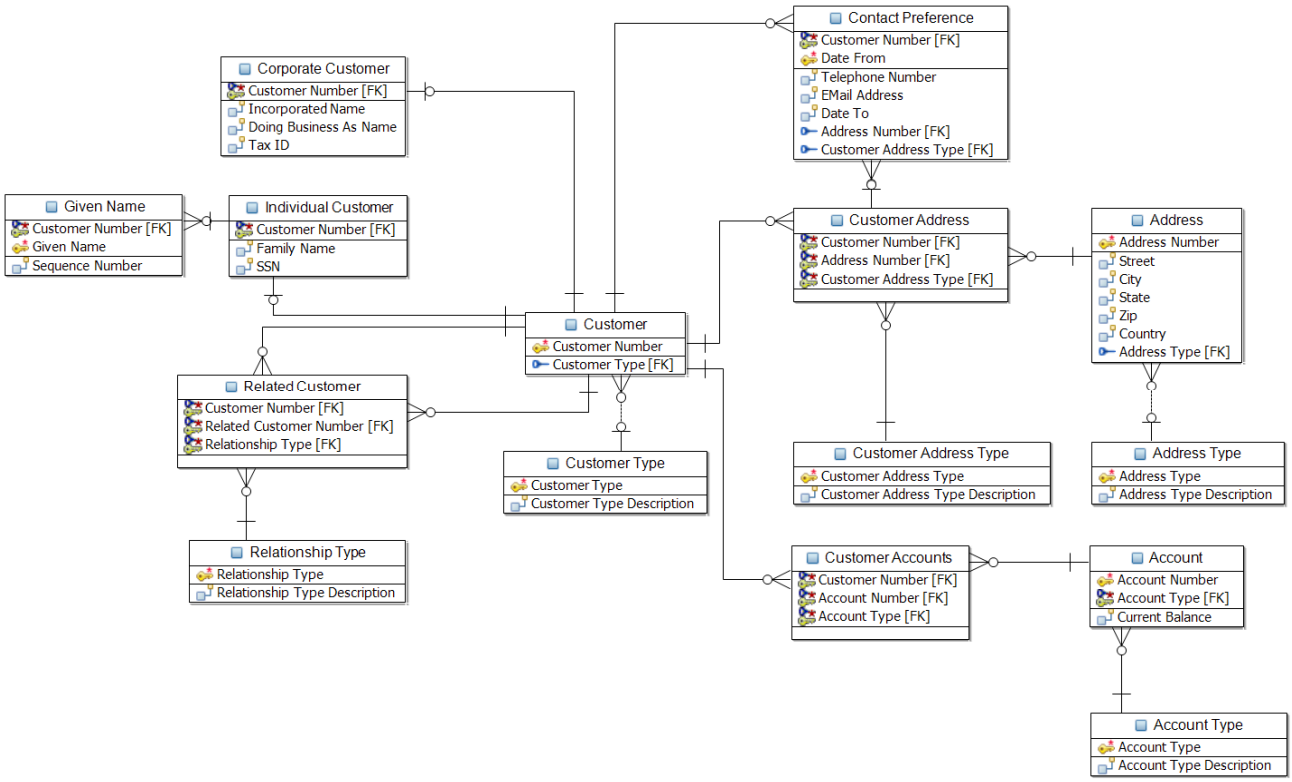

Figure 15: Example Logical Data Model

5.1.7 Guidance

Regardless of the tool used to develop the logical data model, it is important that the conventions used in the model are well defined. The following are some conventions that may be used in creating a logical data model. The important thing is that the all conventions be well documented.

- Specify entity definitions and attribute definitions in the data modeling tool whenever possible. This is especially important when the name doesn't make the definition self-evident.
- Supply domains of values for "type" entities. For example, if there is an entity, Customer Types, specify the valid values as part of the definition.
- Provide standards for names. An example is: entity names are plural. Attribute names are singular. Surrogate keys are suffixed with ID.
- If a logical data model will be used to generate a physical data model, there may be restrictions on the length of the column names. This means that complete logical names may be truncated in the process of generating the physical model. To plan for this use a standard set of abbreviations in the logical model. If abbreviations are used, use them consistently throughout the model. For example: Security_Master is less than the 18 characters allowed for a DB2 column name; Security_Price_Interval is more than 18. Both should be abbreviated as: Sec_Mast and Sec_Pric_Int.
- A subtype entity must have a discriminator in the super-type. It is represented in the model by a one-to-one optional relationship.

- Resolve all many-to-many relationships into associative entities. An associative entity may be named in one of two ways. If the relationship is general in nature, it is named “nnnnn To xxxxx”. This is an association between two entities that can occur for more than one reason. In this situation, the nature of the relationship should be part of the key. This is the relationship role. An example is Related Parties. Two parties can have both a vendor relationship and a customer relationship. In order to manage the customer relationship, it may be essential to consider the vendor relationship as well since it can be a form of leverage.
- Model relationships that can be open-ended as recursive. If the relationship is recursive in nature, it is named “Related *nnnnn*”. Examples are Related Parties and Related Agreements.
- Divide the main data model into information domains that reflect clusters of related tables (e.g. Orders, Events, etc.) or connections between those clusters (e.g. Shipping and Billing). An entity can occur in multiple information domains but only once in the model. Preferably these can be printed to no less than 60% reduction on 81/2” X 11” paper. This makes it easier to explain and review the model since the focus is reduced to a specific topic and printing is more practical.
- Derived attributes may be included in the data model to illustrate concepts to the solution developers. The definition of a derived attribute should include the derivation rules. The decision whether to store a derived attribute or to derive it at execution time is made during physical design.

5.2 Updating the Business Glossary

As the initiative progresses from the identification phase into the specification phase the document will start to become more mature and refined. It will start to have actual agreed up definitions of the final terms and if he documents started out as business glossary of individual sources as we flow into the specification phase the document should start to become more specific and deal with the actual candidate terms and the attributes of the final target terms. The basic physical structure of the artifact will not change, whether or not the artifact is in document form or captured in a tool. The refinement will start to emerge.

The business glossary artifact should include aspects of the newly developed logical data model or new target model being considered. This will mean that certain terms are now composite terms and/or have been deleted or refined.

5.2.1 Definition

The definition of the artifact for business glossary does not change from the definition in SOMA identification but adds additional data and perhaps includes aspects of the actual specification of the initiative. At this point we have progressed past the service identification phase into starting to specify the actual candidate entities and services that will ultimately be usable in SOMA realization.

5.2.2 Approach

In the specification phase there is not necessarily any new information that needs to be gathered it is a matter of maintaining and updating to reflect changes or refinements made as the conceptual model progresses to the logical model and more and more attributes are defined. The artifact itself should have started out with place holders for information or attributes not known in the identification phase and now that the project is in the specification phase any updated or newly collected/defined information should be updated in the business glossary to reflect this progression.

5.3 Commence Data Quality Analysis

For SOA services to be successful and reusable it is important that the data they expose is of acceptable quality for all the consumers.

5.3.1 Definition

Data quality in SOA can be considered in two dimensions:

- The **technical data quality dimension** defines the data quality criteria often found in both the entity integrity and referential integrity relational rules found in logical data modeling.
- The **business-process data quality dimension** defines the understanding of the key data quality elements in terms of what the business definition for a data quality element is and what the business rules are associated with that element.

Both of these data quality dimensions influence SOA service and process design. Data quality analysis quantifies the characteristics of operational data in the context of its intended use. Understanding these characteristics enables the service designer to be confident that each service can meet its service level requirements and in some cases will identify issues that require design changes in the service or the underlying systems to ensure these objectives are met.

This paper details the approach to investigate and qualify data characteristics in these two quality dimensions and provides normative guidance.

5.3.2 The Causes of Bad Data Quality

During SOMA specification, a set of services is identified for which we want to formally specify and make the necessary preparation steps for their realization. We must understand if the data that we may need to expose through a service satisfies the business requirement. For example, to specify a service ‘retrieveFullCustomerDetail’ for which the data resides in multiple repositories, it is important to understand how to integrate the information from the various sources. Can we access each data source and simply join

the resulting information or will there be problems with this approach? Answering this question gives us important information for the realization of the service.

Figure 16 shows some typical data quality problems that can exist and that can complicate service design.

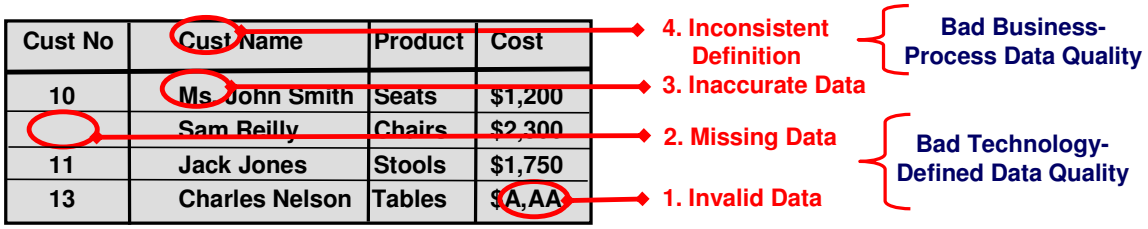

Figure 16: Examples of Data Quality Problems

Poor data quality's affect on the ability to make sound business decisions can be found both at the technical level and business level of data definition.

Technology-driven data quality issues are those caused by not applying technology constraints either at the database or during data integration. These include:

- **Invalid Data:**
  By not applying constraints e.g. alphanumeric data is allowed in a numeric data field (or column)
- **Missing Data:**
  By not applying key constraints in the database e.g. a not null field has been left null

Business-driven data quality issues are those caused by end users inaccurately maintaining data. Examples include:

- **Inaccurate Data:**
  By inaccurately creating a record for "Ms. Anthony Jones", rather than "Mr. Anthony Jones", bad data quality is created. Inaccurate data is also demonstrated by the "duplicate data" phenomenon. For example an organization has a customer record for both "Anthony Jones" and Tony Jones", both the same person.
- **Inconsistent Definitions:**
  By having disparate views on what the definition of bad data quality is, bad quality can be experienced through incorrect use of data. This is especially true in SOA where the business context of the service consumer is not necessarily known by the user or application creating the data.

To investigate data quality for an SOA project the data analyst must have a good understanding of the full scope of both technical and business data quality issues and how they can impact the SOA solution.

5.3.2.1 The Technical Data Quality Dimension

The technical data quality dimension defines the data quality criteria often found in both the entity integrity and referential integrity relational rules found in logical data modeling. Key aspects of this dimension are:

TECHNICAL DATA QUALITY DIMENSIONS		
Name	**Description**	**Examples of Poor Technical Data Quality**
Valid	Data element passes all edits for acceptability	A customer record has a name that contains numbers. Social Security Number field should be numeric integer but is populated with alphanumeric characters instead.
Unique	Data element is unique —there are no duplicate values	Two customer records have the same social security number.
Complete	Data element is (1) always required or (2) required based on the condition of another data element	A product record is missing a value such as weight. Married (y/n) field should have a non-null value of 'y' or 'n', but is populated with a "null" value instead.
Consistent	Data element is free from variation and contradiction based on the condition of another data element	A customer order record has a ship date preceding its order date.
Timely	Data element represents the most current information resulting from the output of a business event	A customer record references an address that is no longer valid.
Accurate	Data element values are properly assigned. E.g. Domain ranges.	A customer record has an inaccurate or invalid hierarchy.
Precise	Data element is used only for its intended purpose, i.e., the degree to which the data characteristics are well understood and correctly utilized	Product codes are used for different product types between different records.

Table 2: Technical Data Quality Dimensions

With these dimensions, technical data quality rules are applied against key data quality elements as shown below.

Customer Data Warehouse Data Quality Work Book													
			Technology Quality Rules							Business-Process Quality Rules			
Id	Data Object or Domain Area	Data Element	Valid	Unique	Complete	Consistent	Timely	Accurate	Precise	Enterprise Definition	LOB Definition 1	LOB Definition 1	LOB Definition n
			Data element passes all edits for acceptability	Data element is unique —there are no duplicate values	Data element is (1) always required or (2) required based on the condition of another data element, e.g. Primary Key	Data element is free from variation and contradiction based on the condition of another data element	Data element represents the most current information resulting from the output of a business event	Data element values are properly assigned. E.g. Domain Ranges	Data element is used only for its intended purpose.	The data element has a commonly agreed upon ***enterprise*** business definition and calculations			
1	**Customer**	Customer Number	**Integer only**	**Non-Repeating**									
2		Customer First	**Not Null**										
3		Customer Last	**Not Null**										
4		Gender			**Not Null**			**Either "M", "F", or "U"**					

Figure 17: Example of Technical Data Quality Assessment

5.3.2.2 The Business-Process Data Quality Dimension

The business-process data quality dimension defines the understanding of the key data quality elements in terms of what the business definition for a data quality element is and what the business rules are associated with that element.

The problem with business-process data quality definitions are that many organizations have inconsistent definitions and different business rules for similar data with each line-of business, each having its own understanding of what that element is:

- Data meaning is always dependent upon the context of a the business domain;
- Data meaning is always dependent upon the context of an application;
- Data meaning is always dependent upon the choices made by the designer;
- Data meaning is always dependent upon the usages and practices of end-users;

And this varies from an application to another. For example:

- Marketing definition of Net Assets = Assets – Expenses
- Finance definition of Net Assets = Assets – Expenses + Owners Equity

Hence, with disparate views on what the definition and business rules of a data quality element is, when information is compared from different Line-Of-Business (LOB) the perception of bad quality is created.

BUSINESS PROCESS DATA QUALITY DIMENSION – SINGLE DATA SOURCE		
Name	**Description**	**Examples**
Definitional	The data element has a commonly agreed upon ***enterprise*** business definition and calculations	”Revenue” is calculated using different algorithms/equations and using different source data for each departments within an enterprise.

Table 3: Business Process Data Quality for a Single Data Source

The perception of bad data quality can lead to the reality of bad data quality when misunderstood or incorrectly used data definitions are used to build integrated information services joining data from multiple data sources. The following table shows a checklist of the semantic interoperability conditions to consider before exposing information through services in an SOA solution.

BUSINESS PROCESS DATA QUALITY DIMENSION – INTEGRATED DATA SOURCES		
Name	**Description**	**Examples**
Common Definitions	Joined data elements with the same name have a commonly agreed upon enterprise business definition and calculations.	Aircraft company sells airplane to “customer” and records that information in the sales system. Aircraft company services the plane and stores the “customer” in the engineering system. If the airplane is leased by an operator then it is quite possible that the two systems have different values of “customer” for the same airplane. A consumer of a service returning the “customer” for a specific aircraft must understand the business context under which the customer is being provided.
Overlapping Populations	Interoperability is possible between two systems sharing a partially common population. Sometimes the cross-constraints might be unsustainable.	Internet sales system stores list of customers who have made a purchase through the website. Retail sales system stores list of customers who have purchased products supported by after sales warranty. Accounting system stores list of all customers holding a fidelity card. These 3 sets are partially overlapping but each can contain customers that do not exist in the others. The services and processes consuming this data must understand the population gaps and overlaps to meaningfully use the data.
Comparable Generalization Levels	Interoperability is possible between two systems sharing a partially similar generalization level. However the mapping rules must be clearly defined to avoid potentially costly impedance mismatch errors.	Education institute stores “PERSON” details in one system but subtypes these in “STAFF”, “STUDENT” and “FACULTY” in another system. Services and processes must understand how to navigate this subtype relationship to accurately combine data taken from the different systems.
Comparable Aggregation Levels	Two data systems may use the same data structure to store semantically different representations of the same real life entities.	1. Retail distribution company stores each customer as a single financial entity in the accounting system but as multiple physical retail locations in the order management system. Integrating information from these two systems requires understanding how this aggregation is resolved. 2. Manufacturer uses the CUSTOMER table to store both upstream and downstream supply chain partners. In some cases the same partner appears multiple times to represent the different relationship roles that exist between them. Services and processes accessing this data must understand the rules that differentiate the reasons why an entity exists in this data store.

Semantic Drift	Unplanned modification of data meaning caused by non-compliant or non-controlled use of operational data.	Scenario: Obsolete customers cannot be removed from operational system because no suitable cleanup exists for orphaned records. Instead the users "flag" obsolete customers by prefixing the customer name with "EXPIRED – DO NOT USE: customer name". A service process not understanding this creative use of the customer name field will return these records in query result sets.
Synchronization Mismatch	When two source data systems are synchronized outside of a shared transaction boundary then there exists a timing window during which a service accessing both systems may encounter mismatched data.	Customer has credit limit in accounting system which is updated nightly based on orders placed. If order process uses a service from the accounting system to check available credit before accepting an order then the result will not include other orders already placed that day. The result is that the process can allow orders that exceed the customer credit limit.
Agreed scale, Codes and units	Scales, units and measures, as well as type codes (such as country codes, part numbers, etc.) must be matched or mapped for all different sources providing data to a service or process.	1. Multinational company sells products using metric units in Europe but imperial units in USA. Analyzing sales volumes by unit may be misleading is measurements are not mapped. 2. Joining two data sets sorted by country code will be meaningless if the country codes are not correlated.

Table 4: Business Process Data Quality for Multiple Data Sources

Applying a commonly agreed business definition and rules against the data elements provides insurance against inconsistent data quality issues. Building an enterprise wide business glossary and conceptual data model provide the foundations for getting consistent standards and rules for shared data elements and entities across the enterprise.

5.3.3 Objective

The objective of data quality analysis in SOA is to establish a detailed understanding of the operational data exposed by services. The level of data quality required to effectively support operations will vary by information system or business unit, depending upon the information needed to conduct that business unit's operations. For example, financial systems require a high degree of quality data due to the importance and usage of the data, but a marketing system may have the latitude to operate with a lower level of data quality without significantly impacting the realization of the granular strategic objectives attributed to its stakeholders. Since the "purpose" varies, so does the bar that is used to measure fitness to purpose.

The objective of this paper is not to set the criteria by which data is deemed "good enough" to support the business need, but is rather to explain the techniques and approach which can be used to quantify the data characteristics so that an objective decision can be made in this regard.

Similarly, there are **preventative** data quality approaches that focus on the development of new data sources and integration processes, and there are **detective** data quality approaches that focus on identification and remediation of poor data quality. This paper will deal only with the **detective** approach to data quality and within that approach the focus is identification of data issues rather than their correction. A more holistic

approach to data quality detection, correction, prevention, and ongoing monitoring is outside the scope of most SOA projects and will not be elaborated here.

5.3.4 Value

The value of performing data quality analysis during SOA analysis and design is:

- To clarify and confirm the data environment in which the SOA services will operate.
- To validate that proposed service designs will meet their functional and non-functional requirements regarding the data that they expose.
- To identify and quantify data issues that could inhibit a service's ability to meet requirements. Understanding the exact nature of these data problems then enables the service designer to make informed choices of how best to mitigate these issues.

The impact of not having this work product is that the success of the project depends upon unknown quality of data which can delay the project (in order to address the issues) or even risk the launch of the project. Without performing data analysis the service designer is dependent on opinion and anecdotal evidence of local experts to support design decisions regarding data quality. If data is already well understood in the proposed context under which it will be used in the SOA environment then this may be adequate. However, any undiscovered issues at design time will surface in testing or even after deployment making corrective action much more expensive to the business and the project.

The reasons for not needing this work product are if the quality is already high, or well understood, or if the expectations are low. The required depth of data analysis is subjective and varies across different business entities in the project. There are cases where little or no analysis is required. These include:

- Adequate data analysis information is already available.
- The data is well understood and will be exposed through an already established interface that has widespread business acceptance. For example, the data will be exposed through a packaged application API that is already in use for business process integration for the organization.
- The business expectation for the data quality is low. The business is willing to take the data "as-is" with the knowledge that there are few guarantees about the inherent quality of the data.
- The data is being accessed from a system or supplier over which the project has no access for analysis and/or no influence to make changes.

5.3.5 Approach

Approaching data quality analysis for an SOA project has many similarities to approaching data quality analysis in any other project. However, before outlining the step-by-step approach in 5.3.5.2 it is worth first remembering the SOA context under which the data is being analyzed and how data analysis compliments the other analysis and design activities in SOMA.

5.3.5.1 Data Quality Analysis for SOA in a SOMA project

The critical component in SOA is the service interface because it is the service input and output (i.e. the message model) which present the public view of the underlying data to the consumers. There may or may

not be a direct relationship between the data elements exposed on the service interface and the underlying physical data stores. Consider the following illustration:

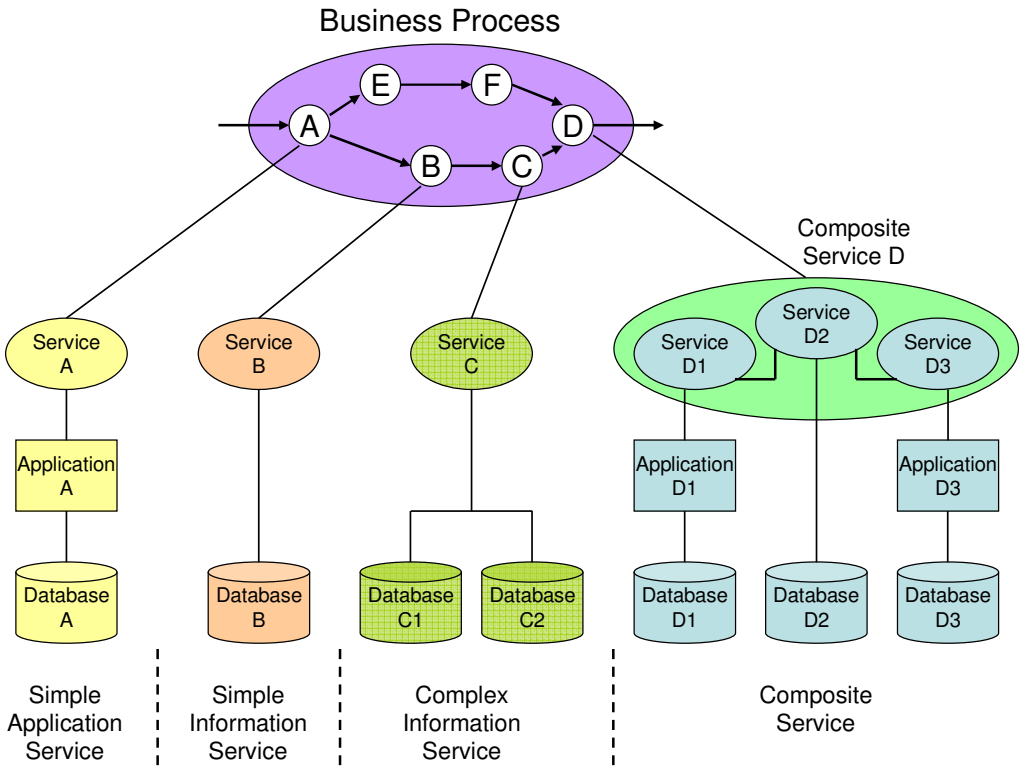

Figure 18: Mapping a Service to Data Stores

Here we see examples of the different types of mapping between service interfaces and physical data stores.

- **Simple Application Service**

  The implementation of Service A is as an application API wrapper. The data exposed on the service interface comes from the underlying database via the application code. Understanding the data quality for this service requires that the analyst first understands how the service interface maps to the application API and then how that API accesses the underlying data. With this knowledge the correct data elements can be identified and analyzed. Depending on the complexity of the business logic, it may not be possible to assess the quality of the information on the service level just by understanding the quality of the information on the persistence layer.

- **Simple Information Service**

  The implementation of Service B is a simple data access directly to the physical database. For this scenario the mapping of the service interface to the physical data is all that is needed to identify which data elements should be analyzed.

- **Complex Information Service**

  The implementation of Service C is a data integration access to related databases. For this case the data integration rules must be identified and the mapping onto the service interface understood. With this information the data analyst can determine which data elements should be checked for data quality.

- **Composite Service**

  Service D represents the most complex case which is the composite service. Here data flows through service composition logic, service implementation logic, and potentially application and data integration logic. All the rules and mappings of these components must be understood for the analyst to identify the correct source data and rules needed to analyze the data quality.

Given the potential complexity of each service implementation a structured approach is needed to understand where data quality analysis should be applied in any SOA project. This structure comes from the data modeling activities which are performed during service identification and specification. This process is outlined in the following diagram:.

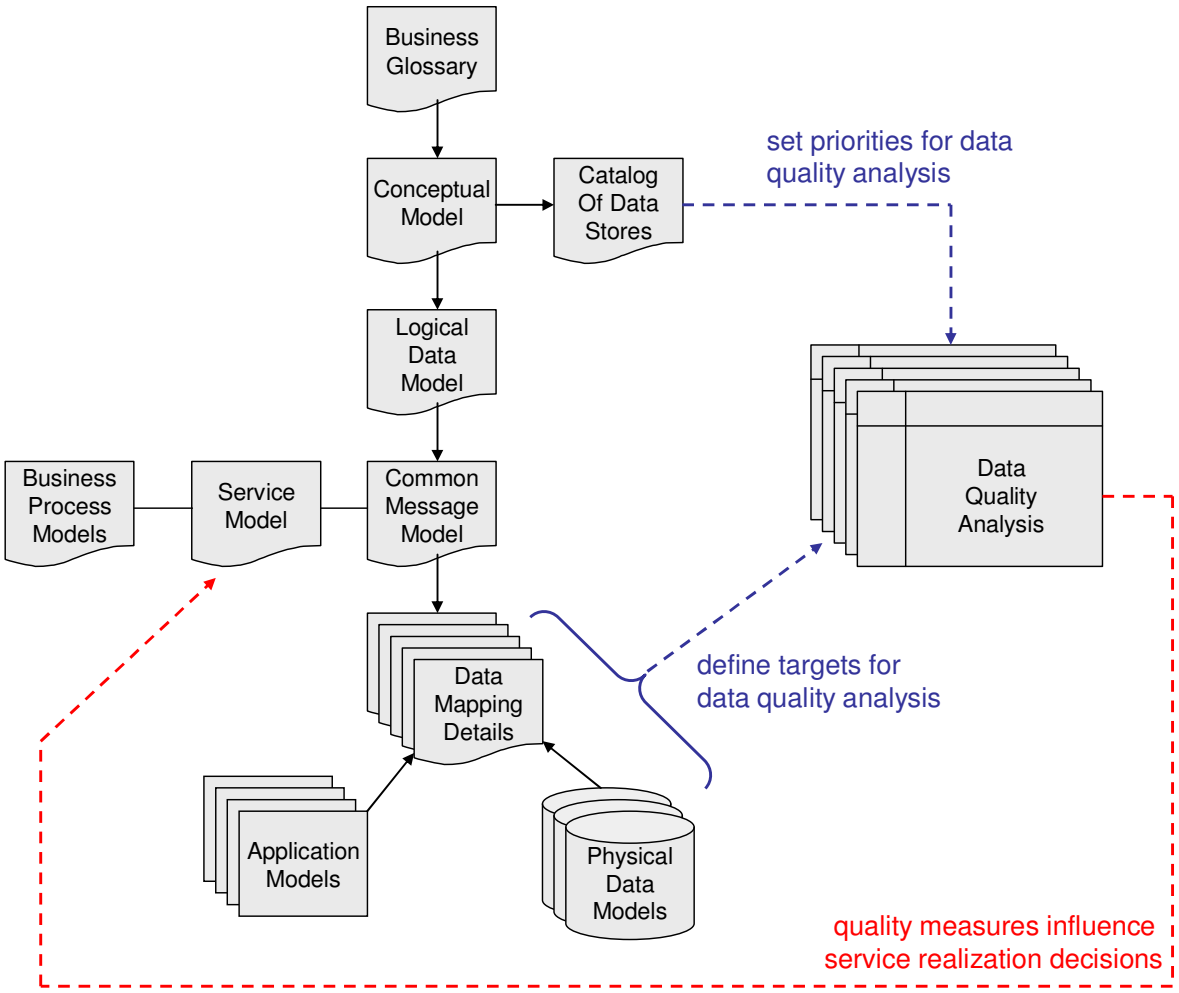

Figure 19: The Role of Data Quality Analysis in SOMA

In SOMA the service design lifecycle is very iterative but from a data design linage perspective we can see following sequence of activities:

- Business terms are defined in the business glossary and these terms set the shared business definitions for terms used across all other design artifacts.
- The high level organization of business information entities is captured in the conceptual data model and this structure is then extended into the dependent work products.
- The conceptual data model is elaborated to include detailed attributes and relationships in the logical data model. This model becomes the data reference for high level business process and service modeling.
- As specification detail is added to the service model we develop the canonical message model which represents the physical instantiation of the logical data model.
- A high level mapping of conceptual entities to physical systems is built in the catalog of data stores work product along with subjective assessment of data quality and complexity. This is used to identify the priority data elements for quality analysis.
- If a mapping exists between the message model or logical data model and the various existing data stores then it can be leveraged to further narrow down the scope of the data quality analysis. Those mappings can be used to identify the mapping/integration rules needed to explore the data quality characteristics within the context of the service model.
- The results of the data quality analysis are then fed back into the service model to validate realization decisions or identify cases where alternative design decisions are needed to achieve target levels of data quality.

So before embarking on a data analysis project plan remember these key points:

- Data quality is being assessed against the service interfaces upon which it is exposed.
- The primary objective of data quality analysis is to arrive at a yes/no answer to the question: “Will the planned service implementation deliver data quality to the level agreed with the business consumers?”

5.3.5.2 Data Quality Analysis Approach

The data quality analysis approach includes the following steps:

1. Define the scope
2. Review existing data quality information
3. Define project level data requirements
4. Identify Data Quality Criteria
5. Review/Augment Data Quality Criteria
6. Design Data Analysis Queries
7. Identify Target Data Sets and Samples
8. Execute Data Quality Analysis
9. Analyze and Report Results

The steps are described in more detail in the following sections.

Step 1: Define the scope

The service model is the final definition of what data from the enterprise is in scope for data analysis. Every element of information in the service model and every data store to which it can be mapped in implementation is a candidate data element for data quality analysis. The first activity in data quality analysis is to reduce this scope definition from what "can be" investigated to what "will be" investigated. This is broken into 3 complimentary tasks:

1. Review the service model and the catalog of data sources to see already established decisions on data quality and use these to prioritize data quality analysis.
   This can commence even during the development of the catalog of data sources but for more complex service implementations may require that data modeling, message modeling, and data mapping are at least partially complete before data quality analysis can be started.
2. Refine the scope based on SME discussions.
   Use facilitated sessions with business and IT subject matter experts to identify critical entities and data elements. Use the following questions to assist in the prioritization of this effort:
   - What is the overall importance of this entity / element in the known services and business processes?
   - What is the impact of not having this entities / element?
   - Are you willing to add staff to review / process exceptions associated with this entity / element?
   - What is the importance of this entity / element in downstream processes?
   - What is the legal risk associated with this entity / element?
   - What is the regulatory risk associated with this entity / element?
   - What is the financial risk associated with this entity / element?
   - What is the customer service risk associated with this entity / element?
   - What is the decision risk associated with this entity / element?
3. Refine the attributes in scope
   Not all defined columns, fields, and elements are relevant to data quality, only those that affect the structure and understanding of information. The data analyst must work with business domain experts and application experts to assess which elements are more meaningful than others in order to identify how data quality will be analyzed. Consider Figure 20

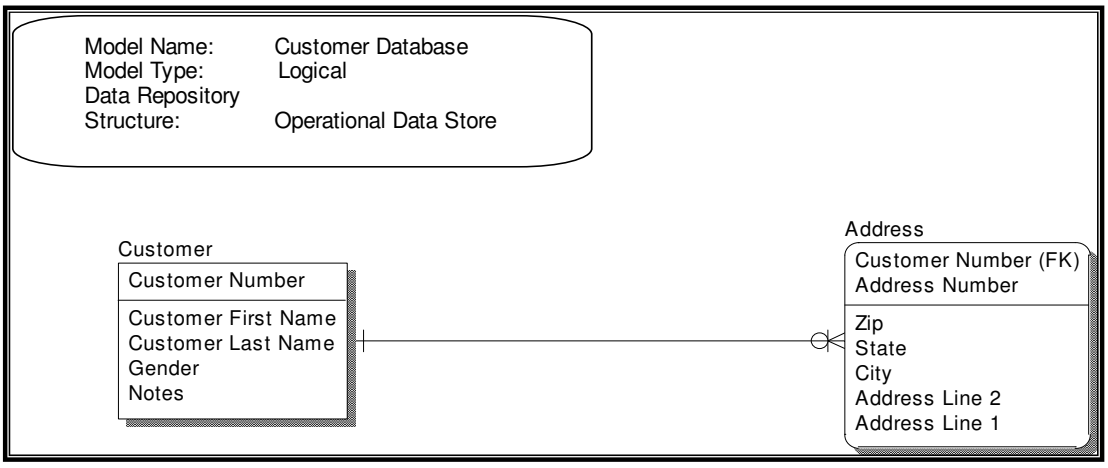

Figure 20: Example Shows which Attributes to Include / Exclude for Data Quality Analysis

In the customer entity for data quality purposes, the “Notes” attribute will not affect data quality; therefore this attribute will not be identified with any data quality rules. Again, only those attributes that affect the structure and understanding of the data will be captured and quality criteria determined.

Step 2: Review Existing Data Quality Information

In this task, the existing data quality information for the intended project’s data elements and entities is reviewed. The following best practices can be applied to this task:

- Review the SOA service model, logical data model and canonical message model to understand the data elements and entities in scope for the project.
- Review the catalog of data sources and any detailed mapping to physical data stores to understand the physical data structures and any implied data integration necessary to understand the context of the business data quality requirements.
- Review the catalog of data sources for data classifications to understand the areas most likely to have data quality issues.
- Work with business and technical subject matter experts to determine whether or not any relevant metadata exists and review if available.

Step 3: Define Project Level Data Requirements

In this task the high-level data quality requirements for a potential assessment are determined. The following best practices can be applied to this task:

- Determine if any existing data quality criteria already exist for the data elements and entities in scope and if they are applicable to the SOA requirements.
- Review the functional and non-functional requirements defined against the service model. Use these as input to create a starting list of data quality criteria organized by logical entity.
- Determine gap between current data stores data quality criteria and potential project criteria additions. Determine number of potential new critical elements. Use this information to assess potential project data quality risk.
- Determine data quality requirements based on completeness of existing data quality criteria, and potential new critical data elements and data stores.
- Gather record counts, field sparseness, and potential critical element counts for each data source to use in determining sample sizes.

Step 4: Identify Data Quality Criteria

In this task, the data modeler identifies the data quality criteria in the logical data model. They identify the critical entities and data elements, the domain values, and business rule ranges. A summary of technical and business process data quality issues for SOA is given in 5.3.2. The following best practices can be applied to this task:

- Use facilitated sessions with business subject matter experts to identify critical entities and data elements. The following questions can help guide these workshops:
  - What is the impact of bad data on this element?
  - Can it still be used?
  - If it contains bad data can it be cleaned up?

  - Can clean-up happen during runtime execution or is it an offline task?

- The catalog of data stores includes a technical view of likely data quality and data integration issues. This can now be cross referenced against the business assessment of data importance identified from the previous step. The combination of these two factors can be used to determine priorities for the following data quality analysis activities.
- Use follow-up facilitated sessions with business and IT subject matter experts to determine the data quality criteria and refine list of critical entities / data elements. If available, profiles of source data for critical entities / elements would be helpful.
  - What is the target level of data quality required for this entity / element? Examples for entities include expected record count and tolerance for duplicate records. Examples for elements include tolerance for scarcity (nulls) and valid dates. Examples for data integration include missing, duplicate or unresolved record matches for deterministic matching or matching success rates for probabilistic matching.
  - Should/can this element be combined with any other elements to determine its metric?
  - What are the business impacts of this entity / element falling below the target metric?
  - If the quality of the entity / element is below the target is the element still usable? If so what is the business impact and recovery process when bad data is encountered.

Step 5: Review/Augment Data Quality Criteria

In this task the application DBA reviews the physical data models to ensure completeness and accuracy of data quality criteria that was extracted from the logical data model and canonical message model mapped to physical data sources. It includes the review from a data quality perspective of any additional entities, attributes, and relationships added for the physical model and the database-specific augmentations. The same best practice listed under Identify Data Quality Criteria can be applied to the data elements added or updated in this task.

Step 6: Design Data Analysis Queries

From the data quality criteria identified, the DBA must now design queries and/or configure data profiling and analysis tools for examining the data and generating quantitative results. Details will be tool specific but examples of analysis are:

- quantifying column and table statistics: number of rows, number of unique values, number of missing values, precision, skew and ranges.
- validating/quantifying primary key constraints for nulls and uniqueness
- validating/quantifying column values for business rule compliance
- validating/quantifying primary-foreign key constraint compliance
- validating/quantifying data matching rules
  - number of matches
  - number of unmatched records in each source data set
  - conflicting values in matched data sources

Step 7: Identify Target Data Sets and Samples

In this task the DBA reviews the set of data analysis queries developed in the previous steps and identifies target data sets against which the queries will be run. For accurate results the queries will be run against full production data but there are many reasons which may make this impossible:

- operational systems cannot support the additional workload.
- privacy and security rules restrict access to the data
- data belongs to new systems or external systems for which no sample data is available
- data volumes are too large to query in the available time
- data store technology does not support available data analysis tools

In such cases the DBA must build representative data samples from existing test data, generated data, or sampled production data. If this is done the results must be reviewed carefully to understand which problems are genuinely representative of the corresponding production data.

Step 8: Execute Data Quality Analysis

In this task the queries are executed and results are gathered. Details will be tooling dependent.

If execution takes place over extended periods of time then care must be taken to ensure no false errors are introduced by timing and data synchronization issues.

Step 9: Analyze and Report Results

In this task, the results are compiled and summarized for presentation and discussion with the business and project team. The format will be tool dependent but will most often consist of summary tables and graphs from which it is possible to drill down into the supporting detail data.

Where results fail to meet the identified requirements, or in cases where results conflict with preconceived expectations it may be necessary to iterate over the tests or add supplementary tests to explain the results.

Where significant problems are discovered there are three options available to the SOA designer:

- Change the service implementation to mitigate the data problems.
- Request that a change is made outside of the SOA project team. This may be a catalyst for initiating an enterprise data quality initiative in the organization if one does not already exist.
- Choose not to expose the service on the grounds that it cannot meet business service level requirements.

5.3.5.3 Ongoing Monitoring of Data Quality

Initiating an ongoing data quality cycle is outside the scope of the SOA project. The analysis in this document may motivate a separate initiative to address ongoing data quality monitoring and preventative data quality practices.

An exception to this would be if there are cases in the project where specific aspects of data quality are identified as critical KPIs for the SOA solution. In these cases it may be appropriate to encapsulate the data

quality tests as services that collect and report on specific aspects of data quality. The results can then be consumed by business processes or displayed on SOA management dashboards.

5.3.5.4 Final Considerations for Data Quality

- In any cost benefit analysis for a project charter or scoping document for any data quality initiative should include the cost of not performing the data quality tasks. In data quality projects the “ounce of prevention is usually worth the pound of cure.”
- Data quality related to the accuracy with which the data reflects reality. An organization’s actions, if based on a “flawed reality’ may create costly mistakes for themselves.
- Organizations need to recognize that not all data is relevant and assess what data is critical to their operations. Focusing on this “critical” data allows an organization to assess the quality of its data without overwhelming the organization.
- Data should be treated with the same respect as any other corporate asset. It should be protected and impacts to it should be analyzed for risks to the organization.

5.4 Create a Canonical Message Model

5.4.1 Definition

The **canonical message model** work product represents the standardized format used for exchanging business information in the SOA. Not every message passing through the different layers of the architecture will necessarily comply with this model, but rather the model provides the default business data interchange formats so that all components need only to know (at most) their own message format and the default message format.

The most common representation used for the canonical message model is as a set of XML schemas. This has the benefit of making the type and message definitions directly reusable in the WSDL schemas that describe the exposed services.

The canonical message model consists of the following:

1. A defined set of types, elements and attributes representing the business entities and their business attributes used in all messages. Each definition includes:
   - Technical data types, formats, structures and names.
   - Rules governing the allowable values of the type.
   - Business semantics of the type.
2. A defined set of messages, each including a related set of the previously defined types, elements and attributes structured to provide a business document with a specific semantic meaning and context.

The canonical message model is based on the logical data model and business glossary. The canonical data model describes the business entities, attributes, and relationships, in a normalized form structured to reflect their business use. However, many entities have direct and/or indirect connections to many other entities via the relationships represented. When exposing a service interface, a decision must be made to limit how these entity relationships are exposed such that sets of high value and highly reusable information are exposed within the context of the business consumers. Failure to adequately restrict the depth of each entity relationship graph exposed can easily lead to proliferation of unnecessary information being passed around the SOA architectural layers.

In addition, the canonical message model is designed to support flexibility and extensibility such that the evolving business requirements on the architecture can be easily accommodated. Industry standards and variation analysis can be used to optimize the chosen formats. Implementation through XML schemas provides both the strong data typing rules and flexible structures needed to meet this goal.

Each service exposed in the SOA solution should have input and output messages which are defined directly by the canonical message model or which have a clear and explicit mapping to the canonical message model. This ensures structural and semantic interoperability across all the components participating in the solution.

The canonical message model does not define technical metadata. It only defines business information. It is common to include technical metadata in the messages passed between systems. Typically, this technical detail can be isolated to message headers so as not to corrupt the business information in the message. Such techniques are common for handling security credentials, transaction states, routing information, message and service versioning etc. These metadata can be defined as enterprise messaging standards but should be kept separate from the business information structures and semantics expressed in the canonical message model.

5.4.2 Objective

The core of SOA is the shared service definition. In practice this means the WSDL schema which describes the functions provided by the service (as a set of operations with request and response messages) as well as binding details for how to find and call the service. Each WSDL schema can be created independently as a self contained file, but a better approach in a large SOA project is to build WSDL schemas by composition from a set of smaller reusable files. This adds consistency and maintainability as the system grows. When using such an approach the canonical message model provides the data and message definitions that form some of the building blocks for the WSDL schemas.

The objectives for the canonical message model are:

- To define a business information interchange language that is at the same time strongly typed as well as flexible/extensible. Each message or message element is clearly defined for both structure and semantic meaning within the business context for which it is intended to be used.
- To align the information exposed in SOA messages with the accepted business view developed in the logical data model. The resulting business data interchange formats and semantics will be useful and meaningful across a broad set of service providers and consumers in the enterprise.
- To accelerate the development of new messages by providing a standard set of basic business information types shared by all messages.
- To reduce integration effort by providing the default syntax and semantics for all information exchange.

- To reduce the complexity and frequency that data mapping rules are required to allow different SOA components to effectively communicate.
- To accelerate the definition and design of services by providing a set of reusable message constructs from which service interfaces can be composed.

5.4.3 Value

Without this work product, the SOA team cannot easily integrate the different components making up the SOA solution because:

- Each message will be defined strictly within the context and requirements of the message provider and consumer. This can lead to proliferation of messages in the solution which offer little potential for reuse and carry a high maintenance cost.
- Inconsistent message types, formats, and semantics in the solution require that for each new integration scenario there will be a need to analyze the participants and develop message maps to address inconsistencies.
- Development of new messages, services and business processes will be slower because each designer will be making new decisions about handling the business data.
- Detailed variations analysis to determine the impacts of future changes on the SOA solution is not possible.
- The accuracy of the estimation of subsequent projects may be affected.
- There is a potential for delay in follow-on projects.

This work product may not be needed for cases where:

- An enterprise message model already exits (as can be the case when an enterprise complies directly with a published industry standard or has purchased a packaged solution that prescribes the data interchange formats).
- The scope of the engagement is limited such that detailed analysis of service interfaces will not be performed. This might be the case when the focus of the SOA project is deployment of infrastructure or integration of pre-existing or outsourced services.

5.4.4 Approach

The recommended notation for this work product is a set of well annotated W3C compliant XML schemas. The reason for this is that the primary use of the work product is as a set of message definitions that can be reused in the WSDL service interface definitions using `wsdl:import` or `wsdl:include`.

The design and specification of all XML schemas should be rigorous. When creating an XML schema you should be working within the project development process under design guidelines and coding standards (e.g. naming conventions, version control, build/test/release controls etc.). The XML schemas should be reviewed for accuracy and compliance with guidelines and standards. Each of the requirements in the list below is a general requirement for the XML schemas describing the canonical message model.

- **Understandable**: The XML schemas should be clear, consistent and unambiguous. They should contain human readable documentation and, where possible, link to dependent design documents such as the service model, logical data model and the business glossary.

- **Semantically complete**: The XML schemas should define, for one or more target XML message definitions, each and every element and attribute that is understood by the solution when processing those messages. For example, if your application uses the value of an attribute or element then a definition (both structural and business meaning) for that item should be included in one of the XML schemas. The scope of the message model is the scope of the SOA solution being built which is reflected in the service model after the specification phase is complete.

- **Constraining**: The XML schemas will form part of the formal WSDL contract published for the services exposed in the solution. As such, both the creator and the recipient of an XML message should have the ability to verify that the message instance is compliant with the published contract. When designing the XML schemas you should constrain the values for all the elements and attributes to the set of values that the solution is designed to handle. A valid message should imply valid data within the limits of what can be specified by the XML schema language. It is important to understand that while we are restricting message content to that defined in the XML schema, we are not necessarily restricting the extensibility of the solution (see below)

- **Non-redundant**: The XML schemas will define types and elements only once and reuse other XML schema files rather than duplicating types and elements locally.

- **Reusable**: The XML schemas should be specified in such a way that types and elements can be leveraged by other XML schemas (especially the WSDL service descriptions). Every type defined in the XML schema that is the content type of an attribute or an element should be defined globally (i.e., at the top level in the Schema). Types that are defined globally can be reused in other XML schemas.

- **Extensible**: The schemas should be designed to be extensible – that is, new elements and attributes can be inserted. Extension points can be made explicit if the schema is being designed to be enhanced. Mechanisms that can be used to enable extensibility include: attribute and element wild cards, substitution groups, and type substitution. Note that this requirement seems incompatible with the requirement for constraining (above), and they do often produce some tension in the design process, but both are important features of a complete schema.

- **Non-modifying**: The XML schemas should not specify default values for attributes and element content. This is because default values may cause XML schema validation to update the message contents in cases where no value is defined for an element for which a default value has been defined.

The canonical message model provides two levels of reusability for service definitions:

1. The set of base types representing the business entities and attributes corresponding to the logical data model. These are imported into message definitions using `xsd:import` or `xsd:include`
2. The set of reusable message definitions. Each message provides a structured view of some subset of business entities. These message definitions are imported into the service WSDL files using `wsdl:import` as shown in Figure 21 .

Figure 21: WSDL Definition by Schema Composition

Overall this approach improves component reusability and ensures more consistent data structures and semantics across a portfolio of services.

The canonical message model can be constructed using the following steps:

1. Identify simple and complex types from canonical data model
    a. Naming standards
    b. Defining namespaces
    c. Schema design style
    d. Using attributes vs. elements
    e. Choosing data types
    f. Adding metadata and annotations
    g. Adding constraints
2. Identify candidate message formats
    a. From service model
    b. From existing assets
    c. From industry standards
    d. From variation oriented analysis
3. Design canonical message set
    a. Extension vs. restriction for complex types
    b. Type Hierarchies vs. composition
    c. Managing relationships

Step 1: Identify Simple and Complex Types from the Canonical Data Model

The starting point for the canonical message model is the definition of the individual data types and then the complex data types which will make up the building blocks for the messages that will eventually be defined. Both these constructs can be derived directly from the logical data model.

Attributes will map to either XML elements or XML attributes. Entities will map to XML elements. Rules such as value constraints, semantic metadata, and cardinality will also be propagated into the XML schema.

The XML schema language cannot fully replicate the structure of the data model concerning the type hierarchies and relationships that may exist. Each XML message definition is restricted to a tree structure. At the same time, it is meaningless to build a single XML message that traverses all the relationships and subtypes of the logical data model. Such a message would be impossible to build and have no practical use in the SOA. Instead, the following sections outline the basic approaches that can be used to capture the XML types and then form the canonical message definitions from them.

Step 1a: Naming Standards

A clear and consistent naming convention should be followed for all XML messages. The naming standard covers XML types, XML elements, XML attributes, and message names.

The naming standard may already be determined from existing rules that exist in the organization or the industry, but if no standards already exist then the following rules can be helpful. Clarity and consistency are the most important properties of the naming conventions chosen.

- Names should be mixed case with no spaces e.g. AccountNumber, ProductID
- Element names and attribute names can be taken directly from the corresponding definitions in the logical data model.
- Type names should be indicated with an appropriate suffix to make it clear that they are not elements or attributes e.g. SocialSecurityNumberType defines a data type with specific characteristics and rules which apply to the values of the XML element called SocialSecurityNumber.
- MessageNames should be indicated with a descriptive name for their content and appropriate suffix (such as *Request, Response, Error)* e.g. CustomerCreditHistoryRequest, OrderShippingStatusResponse, UnknownProductIDError.
- Readability is more important than tag length but it is acceptable to introduce abbreviations into names where there is no loss of meaning e.g. POSDepartmentID is preferable to both ID_DPT_POS and PointOfSaleDepartmentIdentifier.
- If the logical model used qualified names for attributes then the qualifiers can be dropped in the XML schema because the XML structure hierarchy already establishes the relationship of the attribute to the entity e.g. for an attribute ADDRESS.CITY in the logical model the XML element names will be

```
<ADDRESS>
    <CITY>
    </CITY>
</ADDRESS>
```

- Enumeration values should use names only (not numbers) in the primary language for the system e.g. english. This adds clarity to the message and also simplifies translation between languages where required.
- Repetition in names should be avoided when the XML structure already makes clear the repeated part of the name e.g. inside a `<CUSTOMER>` element it is better to tag an included element as `<NAME>` rather than the repetitive `<CUSTOMERNAME>` .

When defining the base business elements, types and attributes in XML schemas, you must make a choice about how to handle namespaces and qualifying names to avoid naming conflicts. This is discussed in the following section.

Step 1b: Defining Namespaces

XML namespaces are used in WSDL files to scope names and resolve ambiguity and conflict issues. There are a number of approaches that will work when building a WSDL file from composite schemas. It is important to define an approach for the project and follow this in all schemas to ensure consistency and readability.

You should aim to create 3 distinct sets of namespaces

1. For fundamental information types define one namespace. If the project is large and the definitions will be maintained independently across separate information domains then each domain should be given its own schema and namespace used to contain those elements for which it is responsible.

2. For reusable messages define another namespace. If the project is large and the message definitions will be maintained independently across separate business domains then each domain should be given its own schema and namespace used to contain those message definitions for which it is responsible.
3. Each service definition will be in a separate WSDL file using the WSDL namespace and importing the required business defined namespaces.

The following example illustrates this technique in simplified form:

1. Define base data type definitions

JKTypes.xsd

```
<schema xmlns=http://www.w3.org/2001/XMLSchema
   targetNamespace="http://www.JK.com/JKTypes"
   xmlns:jk="http://www.JK.com/JKTypes">

   <complexType name="CustomerType">
      <sequence>
         <element name="Number" type="string"></element>
      </sequence>
      <attribute name="Type" type="string"></attribute>
      <attribute name="TypeDescription" type="string"></attribute>
   </complexType>

   <complexType name="CorporateCustomer">
      <complexContent> <extension base="jk:CustomerType">
      <sequence>
         <element name="IncorporatedName" type="string"></element>
         <element name="TradingName" type="string"></element>
         <element name="TaxID" type="string"></element>
      </sequence></extension></complexContent>
   </complexType>

   <complexType name="IndividualCustomer">
      <complexContent> <extension base="jk:CustomerType">
      <sequence>
         <element name="FamilyName" type="string"></element>
         <element name="SSN" type="string"></element>
         <sequence>
            <element name="GivenName" type="string"></element>
         </sequence>
      </sequence> </extension> </complexContent>
   </complexType>

   <complexType name="AccountType">
      <sequence>
         <element name="Number" type="string"></element>
         <element name="CurrentBalance" type="string"></element>
      </sequence>
      <attribute name="Type" type="string"></attribute>
      <attribute name="TypeDescription" type="string"></attribute>
   </complexType>

</schema>
```

2. Define a unique target namespace for each message and use `xsd:import` to pull any referenced type definitions into the message definition. e.g.

JKMessages.xsd

```
<schema xmlns=http://www.w3.org/2001/XMLSchema
   targetNamespace=http://www.JK.com/JKMessages
   xmlns:jkm=http://www.JK.com/JKMessages
   xmlns:jk="http://www.JK.com/JKTypes">

   <import namespace=http://www.JK.com/JKTypes
           schemaLocation="JKTypes.xsd">
   </import>

   <complexType name="CustomerRequestMessage">
      <sequence>
         <element name="Customer" type="jk:CustomerType"></element>
      </sequence>
   </complexType>

   <complexType name="CustomerAccountListMessage">
      <sequence>
         <element name="Customer" type="jk:CustomerType"></element>
         <sequence>
            <element name="Account" type="jk:AccountType"></element>
         </sequence>
      </sequence>
   </complexType>
</schema>
```

3. Bring the messages into the WSDL file using `wsdl:import` to maintain the message namespace in the WSDL file.

JKService.xsd

```
<wsdl:definitions xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
                  xmlns:jks="http://www.JK.com/JKService/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  name="JKService"
                  targetNamespace=http://www.JK.com/JKService/
                  xmlns:jkm="http://www.JK.com/JKMessages">

   <wsdl:import namespace="http://www.JK.com/JKMessages"
                location="JKMessages.xsd">
   </wsdl:import>

   <wsdl:types>
   <xsd:schema targetNamespace="http://www.JK.com/JKService/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="RetrieveAccountsForCustomer">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="RetrieveAccountsForCustomerResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="out" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
```

```
  <wsdl:message name="RetrieveAccountsForCustomerRequest">
       <wsdl:part name="jks:MaxNbrAccounts" type="int"/>
    <wsdl:part name="jks:Customer"
               type="jkm:CustomerRequestMessage"/>
  </wsdl:message>
  <wsdl:message name="RetrieveAccountsForCustomerResponse">
    <wsdl:part name="jks:MoreAccountsExist" type="boolean"/>
    <wsdl:part name="jks:CustomerAccountList"
               type="jkm:CustomerAccountListMessage"/>
  </wsdl:message>

  <wsdl:portType name="JKService">
    <wsdl:operation name="RetrieveAccountsForCustomer">
      <wsdl:input message="jks:RetrieveAccountsForCustomerRequest"/>
      <wsdl:output message="jks:RetrieveAccountsForCustomerResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="JKServiceSOAP" type="jks:JKService">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="RetrieveAccountsForCustomer">
      <soap:operation
                  soapAction="http://www.JK.com/JKService/NewOperation"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="JKService">
    <wsdl:port binding="jks:JKServiceSOAP" name="JKServiceSOAP">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Step 1c: Schema Design Style

The intention of building a canonical message model is to bring reuse and consistency to the definition of data elements in the WSDL service definitions. There are three main choices as to how the base data types, elements and attributes can be structured in the canonical message model:

1. The ***Russian Doll*** style defines root elements globally. Elements that cannot be a document's root are defined as the need arises as are attributes and types. These definitions are then nested in the elements that use them, i.e. they are local to the scope of the root element under which they were defined. This means that the smallest reusable part of the schema is the root element. This approach is useful in SOA for defining entities or hierarchies of entities from the logical data model as XML elements that have no variation in structure from one message to another. It does not support shared type definitions across entities that do not fall in the same element structure.

```
<schema targetNamespace="http://ibm.com/data" elementFormDefault="qualified">
<!-- Russian Doll -->
<element name="Response">
       <complexType>
       <sequence>
      <element name="Collection">
```

```
            <complexType>
            <sequence>
               <element name="Element" minOccurs="0" maxOccurs="unbounded">
                  <complexType>
                  <sequence>
                 <element name="ID" type="int"/>
                 <element name="Name" type="string" minOccurs="0"/>
              </sequence>
              </complexType>
           </element>
        </sequence>
        </complexType>
     </element>
  </sequence>
  </complexType>
</element>
</schema>
```

2. The ***Salami Slice*** style takes the opposite approach and defines all elements globally. This allows reuse at a very fine grained level (i.e. the attribute definitions from the logical model) but loses all the entity-attribute and entity-entity structure which must be recreated in the message definition schemas. Consequently this approach brings very little reusability and is not recommended for SOA.

```
<schema targetNamespace="http://ibm.com/data">
<!-- Salami Slice -->
<element name="ID" type="int"/>
<element name="Name" type="string"/>
<element name="Element">
   <complexType>
   <sequence>
      <element ref="data:ID"/>
      <element ref="data:Name" minOccurs="0"/>
   </sequence>
   </complexType>
</element>
<element name="Collection">
       <complexType>
       <sequence>
      <element ref="data:Element" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   </complexType>
</element>
<element name="Response">
       <complexType name="tResponse">
       <sequence>
      <element ref="data:Collection"/>
   </sequence>
   </complexType>
</element>
</schema>
```

3. The ***Venetian Blind*** style defines all types globally, but exposes only elements that can be the root element for a message. Like the Russian Doll approach, entities or hierarchies of entities from the logical data model can be exposed as reusable XML elements for messages. The advantage of the Venetian Blind model is that XML type definitions can be shared across different XML elements.

```
<schema targetNamespace="http://ibm.com/data" elementFormDefault="qualified">
<!-- Venetian Blind -->
<complexType name="tElement">
<sequence>
   <element name="ID" type="int"/>
```

```
   <element name="Name" type="string" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="tCollection">
<sequence>
   <element name="Element" type="data:tElement" minOccurs="0"
maxOccurs="unbounded"/>
</sequence>
</complexType>
<complexType name="tResponse">
<sequence>
   <element name="Collection" type="data:tCollection"/>
</sequence>
</complexType>
<element name="Response" type="data:tResponse"/>
</schema>
```

Which is the best choice? The answer will depend on the complexity of the logical data model and the number of different message views that overlap a given entity or group of entities. Salami slice brings very little reuse and unnecessary flexibility for most solutions and can normally be eliminated. Then depending:

- If the logical data model uses shared type definitions for different attributes in unrelated entities then these should be globally defined XML types in the canonical message model i.e. Venetian Blind style. If no global type definitions exist in the logical data model then Russian Doll style can be used.
- Entities from the logical model can then be broken down into subsets which are exposed as root elements in the XML schema structure. How granular these subsets become is dependent on the variations that will eventually exist in the WSDL messages. If an entity is always referenced within the context of its parent (e.g. if OrderDetail entities are always contained in the parent Order entitiy) then that part of the logical data model can be exposed in a single element definition in the XML schema. For entities that appear in different relationships for different messages it will be necessary to define the entity as an XML root element and then let the message definition define the relationship.
- XML elements can be defined as either optional or compulsory using the `minoccurs` attribute. If different messages need to include only a subset of attributes from an entity in the logical data model then this is achievable provided `minoccurs=0` is deifined in the XML schema.

Once the different schema styles are well understood it is perfectly acceptable to use a combination of styles provided that a consistent set of rules governs which data characteristics are defined by which of the styles.

Step 1d: Using Attributes vs Elements

Each entity in the logical data model is transformed into an element in the XML schema. Attributes of the logical data model entities however, can be represented in the XML schema as either nested elements of the parent element or as attributes of the parent element.

Consider the following 3 representations of the same information

1. Using all elements

```
<PARTY>
   <TYPE> Employee </TYPE>
   <PERSON>
      <FIRSTNAME> Tana </FIRSTNAME>
      <LASTNAME> Umaga </LASTNAME>
      <SEX> M </SEX>
   </PERSON>
</PARTY>
```

2. Using all attributes

```
<PARTY Type="Employee">
   <PERSON FirstName="Tana" LastName="Umaga" Sex="M" />
</PARTY>
```

3. Using a mix of elements and attributes

```
<PARTY Type="Employee">
   <PERSON>
      <FIRSTNAME> Tana </FIRSTNAME>
      <LASTNAME> Umaga </LASTNAME>
      <SEX> M </SEX>
   </PERSON>
</PARTY>
```

Any of these three forms is acceptable but it is essential to define a standard and apply it consistently across all definitions. The objective is making the resulting document easy to read and understand. Some simple guidelines are:

- Use attributes for metadata about the parent element
- If the value is semantically tied to the parent element then express it as an XML attribute of the parent element
- If the value is a business data value which has meaning in its own right then use an element
- Use attributes when the value will be frequently present in order to improve the human readable form of an XML instance document or reduce its size.
- If in doubt make the value an element (which is more extensible)

In addition, attributes can be very useful for describing relationships from the logical data model, particularly for cases where multiple relationships exist between the same entities. Consider the one to many relationship of PERSON to ADDRESS from a logical data model. The XML schema can define “Role” as an attribute, taken from the relationship, and displayed on the resulting elements

```
<PERSON>
   <NAME> Fred </NAME>
   <ADDRESS Role=”Residential”>
      <STREET> Rocky Road </STREET>
      <CITY> Bedrock </CITY>
   </ADDRESS>
   <ADDRESS Role=”Billing”>
      <STREET> Pebble Park </STREET>
      <CITY> Bedrock </CITY>
   </ADDRESS>
</PERSON>
```

Step 1e: Choosing Data Types

All elements and attributes defined in the canonical message model must be given a fundamental type. These types can be the built in types of the XML schema language or user defined types which can infinitely extend the schema language.

`Simple-Type` and `Complex-Type` are used to specify user defined types.

All entities from the logical data model should be defined as `complex-type` in the canonical message model. Entity names from the logical data model can be directly reused for the `complex-type` name in the message model where they comply with good XML naming standards. It is good practice to add the suffix “`Type`” to the name to indicate that it is a type definition and not an element name. This prevents confusion

when later defining elements based on the complex type e.g. a logical data model entity called `Order` is defined as complex type `OrderType` in the XML schema and then used where appropriate to define XML elements with name `Order`.

Attributes from the logical data model become elements or attributes in the canonical message model. Each element/attribute can be defined using one of the base XML data types (`string`, `Boolean`, `int`, `dateTime`, etc) or by first defining a `Simple-Type`. Choose to define a `Simple-Type` when any of the following conditions apply:

- Entity attributes of that type will be exposed as reusable elements in the schema hierarchy i.e. they will be included in other schemas without their parent complex types.
- there are explicit business rules constraining the format and/or the allowable values for the data type e.g. serial number is always 3 characters and 5 numbers separated by hyphen (DFK-74589)
- the type has constrained format or values and is reused in many complex types for different attributes e.g. temperature, telephone number, GPS coordinates.

`Complex-Types` should always be named and never left as `anonymous`.

If the logical data model makes use of sub-typing to provide extensibility then this can be reflected in the XML schemas representing those types in the message model. XML `extension` attribute can be used in the `complex-type` definition to implement the same inheritance structure in the XML schema as exists in the logical data model. This may or may not be the best choice and is discussed in detail later in this document.

Step 1f: Adding Metadata and Annotations

Remember that the target audience of the canonical message model is downstream developers in the SOA. This includes:

- XML developers building message models from the base data type schemas
- Service developers building WSDL files from the reusable message schemas
- Developers building the service implementation top-down from the WSDL definition
- Developers building business processes, composite services and other components that are consumers of the service interfaces.

Depending on your project's tooling and infrastructure the semantic metadata describing the reusable components developed in the canonical message model can vary. However, the lowest common denominator will always be the XML schemas themselves. For this reason it is always prudent to make heavy use of textual annotations to provide clarity and semantic understanding of the schemas being developed. This can be achieved using the `xs:documentation, xs:annotation, xs:appinfo` and `xs:lang` tags in your schemas.

Step 1g: Adding Constraints

XML schema supports a wide variety of constraints be placed on the possible document supported by the schema. While it is possible to validate XML documents at runtime, most systems will avoid extensive validation of XML because of the high performance overhead. Instead, schema validation is used much more extensively during the development and testing cycle. Beyond that, adding constraints to schemas should be principally used for documentation purposes and not run-time data validation.

Runtime data validation of message content is always the responsibility of the consumer of the message e.g. the service implementation. Any additional runtime validation performed during message composition or

delivery is typically redundant. An exception to this principle is in-flight message transformation where the structure and/or content of the message may be changed in the ESB for example. In these cases using XML schema validation to verify that newly constructed messages are well formed and contain valid data may be appropriate.

Building and documenting example messages used for testing and illustration purposes is a good practice.

Adding constraints to the XML schemas for the canonical message model requires that two opposing objectives are balanced.

1. Strongly constraining the schema provides clear guidelines and controls on allowed messages formats and content. In this way the number of errors caused by "unexpected" message content can be reduced.
2. Flexibility and extensibility are important characteristics needed in the message model. XML provides many possibilities for extending the schemas to support changing requirements and extension points can be designed into the schemas allowing additional message content to be added with no change to the schema. However, there is only value in building such flexibility into the schema if there is a proven business need and if the corresponding providers and consumers of the messages are capable of handling the allowed variations in message content. Use variation oriented analysis scenarios to identify when and where it is worthwhile to build in schema flexibility.

Typical constraints that can be set on the canonical message model include:

- use `minOccurs` and `maxOccurs` to define realistic limits on message sizes for the system and also to define which elements are optional or compulsory in the message. If in doubt set `minOccurs=0`.
- Use `minLength` and `maxLength` sparingly and only in accordance with business rules and not to expose limitations in the underlying technology (which may change).
- It is generally more meaningful to set `minOccurs=0` rather than `minLength=0` unless there is a significant business meaning when the data element holds a `null` value.
- use `pattern` and `enumeration` to constrain the allowed values for a data element when this provides clarity to the schema user. It is normally better to apply these rules as part of `simple-type` or `complex-type` rather than in element definitions
- for numeric and date value you can set `minValue` and `maxValue` to limit the allowable range but again these should be used sparingly and are better suited to `simple-type` and `complex-type` definitions than element definitions.
- Use `unique`, `key` and `keyref` where appropriate to show relationships in your data structures which are not self-evident in the tree structure of your schema.

The following are approaches that can be taken to add flexibility to the schema:

- If the logical data model makes use of inheritance to support reuse and extensibility then this can be implemented by defining a similar hierarchy of complex-type definitions in the XML schema using the `extension` attribute on the `complex-type` definition.
  Define the complex-type as `abstract` only if you are certain that no messages will be needed that pass this type. If you are uncertain that this will always be the case then do not use `abstract`.
- Use targeted extensibility when you have clearly defined extension points identified from variation oriented analysis. This can be implemented using xs:`any` in the element definitions and setting the attributes `namespace="##any" process="lax"`. This allows additional elements and values to be inserted at the end of the element and still pass schema validation. Remember that this is typically only useful for cases when the message consumer can handle additional generic content as name-value-pairs.

- `xs:anyattribute` can be used to allow unnamed attributes to be added to the parent element and still pass schema validation processing.
- Always aim to keep backward compatibility when updating schemas.
  Add version numbers to the target namespace for all schemas. When a schema is changed in a way that is not backward compatible then increment the version number to exclude the messages built according to the previous document versions. To manage simple extensions to the schema which maintain backward compatibility you can define a minor release level in the schema version attribute.

Step 2: Identify Candidate Message Formats

The canonical message model provides a reusable set of types and messages for the SOA solution. Defining the messages requires compromises. The resulting message set must be general enough to be reusable. Ideally each message will be used by more than one service. But each service must be able to construct messages from the message set which are appropriate for that services interface. The message designer must therefore consider the full range of potential message uses in the system along with the most likely areas of extensibility for the messages. The following four areas will normally provide the candidate message formats from which the message modeler can resolve a core set of reusable messages:

- the service model
- existing asset analysis
- industry standards
- variation oriented analysis and design

Step 2, Option a: From Service Model

The service model and the corresponding logical data model provide the major input to the canonical message model design. During the SOMA specification phase there will be significant iteration and interaction between the designers of these three work products.

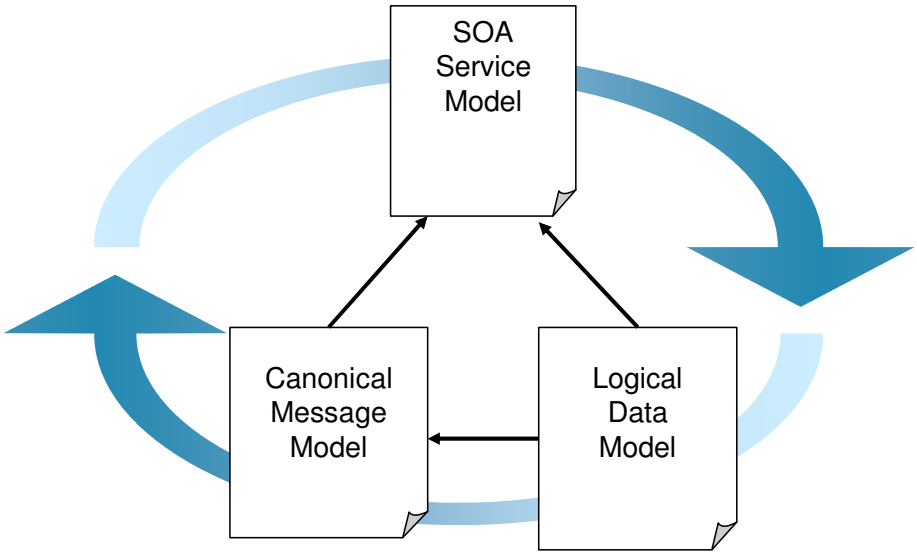

Figure 22: Canonical Message Model is aligned to SOA Service Model

During the early part of SOMA specification phase the service model includes a list of candidate services with partially developed interface definitions. Both the logical data model and the canonical message model are developed in parallel with the service model during the specification phase. Each provides inputs to the others as the three models are refined into a fully detailed set of aligned models.

The canonical message model initially uses the logical data model as a guide to data elements structures for the solution and uses the service model as a guide to the intended message usage. These requirements are balanced against the other requirements identified from existing assets, industry standards, and variation oriented analysis to build the canonical message model.

As the canonical message model matures, the service model is refactored so that all service interfaces are expressed using messages from the message model.

The canonical message model may also identify data structure issues that drive changes back into the logical data model.

Step 2, Option b: From Existing Assets

It is very likely that messaging solutions already exist in the organization before the SOA project is started. The messages already in use for these solutions should be reviewed in the design of the canonical message model. In some cases the existing messages may be directly reusable in the canonical message model while in others they may simply provide a good cross-reference for completeness and semantic meaning.

Where existing messaging systems will be integrated into the SOA solution the existing message schemas will eventually be either migrated or mapped to the canonical message model.

Step 2, Option c: From Industry Standards

Industry standard message models provide a valuable source of information when developing the canonical message model.

If the SOA solution has a goal of external integration then compliance with appropriate message interchange formats (e.g. EDIFACT, ACORD, HL7, TLOG, etc) will almost certainly dictate a large part of the canonical message model. In other cases the industry standard message formats may provide useful guidelines for structuring a proprietary canonical message format in such a way that the messages are reusable and extensible.

Step 2, Option d: From Variation Oriented Analysis and Design

The canonical message model should be designed to be flexible and extensible but without losing the benefits of strong typing and control of semantic meaning. XML provides many facilities for adding flexibility and extensibility into the schemas but simply adding `xs:any` to every element in the schemas would significantly undermine the effectiveness of the message model.

In SOMA, the variation-oriented analysis (VOA) isolates functional and structural aspects of the domain that have variations or are likely to change from those aspects that represent commonalities or likely to remain unchanged. Use the outputs from this activity to identify those areas of the canonical message model where flexibility is really required verses those areas where more rigid structures, types and constraints are appropriate.

Step 3: Design Canonical Message Set

The following are some of the common design issues encountered when finalizing the message design for the canonical message model:

- Choosing between extension and restriction when implementing complex types
- Choosing between using type hierarchies and composition
- Dealing with entity relationships, especially many-to-many and recursive relationships.

Step 3, Decision a: Extension vs Restriction for Complex Types

In XML schema, as well as deriving new complex types by extending existing content type models, it is possible to derive new types by restricting the content models of existing types. Restriction of complex types is conceptually the same as restriction of simple types, except that the restriction of complex types involves a type's declarations rather than the acceptable range of a simple type's values. A complex type derived by restriction is very similar to its base type, except that its declarations are more limited than the corresponding declarations in the base type. In fact, the values represented by the new type are a subset of the values represented by the base type (as is the case with restriction of simple types). In other words, an application prepared for the values of the base type would not be surprised by the values of the restricted type.

For an SOA canonical message model it is very rarely a good idea to use complex type restriction. The reason is that this pattern in a model is not a natural fit for almost any service implementation choice (e.g. it does not exist in data modeling, OO class modeling, etc.)

Restriction causes the redefinition of the complex types elements and attributes, which leads to fragility in your XML schema; any change to a super type will require corresponding changes in all restricting subtypes, even those in other schemas.

Step 3, Decision b: Type Hierarchies vs Composition

Design by sub-classing has an important role to play in XML schema design. It is likely that the logical data model will include examples of sub-typing and it is easy to assume that the best XML implementation of this type of data structure is to mirror the type hierarchy into the XML schema. This is frequently not the case. Long, extended type hierarchies can lead to brittle, non-modifiable designs that are virtually impossible to understand.

A good rule of thumb is that type hierarchies in the XML schema should rarely exceed 2 levels i.e. one level of sub-typing.

Building large complex types through composition of smaller complex types using `xs:sequence`, `xs:choice and xs:all` is typically a more flexible approach and leads to more easily understood schemas.

Step 3, Decision c: Implementing Relationships

In the logical data model there will be many relationships connecting the entities. An XML schema is more limited in how it can represent these relationships into a tree structure. The message modeler must therefore think carefully about which relationships will be exposed in which message. This is especially true in the case of relationships which could lead to deep levels of nesting in the XML documents produced i.e. many-to-many relationships and recursive relationships.

Many-to-many relationships are common in entity relationship diagrams and pose some challenges when attempting to represent them in XML documents. The most complex is the recursive many-to-many relationship. For example, a person is associated with multiple addresses (e.g. home, postal, office, …) and any given address can be shared by multiple people (e.g. many people are employees at a specific office address). This would typically be represented in a physical data model as follows in Figure 23

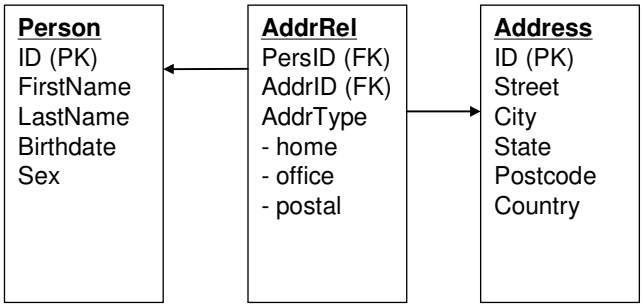

Figure 23: Many-to-many Relationship

There are several options for how these relationships can be expressed in XML schema. Before choosing one, you should first understand all the possible ways this relationship will be exposed through the services sharing the canonical message model.

Approach 1: Discard some information

In the above example the known use cases might be:

- retrieve address of specified type for specified person
- retrieve all addresses for a given person
- retrieve all people for a given address for a specified address type
- update address of specified type for specified person

For the services listed above there is no requirement to resolve beyond one level of relationship and the resulting messages can be simple tree structures in XML. To the user of the schema, the many-to-many relationship has been collapsed into a normal one-to-many relationship.

```
<schema>

<complexType name="PersonType">
              <sequence>
                     <element name="FirstName" type="string"></element>
                     <element name="LastName" type="string"></element>
                     <element name="Sex" type="string"></element>
                     <element name="BirthDate" type="string"></element>
              </sequence>
       </complexType>

       <complexType name="AddressType">
              <sequence>
                     <element name="Street" type="string"></element>
                     <element name="City" type="string"></element>
                     <element name="PostCode" type="string"></element>
                     <element name="Country" type="string"></element>
              </sequence>
              <attribute name="Type" type="string"></attribute>
```

```
        </complexType>

        <complexType name="AdressesByPersonMessage">
                <sequence>
                        <element name="Person" type="tns:PersonType"></element>
                        <sequence>
                                <element name="Address" type="tns:AddressType"></element>
                        </sequence>
                </sequence>
        </complexType>

        <complexType name="PeopleAtaddressType">
                <sequence>
                        <element name="Address" type="tns:AddressType"></element>
                        <sequence>
                                <element name="Person" type="tns:PersonType"></element>
                        </sequence>
                </sequence>
        </complexType>
</schema>
```

And so the resulting messages would follow the pattern shown below.

GetPersonsAddresses( )

```
<AdressByPerson>
   <Person>
      <Address Type= > </Address>
      <Address Type => </Address>
   </Person>
</AdressByPerson>
```

GetPeopleAtAddress( )

```
<PeopleAtAddress>
   <Address Type= >
      <Person> </Person>
      <Person> </Person>
   </Address>
</PeopleAtAddress>
```

Approach 2: Use xs:ID and xs:IDREFS to expose the relationship in the documents

But what if we add the following service requirement (e.g. for a police investigation application):

> find all the people who can be associated with the specified person through less than 3 address relationships e.g. Pebbles Flintstone lives at 10 Rocky Road which is the home of Fred Flintstone who works at Bedrock Quarry which is also the workplace of Barney Rubble who lives at 12 Rocky Road which is also the home of Bam Bam Rubble, therefore Pebbles Flintstone may have an association with Bam Bam Rubble.

For this case it is possible to extend the message tree structure to include the three levels of nesting and simply expand the resolved relationship instances into the message. However, there are two significant problems with this approach:

1. The nesting level is hard coded into the schema and is therefore inflexible (what if the user decides he wants to allow search of 4 levels?)
2. The resulting message documents can quickly become huge because they contain many redundant copies of the same information (e.g. if Fred Flintstone is associated to 20 people then his name/address elements will appear 20 times in the message under the Pebbles Flintstone parent)

An alternative approach is to expose the relationship itself as an element in the XML schema and leave the consumer of the message to resolve the relationships from the data set as they see fit. You can use `xs:idref` to achieve this in the schema as follows:

```
<schema>
   <complexType name="PersonType">
      <sequence>
         <element name="PersonID" type="ID"></element>
         <element name="FirstName" type="string"></element>
         <element name="LastNAme" type="string"></element>
         <element name="Sex" type="string"></element>
         <element name="BirthDate" type="string"></element>
         <element name="AddressIDREFS" type="IDREFS"></element>
      </sequence>
   </complexType>

   <complexType name="AddressType">
      <sequence>
         <element name="AddressID" type="ID"></element>
         <element name="Street" type="string"></element>
         <element name="City" type="string"></element>
         <element name="PostCode" type="string"></element>
         <element name="Country" type="string"></element>
         <element name="PersonIDREFS" type="IDREFS"></element>
      </sequence>
   </complexType>

   <complexType name="PeoplesAddressesMessage">
      <sequence>
         <element name="Person" type="tns:PersonType" minOccurs="0"
                        maxOccurs="unbounded"> </element>
         <element name="Address" type="tns:AddressType" minOccurs="0"
                        maxOccurs="unbounded"> </element>
      </sequence>
   </complexType>

</schema>
```

The output when using this approach is shown below.

```
<message>
  <Person>
    <PersonID>p1</PersonID>
    <FirstName>Fred</FirstName>
    <LastName>Flintstone</LastName>
    <Sex>Male</Sex>
    <BirthDate>12 Nov</BirthDate>
    <AddressIDREFS>a1 a2</AddressIDREFS>
  </Person>
  <Address>
    <AddressID>a1</AddressID>
    <Street>10 Rocky Road</Street>
    <City>Bedrock</City>
    <PostCode>123</PostCode>
    <Country>USA</Country>
    <PersonIDREFS>p1</PersonIDREFS>
  </Address>
  <Address>
    <AddressID>a2</AddressID>
    <Street>Bedrock Quarry</Street>
    <City>Bedrock</City>
```



```
    <PostCode>123</PostCode>
    <Country>USA</Country>
    <PersonIDREFS>p1</PersonIDREFS>
  </Address>
</message>
```

This approach can be useful when there is no qualification associated with the relationship. In the example you will have noticed that we can no longer associate the address type (home, office, postal) with the address because there is no suitable place in the structure to hold it.

Approach 3: Expose the relationship as an element

If the relationship between 2 entities holds a variety of semantic meanings depending on attributes of the relationship then the best option may be to expose the relationship explicitly as shown in the following schema.

```
<schema>
   <complexType name="PersonType">
      <sequence>
         <element name="PersonID" type="ID"></element>
         <element name="FirstName" type="string"></element>
         <element name="LastName" type="string"></element>
         <element name="Sex" type="string"></element>
         <element name="BirthDate" type="string"></element>
      </sequence>
   </complexType>

   <complexType name="AddressType">
      <sequence>
         <element name="AddressID" type="ID"></element>
         <element name="Street" type="string"></element>
         <element name="City" type="string"></element>
         <element name="PostCode" type="string"></element>
         <element name="Country" type="string"></element>
      </sequence>
   </complexType>

   <complexType name="PersonAddressRelationshipType">
      <sequence>
         <element name="PersonIDREF" type="IDREF"></element>
         <element name="AddressIDREF" type="IDREF"></element>
      </sequence>
      <attribute name="Type" type="string"></attribute>
   </complexType>

   <complexType name="PeoplesAddressesMessage">
      <sequence>
         <element name="Person" type="tns:PersonType" minOccurs="0"
                           maxOccurs="unbounded"> </element>
         <element name="Address" type="tns:AddressType" minOccurs="0"
                           maxOccurs="unbounded"> </element>
         <element name="Relationship"
                           type="tns:PersonAddressRelationshipType" minOccurs="0"
                           maxOccurs="unbounded"> </element>
      </sequence>
   </complexType>

</schema>
```

And the resulting message is as follows.

```
<message>
   <Person>
      <PersonID>p1</PersonID>
      <FirstName>Fred</FirstName>
      <LastName>Flintstone</LastName>
      <Sex>Male</Sex>
      <BirthDate>12 Nov</BirthDate>
   </Person>
   <Address>
      <AddressID>a1</AddressID>
      <Street>10 Rocky Road</Street>
      <City>Bedrock</City>
      <PostCode>123</PostCode>
      <Country>USA</Country>
   </Address>
   <Relationship Type="Home">
      <PersonIDREF>a1</PersonIDREF>
      <AddressIDREF>p1</AddressIDREF>
   </Relationship>
      <Relationship Type="Postal">
      <PersonIDREF>a1</PersonIDREF>
      <AddressIDREF>p1</AddressIDREF>
   </Relationship>
</message>
```

While this approach offers the most flexibility, it places a much higher burden on the consumer of the message to understand the technical data structure and process the relationships.

5.5 Begin Data Mapping Across SOA Layers

5.5.1 Definition

Data passes through the layers of the SOA reference architecture both horizontally and vertically.

- During execution of a business process each task receives input data from the previous one, executes the actions for which it is responsible, and then passes outputs to the next task. Data is shared horizontally through the process. A similar scenario exists in user interface aggregation, service composition, etc.
- During the execution of a specific task in a business process, data is passed vertically through the service implementation layers e.g. from the business process thru the service interface, onto the ESB, into the service implementation provider which may call an application API which accesses a database.

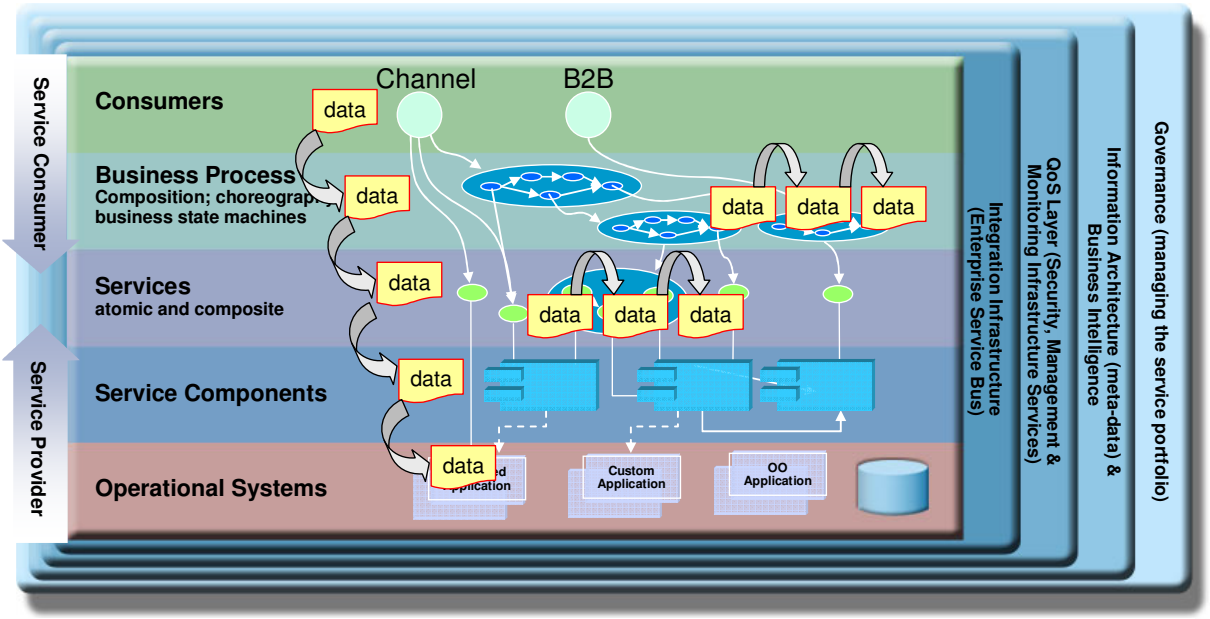

Figure 24: Data Flows in SOA Solution Stack

Whenever data is shared, either horizontally or vertically, there must be a common understanding between the two participants of both the structure and the meaning of the data being exchanged. If a common data representation is agreed between the participants, such as passing XML messages as defined in the canonical message model, then this task is trivial. However, there are many cases where conformity to the canonical message model is not possible e.g. legacy applications, legacy databases, packaged applications, and external service providers will all have developed components without knowledge of the canonical message model.

Data mapping provides the rules for exchanging data when no common format is already in place. These rules include:

- names of data elements and structures
- datatypes for each data element e.g.
  string including language, character set and code page
  decimal including length, precision and representation,
  datetime including format and precision
- possible values for data elements including known validation rules and any special values e.g. country code lists, “N/A”, “*NONE*, null, etc.
- description of the content of the data element in understandable business language.

If two components in the solution must share data then there are two possible approaches for defining the mapping:

1. Map each data format to the canonical message model. During execution each participant need only understand one external data format i.e. the canonical message model.
2. For each mapping requirement, map directly between the native data formats of each of the participants.

5.5.2 Objective

Performing detailed data mapping across SOA layers has the following objectives:

- to define the data sharing rules that will be implemented between those components in the solution which do not conform directly to the logical data model and canonical message model.
- to identify inconsistencies in data structure or data meaning across components that could prevent successful realization of the solution.
- to validate that the business process models, service model, logical data model and canonical message model have been designed appropriately to support all components in the solution.
- to provide design specifications for implementing run-time data mapping and transformation components in the solution.

5.5.3 Value

The value of defining this work product is to identify data sharing problems early in the design stages of the solution and provide specifications for the implementation of the run-time data mapping components.

Failure to complete this activity may lead to increased development times for the solution or even runtime errors if data mismatch errors are not identified during design.

This activity may not be required if the solution is entirely new and there is no data sharing with existing or external components.

5.5.4 Approach

Data mapping is defined during SOMA specification to capture the rules for sharing data between different participating components in the solution. The choice of work product for recording the mappings can vary based on the project technology choices but a simple and common approach is to record the data mappings in spreadsheet form.

The following illustration is an example of a simple spreadsheet mapping definition:

Source	Data Type	Length	Rules	Target	Data Type	Length
getCustomerDetailsResponse				**CRM:getPerson**		
FirstName	string	25	None	FNAME	varchar	20
Initial	string	1	First initial only	MNAME	varchar	20
LastName	string	25	None	LNAME	varchar	20
DateOfBirth	date		Convert YYYYMMDD to iso date	BIRTHDAY	varchar	8
SocialSecurityNumber	integer		If non numeric value set = 0	SSN	varchar	10
Sex	string	1	None	SEX	char	1
Height	decimal	5.2	Not available so default to null			
			Not used	PERSON#	long	

Figure 25: Using Spreadsheets to Capture Mapping Rules

Clearly such an example is the minimal approach but often this will suffice. If the project is using products that support automated generation of runtime transformations from design artifacts then it can be efficient to begin high level mapping directly in the implementation tools with a view to completing, testing and deploying these later in the project. Figure 26 is an example of such a tool used for mapping XML structures and generating XLS transformations from the graphical mapping definition:

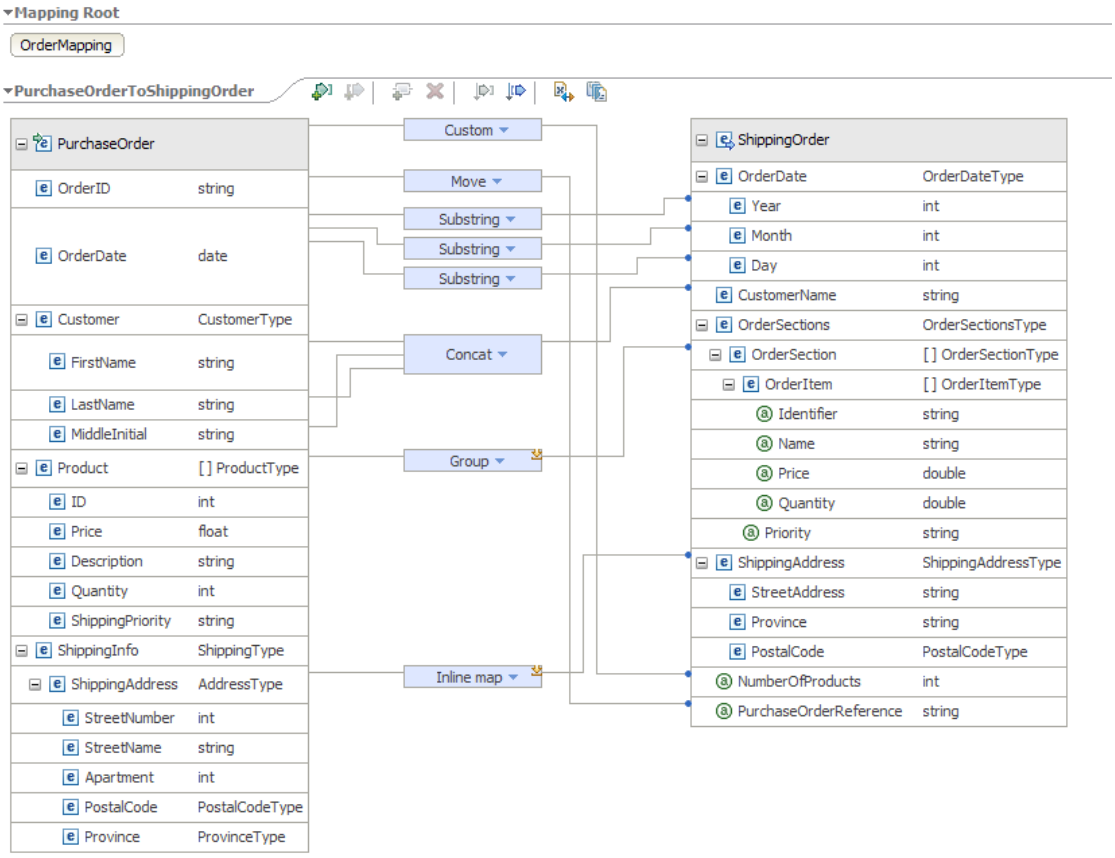

Figure 26: Using Mapping Tools to Capture Mapping Rules

There are many data interchange points to consider when defining data mappings for the solution. They will vary depending on the scope of each project but at the very least the following list should be checked:

- horizontal data flows through business process should be verified that each data exchange is complete and correct for both structure and meaning.
- for new services to be developed map the service interface messages to the implementation class model and underlying database model and verify that the data being exposed is correct and complete in both structure and meaning.
- for services implemented through existing application API's, map the service message interface to the application API interface(s) and verify that the data exposed through the service interface is complete and correct in both structure and meaning
- for information services map the service interface messages to the underlying data systems and verify that the data exposed through the service interface is complete and correct in both structure and meaning
- for external services and processes map the service interface messages to the local data/message model and verify that the data shared is complete and correct in both structure and meaning.

6. SOMA Realization Phase

Realization related decisions are made and refined throughout the SOMA process. However, it is during the realization phase that the design created so far is formally mapped to the technologies that will realize them. At this stage, architectural and design decisions regarding how components will be implemented are made: the technologies, the standards, the legacy system functionality that may be leveraged, the adapters required to wrap the legacy systems. Some of these decisions may require further validation of their technical feasibility.

The SOMA realization phase deals with the architectural and realization decisions that have to be made to implement the service functionality, taking into consideration quality of service as well as business and IT constraints. These decisions may be started earlier in the SOMA process but are completed during realization through the following activities:

- Service allocation - deals with the allocation of services into their appropriate component containers.
- Component allocation to layers – identifies appropriate architectural layers where components are to be placed and the rules for how the layers interact with each other.
- Technical feasibility exploration – assessment of feasibility of technical constraints and technology identified to realize services.

In previous sections of this paper, we have described a set of tasks from the information discipline in SOMA that do not depend on the realization of a service. In this section, we focus on a specific type of service that we call information services where we apply various patterns and capabilities from information architecture and management technologies. We will first briefly recap the major activities in SOMA realization and then introduce new tasks within those activities.

Service Allocation

Service allocation associates services with service components that will realize them. Service allocation is an ongoing and iterative process that can start as soon as services and service components are first identified (i.e. during SOMA identification). During service realization decisions, it is essential to verify that services have been allocated to service components. Validation of service allocation decisions can lead to some re-factoring in cases where a service that was initially assigned to one component needs to be re-assigned to another as a result of evaluation of refined requirements.

There is only one new task in this activity from the information discipline:

- **Validate the Identified Information Services (see Sect. 6.2)**

  The tasks in the information discipline of SOMA 3 have so far been independent on how we actually realize the services. At this point in SOMA realization we have to make a final decision which services are information services and require specific considerations from an information perspective. The initial categorization of services in the SOMA identification phase (see Sect. 4.5) needs to be revisited and ultimately decided in this activity. This decision will be closely related to the design in the realization (see below).

Component Allocation to Layers

Component allocation to layers (also known as component layering) involves identifying suitable architectural layers within which components can be placed. Components are then placed into these layers according to their functionality and ability to fulfill non-functional requirements. Components (service, functional and technical) are assigned to appropriate layers in the SOA reference architecture. For example:

- All components that provide application-independent business functionality are typically allocated to the business process layer. Application-independent business functions are things like “acquire new customer” and “introduce new product” that span multiple applications.
- All components that provide technical functions, such as error handling, authentication, logging and audit are allocated to the corresponding technical service layers. These components are both business and application independent. In some cases, proximity of technical to functional components may require they be placed in a common layer.
- These are architectural decisions and need to be documented as such.

The following activity needs to be considered in the information discipline during the component allocation to layers:

- **Select Information Service Patterns (see Sect. 6.3)**
  For the information services that we have identified, we then need to determine how we realize the services by leveraging patterns that have been introduced by Information On Demand (see Sect. 2.2).

Technical Feasibility Exploration

As service allocation and component layering decisions are being made there will be varying levels of confidence that the resulting design can meet all the known functional and non-functional requirements. Where uncertainty exists, further investigation is performed in the form of technical feasibility exploration. Technical feasibility exploration can commence earlier in SOMA (e.g. during service exposure decisions and service specification) but should be completed during SOMA realization to enable detailed design and implementation to take place with confidence that good logical design and technology choices have already been made.

Technical feasibility exploration may be limited to theoretical analysis of system capabilities and characteristics or can extend to physical prototyping, testing and performance benchmarking where this is needed to justify service realization decisions.

The results of technical feasibility can include:

- Confirmation that the applications, data and technology choices made will support the solution as-is and detail design can begin accordingly.
- Modifications to the planned functional design, such as data or application transformations, are needed to meet requirements for the solution.
- Alternative technology choices are required to meet all the requirements of the solution.
- Decision not to expose the planned service at all because requirements cannot be met.

6.2 Validate the Identified Information Services

During this activity, we will validate and confirm the initial categorization of services in the SOMA identification phase where we have identified an initial set of information services. The final categorization of services and

therefore the identification of information services goes hand in hand with the selection of information service patterns which is a further specialization of information services (see Sect. 6.3 on how to select information service patterns).

We start with a definition of information services and why we want to treat them differently than some of the other services (such as application services). This will help us to better understand the criteria (focus and risk areas) to identify information services that follow later.

6.2.1 Definition and Objective of Information Services

An information service decouples consumer and provider of information. It specifies a service-oriented interface to access information by leveraging various information management patterns and technologies.

The decoupling of information from applications and processes is the first and most important characteristic of information services. The concept of 'separation of concerns' is a familiar architectural principle. For example, separating data from application logic from presentation from workflow in the design of a (single) application and has demonstrated its value in many successful implementations. Does the same concept apply across the enterprise, and in particular in a SOA? Separation of concerns remains a valid and fundamental principle in SOA. For example, the enterprise service bus (ESB) plays the role of the standard hub to connect the various components through service interfaces. This is an accepted best practice in almost all SOA solutions.

Published services in the SOA solution expose information and function to their consumers. A very common implementation of services is to leverage existing application API's. The value of this approach is that existing rules governing the access and modification of business data are respected through the reuse of the responsible application code. However, the data and corresponding rules in the application have been developed to support specific aspects of the business, and do not necessarily apply equally well to all parts of the business. For example, a Sales and Distribution system has a different set of product data to a Manufacturing system for the same products – not only do the two systems contain different views of the product data but they also quite probably contain information gaps, contradictions and inaccuracies for the same products. When a business requirement demands access to information entities that are not already appropriately encapsulated within an existing application then information services provide an alternative approach.

The second important characteristic of information services is that they leverage architecture patterns from Information On Demand to realize the service i.e. information service patterns. Information On Demand has five major areas that are of importance in SOA:

- data services
- content services
- information integration services
- master data services
- analytic services

Information service patterns address how to effectively manage and provide access to structured and unstructured information that resides in a single repository or in multiple diverse repositories. The patterns explain how to cleanse, transform, and integrate heterogeneous information and to deliver it as trusted information. We leverage those information integration patterns for master data elements and add additional patterns to manage master data and keep it consistent. Finally, Information on Demand includes patterns on

how to derive additional analytical insight from structured and unstructured information and to provide them as a service.

6.2.2 Approach

Some of the activities in the SOMA specification phase may influence the categorization of services and in particular the identification of information services.

The data quality analysis (Sect. 5.3) may uncover data quality issues such as non-matching keys, duplicate entries across the legacy systems, etc. Before we can expose the information, we may need to address those data quality issues.

During the data mapping across SOA layers (Sect. 5.5), we may identify that the information that needs to be exposed in a service does not reside in a single system but in multiple databases. This may lead to information integration services as described below.

When an architect tries to identify an information services in the service portfolio, it is important to understand which areas to focus on – i.e. the criteria indicating which services to select – and where to proceed with caution.

6.2.2.1 Focus Areas When Identifying Information Services

After defining information services, this section clarifies how to identify information services from the list of services that have been defined in the SOMA specification phase. The following list includes indicators or criteria to help you in this process.

- **Reusable information access**

  Multiple business processes and/or composite services need to access the same information as part of their realization. We can separate the concerns by splitting the complete service realization into a pure information access – such as getAccount – and the business logic and workflow related tasks – such as registerAccount. This separation of concerns allows implementing the data access in one place, govern it in one place and reuse it multiple times. This helps to avoid that multiple composite services and business processes implement the same information access in different ways, leading not only to unnecessary development costs but also to possible inconsistencies.

- **Services that require information integration**

  In many SOA projects we will have to deal with heterogeneous data that is distributed across multiple systems. From an architecture standpoint, the service consumer needing integrated data should be decoupled from the component that implements the integration. This will allow us to leverage the most efficient information integration technology that can meet the functional and non-functional requirements. Such an information integration service component can then provide this functionality for multiple services that need to retrieve data from heterogeneous sources.

- **Services that access master data**

  A set of key candidates for information services are the access – read and write – to master data entities such as customer, product, etc. Master data entities play a central role in an organization: from a business perspective, they are the core entities that the business is focused on such as a product, a party (e.g. customer, member, citizen), etc. They are shared across multiple business domain systems and that link together the business data and business processes spanning multiple LOBs. From a technical perspective, almost all information in an organization is more or less directly

related to the master data entities. Since organizations have many different systems, and not just a single data store, all of those systems capture some information of a master data entity, and even if it is just the identifier of a master data entity. This will allow each system to relate its information to the core business drivers but also to the rest of the information outside its own boundaries. However, in most cases the various systems are not consistent and in particular the master data information is not consistent due to many reasons such as lack of governance, mergers and acquisistions, decentralized control and design, etc. As a consequence, access to master data requires access to distributed and heterogeneous systems. In many cases, integrated and accurate master data is not readily available without cleansing, transforming and integrating data.

- **Services that access unstructured information**

  Many organizations still rely and must continue to rely from a legal perspective on paper in some of their core business processes. Many organizations strive to optimize and automate such processes which will require services that access the paper-based information more effectively. Even if an organization's unstructured information such as media, documents, etc. is available in digital form, it does not always reside in a single repository in the format that the consumer needs. Advanced content-oriented technologies may be required to digitalize paper-based information, to integrated content from various repositories, and to provide a robust platform to manage and access/search the content. This content-oriented functionality should be separated thorugh a service interface from multiple consumers.

  Access to content can quickly become more complex than a single read/write or checkin/checkout operation and require content-centric processes that can be managed and provided by content-oriented technologies. An overall business process can include content-centric workflows as sub-processes or process services and leverage content service functionality.

- **Services that access analytic information**

  Access to analytic data such as retrieving a credit score may require advanced technology to gather the necessary raw information from disparate structured and possibly unstructured repositories and to aggregate / calculate the analytic result. Also in this case, advanced technology may need to be leveraged to most effectively gather/integrate source data and to derive the analytic result.

6.2.2.2 Risk Areas When Identifying Information Services

As with any other concept, we need to be careful when we apply this approach to ensure that it is an appropriate solution for the given requirements. The following list gives some indications on when not to realize information services.

- **Services with extensive business/application logic, presentation logic or workflow**

  We distinguish between two categories of logic:

  – (Business) logic exposed in the business process which is most likely designed by the business analyst and implemented in the process layer

  – (Data validation and application) logic encapsulated behind the service interface (such as technical rules) which is typically implemented within the application and database layers.

  Information services may be appropriate to realize some data validation rules, in particular around data standardization such as address cleansing. However, it is typically not the responsibility of data architects and corresponding developers / administrators to implement business logic nor is it their focus of expertise or the focus of the tools that they are using.

  Similarly, requirements related to the presentation of information are addressed by interaction services. They are also typically designed by application architects and are not the responsibility of data architects and their related tools. The focus on information services is to deliver the information

but not to implement how the information is displayed. The only exception are analytic services that can include specialized functionality to address specific reporting and visualization needs for data.

The orchestration of control flow across various activities is addressed by process services and are outside the scope of data architects and their tools. Information services may include limited control flow: the objective is not to address business requirements – as given by business analysts – but technical requirements to satisfy certain non-functional requirements.

- **Any access to data is an information service**

  In the same way that not every access to business logic from a portal goes through a service interface, not every access to data should go through a service interface either. It is a common practice not to expose every API call as a service and for the same reason, not every data access should be exposed as a service either. Not all CRUD access operations to information access satisfy the service litmus test which should also be applied to information service candidates. The standard SOMA approach applies also for information services: services are identified during the SOMA identification phase with an emphasis on a top-down design approach and goal service modeling. Only the services that have passed the service litmus test and are further considered then in the SOMA specification and realization phase should be realized as information services.

- **Attempt to simulate the query language at the service interface**

  A service interface should not and cannot replace a query interface and vice versa A service is specific and composable. A good service name should be sufficient to understand what the service does. Due to clearly specified and limited scope, the service guarantees that only the authorized consumers access its functionality and not beyond. This is one of the reasons why the service can be then shared broadly across and beyond the enterprise. However, a service does not and should not provide generic access to data such as “give me any information that I need in any format that I define” which is the focus of a query language. A query language has this flexibility which is required in many uses cases.

6.2.3 Deliverable

The deliverable of this SOMA task is a revised service model that identifies the information services. The GS Method work product is SOA101.

6.2.4 Dependencies

The input to this task is the service model as defined in the work product SOA101.

The output of this task is the revised service portfolio that has the information services identified. This list is required to select the appropriate information service pattern (see Sect. 6.3).

6.3 Select Information Service Patterns

After we have identified information services, we need to decide and design how to realize them. Towards the end of the macro design and before we begin with the micro design, we need to select the most appropriate solution for the given problem. We apply the patterns approach to guide this architecture decision process.

A pattern is a solution to a reoccurring problem in a given context. In this paper, we focus on architecture patterns that are product agnostic. Depending on the level of abstraction or granularity of the problem definition, multiple patterns can address the same problem or problem domain. In this paper, we focus on patterns that address the common problem domain on how to realize information services. In order to guide the architect which pattern to choose, we highlight the difference between the specific problems for each pattern and the different solution characteristics of the various patterns. The following table gives an overview of the key information service patterns.

		INFORMATION SERVICE PATTERN				
		Decouple your information from applications and processes by leveraging Information on Demand patterns				
		Content Service Pattern	**Master Data Service Pattern**	**Information Integration Service Pattern**	**Analytic Service Pattern**	**Data Service Pattern**
Problem		How to best manage (possibly distributed and heterogeneous) unstructured information so that a service consumer can access the content effectively	How can consumers access to consistent, complete, contextual and accurate master data even though the data resides in heterogeneous inconsistent systems	How to provide a service consumer access to consistent and integrated data that resides in heterogeneous sources	How to access analytic data out of raw heterogeneous structured and unstructured data.	How to expose structured data as a service
Solution		Provide a consistent service interface to content no matter where it resides	Establish and maintain an authoritative source of master data as a system of record for enterprise master data	Understand your legacy data and its quality, cleanse it, transform it, and deliver it as a service	Consolidate, aggregate and summarize structured and unstructured data and calculate analytic insight such as scores, trends, and predictions.	Implement a query to gather the relevant data in the desired format and then to expose it as a service
Differentiating Characteristics	Type of data	Unstructured information (documents, media, etc.)	Primarily but not exclusively structured data	Heterogeneous data	Analytic results based on structured and unstructured data	Structured data
	Supported functionality, type of operations	Content-centric workflow and content access (search, check-in/ check-out, etc.)	CRUD operations on master data as well as more composite master data services	Mostly read access to integrated data (incl. matching, joining), data standardization & validation rules.	Read access to analytic data	CRUD operations on structured data, mapped directly to queries / stored procedures
	Data domain	Any domain (unstructured)	Master data (product, party , agreement, address, account etc.)	Any domain	Analytic results	Any domain (structured)
	Data Sources	Any number of repositories that hold unstructured data	Many legacy applications that hold (portions of) master data	Any number of heterogeneous sources	Any number of structured and unstructured sources for analytic data	Single structured database
for details see…		Sect. 6.3.1	Sect. 6.3.3	Sect. 6.3.2	Sect. 6.3.4	Sect. 6.3.5

6.3.1 Content Service Patterns

The problem that content service patterns address is how to best manage (possibly distributed and heterogeneous) unstructured information so that a service consumer can access the content effectively. The unique characteristics of this particular problem are:

- focus on unstructured information and
- expose the content as a service.

The solution to this problem is to provide a consistent service interface to content no matter where it resides.

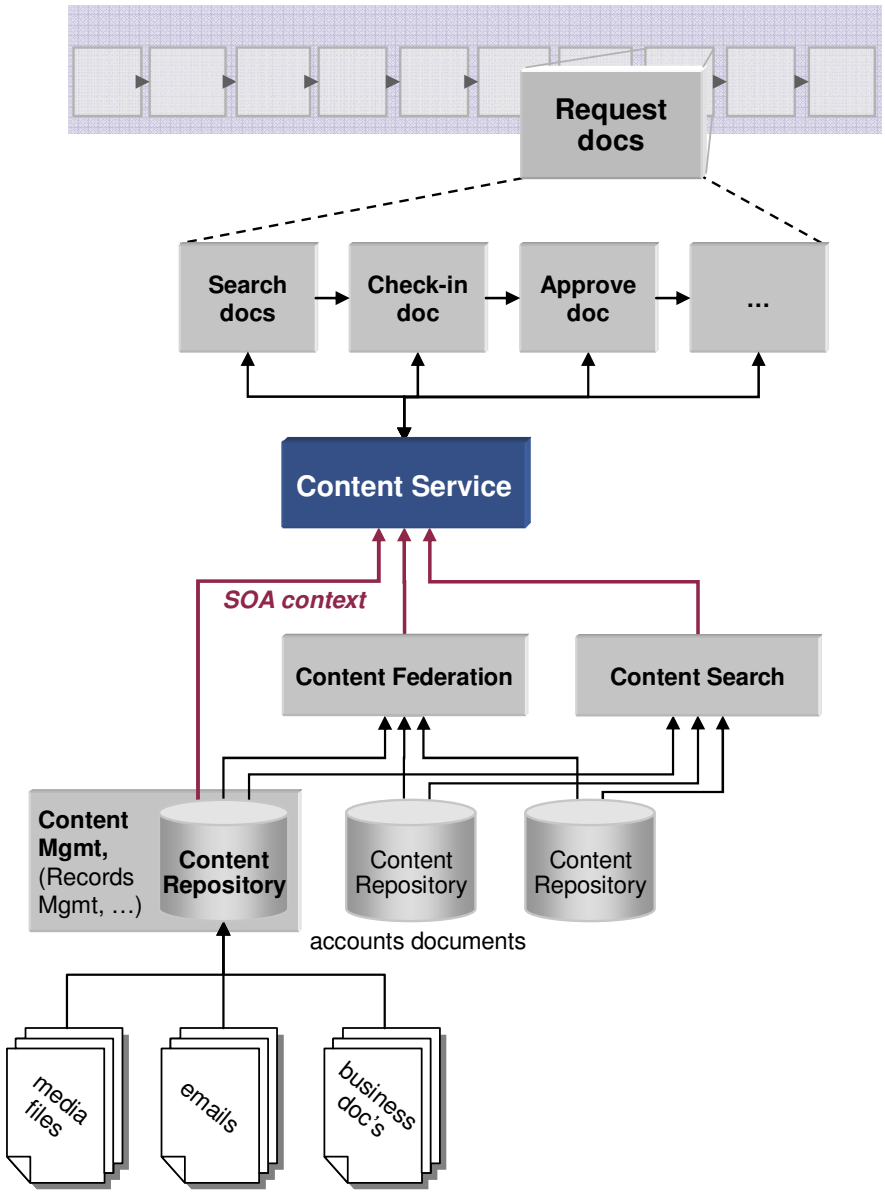

Figure 27: Content Service Pattern

There are multiple sub-patterns in this domain that address more specific problems when implementing this approach. We highlight some of the most important content service patterns here:

- **Management of Diverse Content Types**

  One of the most important content service patterns is to address the problem on how to effectively manage and access various types of content. The solution is to provide a content repository that can store different types of content and provide the appropriate services to manage the content. There is a set of common operations that are required independent of the type of the content such as store (checkin), retrieve (checkout), search, classify, etc. And then there are specific operations based on the type of content such as operations to manage documents, digital assets, (email) messages, web content etc Different content types may also require specific metadata to be associated with the content to manage it effectively. A content repository must be able to store those different types and a content management layer must provide common as well as specific operations to retrieve and manipulate the content.

- **Content-Centric Process Management**

  If the retrieval of and/or access to content must be orchestrated in a certain sequence of individual operations, workflow capabilties must be provided to support this requirement. Some business processes may have a mixture of regular operations and content-centric operations such as an insurance claims process. Various types of documents may have to be collected (police reports, accident photos and diagrams, repair estimates, etc.) and then approved through specific processes. The content-centric processes must be managed most efficiently but also related to and aligned with the overall business process.

- **Information Capture and Classification**

  Not all unstructured information is already in digital form available in a content repository. This pattern addresses the problem on how to take non digital information and to make it accessible through services. The approach is to capture the non-digital information and to store it in digital form in the content repository. The captured document must not only be stored but must be classified and annotated with appropriate metadata to enable users to find it through navigation and/or search within the context of their business role. The pattern can then be combined with the management of various content types to expose the captured information as a service.

- **Content Federation**

  How can a consumer access most effectively content that resides in diverse content repositories? The content federation pattern addresses this problem by virtually integrating the content under one common content interface. This content service interface presents the consumer with the appearance of a single content management system. However, the implementation of the interface does not retrieve the content from its own repository but from any integrated repository that is registred at the federated system. The content federation pattern from Information On Demand can be leveraged in SOA by exposing specific operations to specific content that are required by SOA consumers as services.

- **Content Archiving**

  This pattern addresses the problem of how to retain outdated unstructured information (emails, files, etc.) that are possibly distributed across diverse applications. The archival of content allows access to information that has been removed from the applications themselves in order to reduce the volume of content that they have to manage and therefore their complexity. If managed centrally, the approach guarantees a consistent process and more efficient access to archived information that can then be exposed as a service.

- **Records Management and Compliance**

  The records management pattern addresses legal concerns and regulations to ensure the availability and proper management of records. The ISO defines records as "information created, received, and maintained as evidence and information by an organization or person, in pursuance of legal obligations or in the transaction of business". Therefore, records management focuses on identifying,

classifying, archiving, preserving, and destroying records. The ISO 15489: 2001 standard defines it as "The field of management responsible for the efficient and systematic control of the creation, receipt, maintenance, use and disposition of records, including the processes for capturing and maintaining evidence of and information about business activities and transactions in the form of records".

Content services and the underlying concept and technology of content management are a broad area. The purpose of this section is not to define all the various content service patterns but to start with an introduction into this domain so that architects understand if those patterns are applicable for the specific client scenario. We conclude the considerations around content services by summarizing focus and risk areas for applying the content service patterns.

6.3.1.1 When to Apply the Content Service Pattern(s)

- Access through a service to content of various content types such as documents, digital assets, (email) messages, web content
- Leveraging content-centric processes in SOA
- Aligning business processes and content-centric processes in SOA
- Exposing as services information that is not available currently in digital form
- Service access to content that resides in heterogeneous repositories
- Access to archived unstructured information through a service
- Exposure of records related information and functionality as a service

6.3.1.2 When the Content Service Pattern(s) May not Apply

- Management of business processes that do not require access to unstructured information
- The information within the scope is not “unstructured”
- Access to derived and aggregated information that is based on enriched and processed structured and unstructured information.

6.3.2 Information Integration Service Patterns

Information integration service patterns address the problem of how to provide a service consumer access to consistent and integrated data that resides in heterogeneous sources. The unique characteristics of this particular problem are:

- focus on integration of heterogeneous (structured) data and
- expose integrated data or integration capabilities as a service.

The solution to this problem is to integrate the data from heterogeneous data sources into a common and consistent format that can then be exposed as a service to the consumer. The first major component is to understand the underlying repositories, how they map to the integrated format that the consumer expects, and if the degree of data quality in the sources meets the requirements of the project. Depending on this gap between the level of consistency in the data sources as is and the required degree of consistency, the data cleansing pattern may need to be applied to standardize structures, to reduce redundancies and to resolve inconsistencies. Various transformation and integration patterns are then applied to aggregate the data from

the various sources into a common canonical format of the service so that the integrated information can be delivered to the consumer.

The patterns below are more specific information integration service patterns that address more specific problems when integrating the data. Some of the patterns that are focused on design-time challenges have already been positioned in this paper.

- **Business Glossary Pattern (Design Time Focus)**

  When trying to integrate structured information to realize a service, it is important to agree on the meaning of the output data elements. It is in particular critical when the scope can be as broad as addressed by this pattern: any structured information that consumers need to have integrated access to. This pattern addresses the problem of how to ensure a common understanding of the terminology and its taxonomy by defining the terms and their meaning. The concept of this pattern is described in detail in Sect. 4.1.

- **Canonical Data Model Pattern (Design Time Focus)**

  When a consumer requires access to integrated data, it is necessary not only to agree on the meaning of the data but also about its structure and format. The canonical data model pattern addresses the problem of how to represent in a common (i.e. canonical) way the data as required by the consumer by defining the canonical format of the integrated data on a conceptual and/or logical level. The concept of this pattern is described in detail in Sect. 4.2 and 5.1.

- **Data Profiling Pattern (Design Time Focus)**

  Before we start designing the data transformation and integration operations that actually integrate the information from the various sources, we need to identify areas of inconsistencies in order to address them properly during the integration. The data profiling pattern addresses the problem of how to identify areas of data quality issues by analyzing the data and its relationships. The concept of this pattern is described in detail in Sect. 5.3.

- **Data Cleansing Pattern**

  The data cleansing pattern addresses the problem of how to improve data quality by specifying and enforcing data standards, and by identifying and removing data redundancies (see Reference [13]).

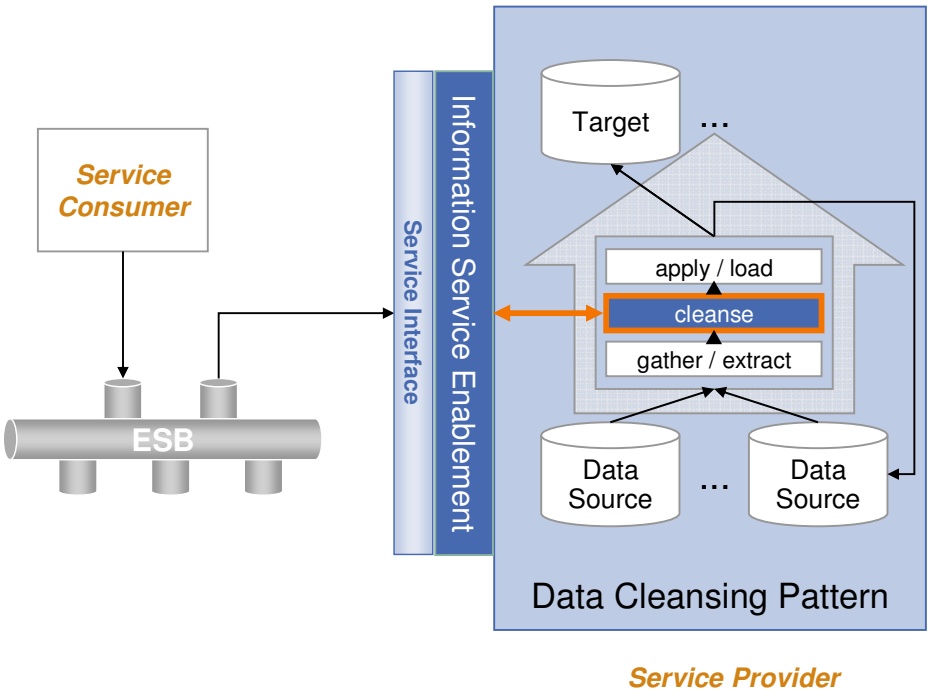

Figure 28: The Data Cleansing Pattern

Data quality issues arise when integrity rules are not specified or appropriately enforced, when repositories are created and maintained in an isolated matter, and when new external data has to be integrated e.g. due to mergers and acquisitions. Data quality issues are most severe when information is scattered across isolated and heterogeneous data stores. The heterogeneous and isolated nature of such an environment often goes along with different formats and inconsistent values. Even within a single database, the quality of persisted data can be compromised if the appropriate rules are not enforced. Whether information is at rest in a data store or manipulated on the fly by an application, data quality is often either not enforced at all or controlled (insufficiently) by different components using inconsistent rules that are embedded in application code.

Common problems include:

- missing and null values where data is expected,
- lack of standards for data formats and values,
- lack of consistent values for identifiers,
- incorrect placement of values into attributes,
- incorrect values, and
- duplicate records.

Data quality needs to be addressed by applying data cleansing consistently – i.e. using consistent cleansing rules – throughout the enterprise, not only in the database layer but also in the application and process layers.

The data cleansing pattern is defined as the standardization, clean-up, and ultimately matching (de-duplication) of records based on the content of freeform text fields. In the traditional, non-SOA context, this process is usually a batch function that is executed periodically against databases: data is extracted, modified if necessary based on the cleansing rules and applied either back to the data source or to some target system (this is illustrated in Figure 28 on the right side within the data cleansing pattern box).

The SOA context for the data cleansing pattern takes advantage of sophisticated standardization and matching techniques and extends them to the front line of near-real time applications. The data

cleansing pattern viewed in this context allows an enterprise to extend its capabilities for validation and matching to the point of creation. Furthermore, the same de-duplicating and matching logic used in batch operations can be integrated into sophisticated search methodologies or used to enhance the capability to locate customer information when information or identifiers are either unknown or incomplete.

The SOA context for data cleansing allows for standardization and matching of individual request strings. A single name or address is dynamically cleansed, returned in a standardized format, or in the case of discovery, returned along with a set of possible candidates that are identified in the matching process. In data entry solutions, this improves data representation (consistent abbreviations of street types and states, for example), and increases the odds of finding a duplicate *before it is persisted.* Avoiding the problems of duplicates in advance is far less expensive than trying to correct them afterwards, or suffering the consequence of mishandled accounts because a single view of customer was not achievable.

The design time characteristics of the data cleansing pattern pivot around establishing standard rules for conversion and cleansing of the data sources, defining matching criteria to support de-duplication, and identifying how to determine the most current or correct data. The design is the most critcal and complex phase in the data cleansing process. Once this task is complete, the application of the cleansing, matching and survivorship rules are used in the run time processes.

- **Data Consolidation Pattern (a.k.a. ETL Pattern as in Extract-Transform Load, or Data Aggregation Pattern)**

  The data consolidation pattern creates a trusted source of integrated information to service consumers when the data is distributed across significantly different sources (see also Reference [12]).

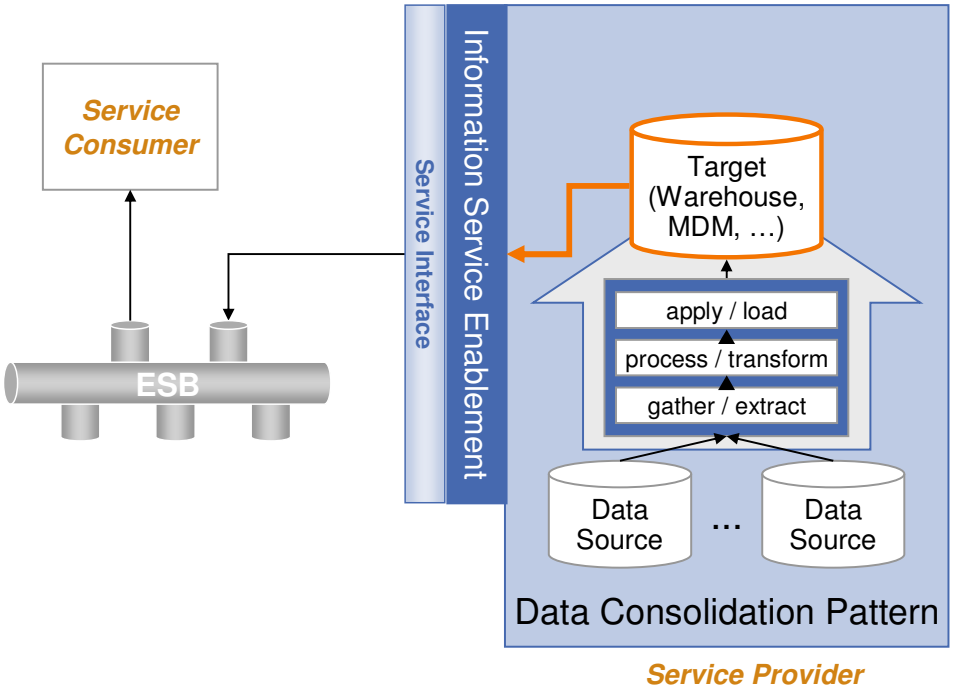

Figure 29: The Data Consolidation Pattern

In the scenario where this pattern is applied, the data sources have been designed, developed and evolved independently so that they have significantly different representations of the same types of data. The heterogeneity can occur on an instance level – e.g. lack of a common key (because of different formats) – or on a model level – e.g. the same real world entity is modeled in a different

number of database entities. Another aspect is that we may have different data populations: one database may contain all customers that have purchased the product in a retail store and another database may include customers based on e-commerce activity through the company's web site. The SOA consumers must not be aware of this underlying heterogeneity but must be able to access the integrated information transparently.

Data consolidation provides a high level of availability of integrated information. Often, source system availability is constrained due to resource utilization or maintenance schedules. At the same time, complex data processing operations may be needed in order to provide the requested service.

The goal when applying this pattern is to:

- Integrate information from sources that have a high level of heterogeneity to produce a read-only integrated information set with high availability, scalability and performance.
- Provide transformation capabilities to resolve structure and content differences between sources to produce a desired target data model.
- Decouple access to the integrated target data from the process of maintaining it in order to allow for scalability and performance.

The data consolidation pattern in the SOA context has four major phases.

i. The consolidation server – i.e. the component that implements the consolidation pattern – gathers (or "extracts") the data from the various sources.
ii. The source data is integrated and transformed to conform to the canonical model.
iii. The consolidation server applies the transformed data to a data store that realizes the trusted source of information.
iv. Access to this data store is provided through a service By pre-loading the consolidated data store with integrated data before service consumer access it, the response time of the service request is not dependent on the complexity of data transformation and integration operations.

- **Data Federation Pattern**

  The data federation pattern provides real-time access to heterogeneous data. This pattern creates an integrated view into distributed information without creating data redundancy while federating the heterogeneous data (see also Reference [11]).

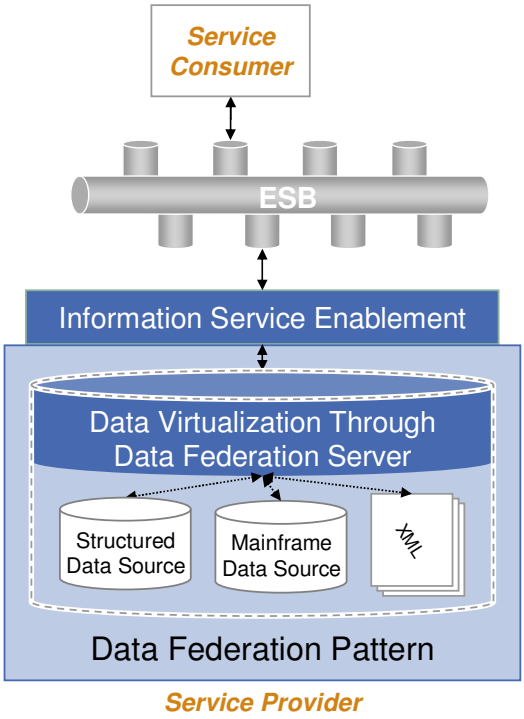

Figure 30: The Data Federation Pattern

The business drivers for the data federation pattern are very similar to the data consolidation pattern. The primary difference is that data federation focuses on real-time access to integrated data without creating data redundancy. Another important factor is that the federated approach allows the consumer to update distributed data, under certain conditions[3] even through two-phase commit.

As an example, a service `retrieveFullCustomerDetail` needs to retrieve comprehensive information about a customer and the information does not reside in a single system.

The data federation server – i.e. the component that implements the data federation pattern – provides a solution to effectively join and process information from heterogeneous sources. This pattern realizes a synchronous, real-time integration approach to distributed data. The data federation server is responsible for receiving a service request directed at an integrated view of diverse sources. It transforms it using complex optimizing algorithms that result in breaking the request down into a series of sub operations, applying the sub operations against the appropriate sources, gathering the results from each source, assembling the integrated results and finally returning the integrated results to the origin of the request. This processing sequence is done synchronously and in real time. The most current information is accessed from the various sources when the data is requested. The time that the federation server needs to translate the request and to forward it as sub-operations to the sources impacts negatively the response time.

6.3.2.1 When to Apply the Information Integration Service Pattern(s)

The criteria for when to apply information integration service patterns in general:

- A service needs to provide information that is distributed across heterogeneous sources
- A service needs to access standardized and cleansed information over inconsistent sources.

[3] If the source databases support the two-phase commit protocol.

The data consolidation pattern is best applied in the following scenarios:

- Integrating data from a wide range of sources with a high degree of heterogeneity. This approach has powerful capabilities to resolve the conflicts and merge the data together. The data consolidation pattern is often combined with the data cleansing pattern so that data quality issues can be addressed during consolidation.
- Providing integrated information for consumers that demand high data availability, high level of concurrent access, high scalability and performance. The data consolidation pattern materializes the integrated information in a new target copy that the consumers can access independently of the transformation and integration process.

The data federation pattern is best applied in the following scenarios:

- Data federation supports requirements with respect to the replicating and duplication of data by enabling access to the data as it resides in the source. These requirements can be in response to regulations or rules restrict the movement or replication of data, e.g. subscription data or the combination of personal information from different countries.
- Real-time access to distributed information as if from a single source.
- Frequently changing environment and requirements - in particular schema evolution. Due to the lack of data redundancy, changes in the federated schema reduce the impact of changes to integrated systems.
- The advantage of data federation is best exploited when a modest number of requests are received against limited results sized from multiple homogeneous, complementary data sources.

6.3.2.2 When the Information Integration Service Pattern(s) May not Apply

The list below describes criteria on when to apply information integration service patterns only with caution:

- If access (in particular write access) to distributed data needs to be performed in a specific sequence of operations and control flow needs to be orchestrated then workflow and process service approaches might be more appropriate.

The data consolidation pattern should be applied with caution in the following scenarios:

- Real-time access to distributed data that is frequently changing. Addressing this scenario with data consolidation requires frequent movement and consolidation of the source data. If the consumer rarely needs access to this integrated information, this approach might not be as cost effective as other approaches and might not deliver the data as up-to-date as the consumer expects.

The data federation pattern should be applied with caution in the following scenarios:

- Integration scenarios that require complex transformations to build the integrated view will have a negative impact on the response time particularly in this approach.
- Source servers may be negatively impacted by increased workload when they have to return data that is requested in a federated query. In order to process a request to the integrated view, the federation server will send sub operations to integrated sources. The more complex those sub operations are and the more frequently they are sent to the sources, the more additional workload the source servers need to support.
- Scenarios that result in large intermediate result sets being moved from the target data sources to the federation server may have significant network performance and response time implications.
- Situations in which applications require a relatively high degree of availability of the integrated data may not be good candidates to apply this pattern. The availability of the integrated data is wholly

dependent upon the availability of all federated and source servers involved in the process as well as availability, capacity and responsiveness of the network.

6.3.3 Master Data Service Patterns

Master data service patterns address the problem of how consumers can access consistent, complete, contextual and accurate master data even though the data resides in heterogeneous inconsistent systems. The unique characteristics of this particular problem are:

- focus on master data,
- providing the single version of the truth of master data,
- integrating and synchronizing master data that currently resides in heterogeneous, inconsistent systems, and
- exposure of master data as a service.

The solution to this problem is to establish and maintain an authoritative source of master data as a system of record for enterprise master data, which we call the master data repository. The repository is populated from and synchronized with the various systems that hold master data as shown in the figure below.

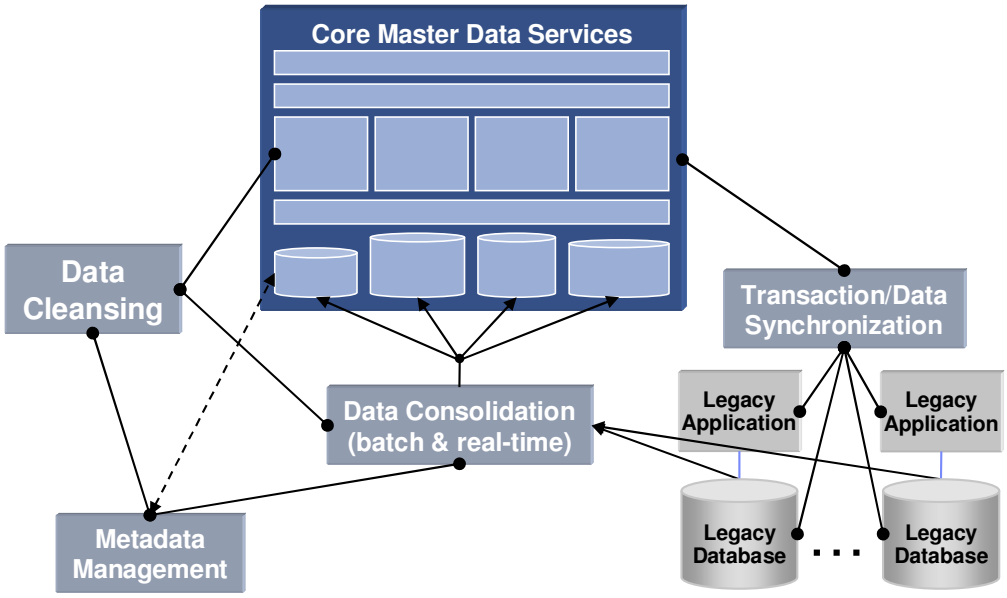

Figure 31: Master Data Management Related Patterns

The master data service pattern is a composite pattern that focuses on a particular type of data, i.e. master data, and relies on following other patterns:

- **Information Integration Service Patterns (see Sect. 6.3.2)**

  The master data management pattern addresses the problem that existing legacy applications have incomplete, inaccurate and inconsistent master data. The starting point is that a master data management system and a master data repository does not exist nor is there any repository that keeps the single version of the truth for master data. The information integration service patterns

(data cleansing, data consolidation) provide integration and data quality services so that the master data repository can be populated with cleansed master data from the various legacy sources[4].

- **Metadata Management Patterns**

  Many metadata artifacts – such as the master data model, the cleansing rules, the data integration mappings, etc. – need to be shared and their consistency needs to be enforced.

- **Transaction Synchronization Patterns**

  The master data management system will provide the single version of the truth for master data by storing the data in its repository. Most likely, the legacy applications will still maintain at least some portion of master data (due to performance requirements, high costs of changing legacy applications etc.). That means that updates to master data need to be synchronized between the master data management system and the legacy applications.

The master data management system is more than just a repository of master data. The following diagram illustrates the core master data services.

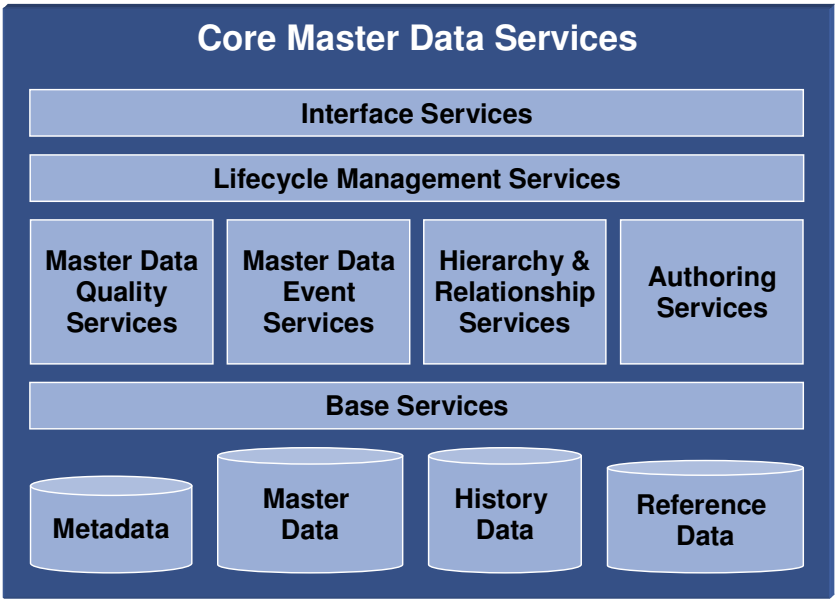

Figure 32: Core Master Data Services

- **Interface Services**

  The same service to access and manage master data may need to be exposed in various bindings or through different methods such as messaging, method calls, web services and batch processing for requesting a lifecycle management service. The same service should be invoked regardless of whether it was invoked as part of a real-time transaction or as part of a batch process in order to maintain consistent business logic.

[4] Most MDM deployments require a dedicated physical repository that maintains master data because of the inconsistency between master data in the current legacy repositories. Theoretically, MDM data may continue to remain only stored in its current locations but that requires a low degree of inconsistency or a simple identification of the current legacy system that holds accurate data. Most MDM scenarios have in common that master data is inconsistent and conflicts are not easy to resolve on the fly. Often, elements of data have to be compared and assembled in different ways to create the trusted, single version of truth.

- **Lifecycle Management Services**

  They provide CRUD, archiving and purge support for master data. When accessing master data, some of the underlying services in Figure 32 may need to be leveraged such as master data quality services, master data event services, hierarchy & relationship services and authoring services.

- **Master Data Quality Services**

  When master data is initially populated into the master data repository, the source data may need to be cleansed. After the initial load processes, the master data quality services validate and enforce that the information remains accurate when service consumers perform updates. They leverage the same cleansing rules as defined during the initial load of master data. The master data quality services apply the data cleansing pattern as defined in Sect. 6.3.2.

- **Master Data Event Services**

  They are used to make information actionable and trigger operations based upon events detected within the data. Events can be defined to support data governance policies, changes to critical data, based upon business rules or time and date scheduled. An important use cases for this capability is the synchronization of master data with external systems. The events help to trigger the synchronization processes.

- **Hierarchy and Relationship Management Services**

  Master data management is broader than the management of a single master data entity: it is important to manage hierarchies of master data entities – such as different types of customers and the generalization of a customer to a party – as well as relationships between master data entities – such as the relationship between the organization itself and external organizations such as suppliers, business partners, etc.

- **Authoring Services**

  They provide services to author, manage, customize and extend different master data objects such as product and supplier.

- **Base services**

  They are available to support micro workflows, security and privacy, search and audit logging. Base services can be implemented to integrate with enterprise common services for workflow, an enterprise LDAP and audit logging.

- **Master Data Repository**

  It consists of both instance and metadata that describes master data, taxonomies, and rules for data validation, and history data that records changes to master data.

6.3.3.1 When to Apply the Master Data Service Pattern(s)

- Multiple consumers need access to a single set of consistent master data but the data sources have conflicting information (incl. inconsistent identifiers).
- When a consumer needs to obtain a comprehensive view of a master data entity (e.g. customer).
- Core shared entities exist for which there is no agreed ownership process and lifecycle management in the organization. Different systems arbitrarily create and maintain instances of the entity with inadequate control over how that is synchronized with other systems.
- An existing MDM solution is in place (now or planned) and can be leveraged to provide services needed by the SOA project.

6.3.3.2 When the Master Data Service Pattern(s) May not Apply

- Data quality issues exist only within a single application: the data cleansing pattern (see Sect. 6.3.2) would be more appropriate for a single application or data source. Master data management is more appropriate for resolving data conflicts across systems..
- Not all data is master data. If the scope of the data becomes too broad, it may lead down a path where the master data repository ultimately will become the single repository for all data which is not practical.

6.3.4 Analytic Service Patterns

The analytic service pattern addresses how to access analytic data out of raw heterogeneous structured and unstructured data. The solution to this problem is to extend the data consolidation pattern with functionality to aggregate and summarize data and apply complex or specialized analysis such as statistical, probabilistic, exploratory, geospatial, and pattern recognition.

The unique characteristics of this particular problem are:

- focus on analytic data,
- expose analytic data as a service.

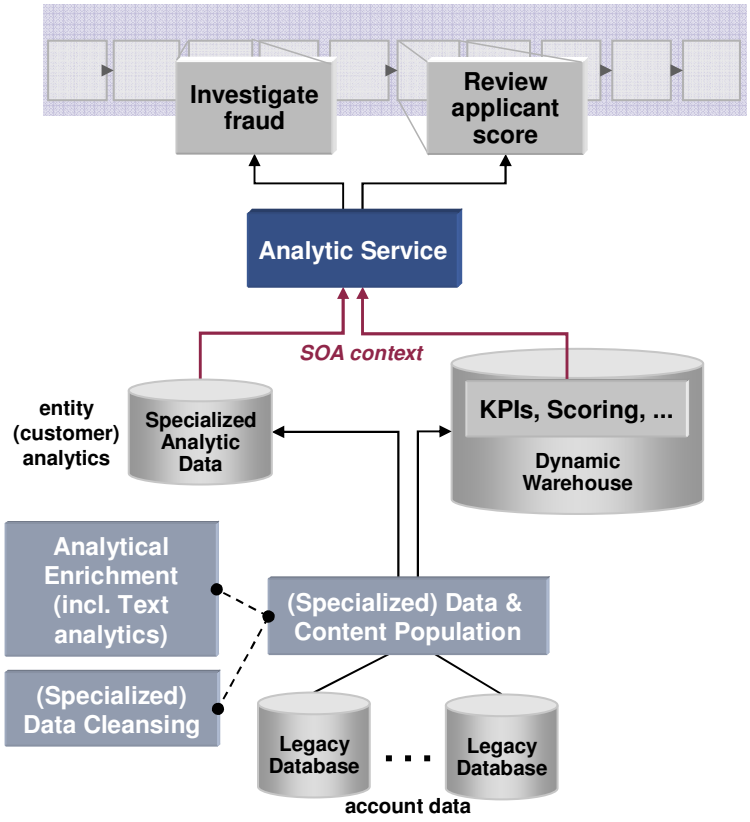

Figure 33: Analytic Service Patterns

The data consolidation pattern gathers and extracts the relevant raw input data – structured and unstructured –from various sources and then loads it into a specialized analytical data store such as a data warehouse. The analytic functions – aggregations, scores, etc. – are then performed on the consolidated data either when the data is loaded into the store or when the consumer requests the data. The analytic data is then exposed as a service. For example, a credit score service will invoke an algorithm to calculate the score based on various raw input data that was previously collected from various account applications and stored in the warehouse.

6.3.4.1 When to Apply the Analytic Service Pattern(s)

- Promote existing data analysis capabilities from an organization's back office into real-time services exploited in front office and automated business processes.
- Expose access to summarized or aggregated information as a service
- Leverage data mining / scoring algorithms to expose analytic data as a service

6.3.4.2 When the Analytic Service Pattern(s) May not Apply

- Applying analytic techniques against data that has not been carefully prepared can produce unpredictable results e.g. calculating an average when some values are missing.
- Complex analysis in real-time may not deliver acceptable response times

6.3.5 Data Service Patterns

Data service patterns address the problem of how to expose structured data as a service. The unique characteristics of this particular problem are:

- focus on structured data,
- exposure of structured data as a service.

The solution to this problem is to implement a query to gather the relevant data in the desired format and then to expose it as a service.

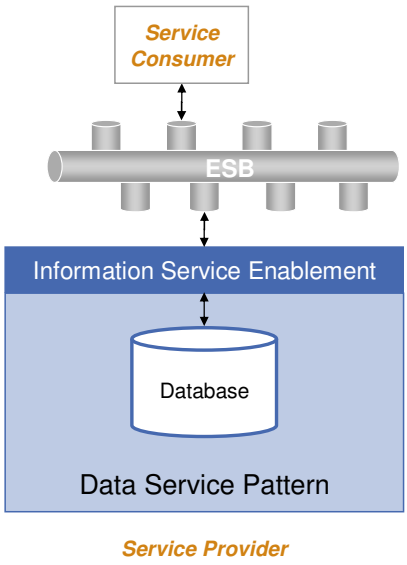

Figure 34: Data Service Pattern

6.3.5.1 When to Apply the Data Service Pattern(s)

- Access to specific data needs to be exposed as a service to a broad range of consumers
- Data exists in application systems for which there is no possibility to expose application API based services e.g. no public API's exists and no source code is available.

6.3.5.2 When the Data Service Pattern(s) May not Apply

- Same concerns as with information services in general: the SQL and the service interface have different characteristics and should complement but do not replace each other

7. Summary

In this paper, we have explained when to apply the SOMA 3.1 Information Discipline. We have introduced the following new or enhanced activities according to the SOMA phases:

SOMA Identification Phase

- Create a business glossary
- Create a conceptual data model
- Create a system context diagram
- Create a catalog of data stores
- Identify information services
- Service litmus test (including the information criteria)

SOMA Specification Phase

- Create a logical data model
- Update the business glossary
- Commence data quality analysis
- Create a canonical message model
- Begin data mapping across SOA layers

SOMA Realization Phase

- Validate the identified information services
- Select information service patterns

The table on the following page summarizes those activities and illustrates their relevance. We distinguish between core activities – that should be incorporated in any SOMA engagement – and optional activities (core/optional column). We also differentiate between activities that are applicable for any SOA design approach irregardless of the type of service and those activities that are specific for information services (scope column). We also summarize some of the statements from previous sections regarding the impact of not having this work product and reasons for not needing this work product.

Activity	Core?	Scope	Impact of not having this activity	Reasons for not needing this activity
SOMA identification phase				
Business glossary	core	SOA design	• Risk of missing fundamental information requirements of the business • Cost in moving forward with projects where the information requirements are not well understood • Cost in time expended reworking unclear and/or misunderstood requirements • Cost in credibility, business error recovery, system rework up to lost business when delivered systems do not match the needs of the business	• The only time this work product is not needed is when the work being done is purely technical in nature, e.g. simply re-architecting a middleware layer
Conceptual data model	core	SOA design	• Lack of enterprise wide, system-independent definition of business entities and their relationships • Risk of limited extensibility and inconsistency between information domains • Higher costs for dependent activities such as catalog of data stores and in particular data quality analysis	• Data model already exists • Limited scope such that detailed analysis of service interfaces will not be performed (e.g. integration of pre-existing or outsourced services)
System context diagram	core	SOA design	• In the early stages of a project, without an agreed-on context for the system, it is difficult to define the boundaries of the project effort • There is risk of either expanding the development effort into areas that are not part of the system or of overlooking areas that should be developed	• The system is not complex and has no external interface or data conversion requirements • The client or another third party vendor developed this work product and the content has been shared with the current project team
Catalog of data stores	optional	SOA design	• Current environment is not well enough understood before designing the solution • Ability to access and share common data is limited	• If data requirements are not considered to be challenging • If data access is 'unsophisticated' (e.g. where a major portion of the data is 'read only' and stable)
Information service identification	core	SOA design	• Complicated project planning due to unknown service implementation choices until after SOMA realization is complete	• The project does not include implementation. • The goal of the project is to exclusively assess implementation options • The goal of the project is to deploy infrastructure with business process and services for test cases only

Activity	Core?	Scope	Impact of not having this activity	Reasons for not needing this activity
SOMA specification phase				
Logical data model	optional	SOA design	• No detailed unified view of data • The canonical message model cannot be built; it cannot be canonical without an underlying and logical data model • Data normalization not possible	• Data model already exists or the project is adopting and industry standard model • Limited scope such that detailed analysis of service interfaces will not be performed (e.g. integration of pre-existing or outsourced services)
Data quality analysis	core	SOA design	• Success of the project depends upon unknown quality of data • Service designer is dependent on opinion and anecdotal evidence to support design decisions regarding data quality • Any undiscovered issues at design time will surface in testing or even after deployment making corrective action much more expensive	• Adequate data analysis information is already available. • The data is well understood and will be exposed through an already established interface that has widespread business acceptance • The business expectation for the data quality is low. • The business is willing to take the data “as-is” • The data is being accessed from a system or supplier over which the project has no access for analysis and/or no influence to make changes.
Canonical message model	core	SOA design	• Proliferation of messages which offer little potential for reuse and carry a high maintenance cost. • Inconsistent message types, formats, and semantics in the solution • Slower development and higher costs for new messages, services and business processes, lack of reuse • Detailed variations analysis is not possible.	• An enterprise message model already exits • The scope of the engagement is limited such that detailed analysis of service interfaces will not be performed.
Data mapping across layers	optional	SOA design	• Increased development times for the solution or even runtime errors if data mismatch errors are not identified during design.	• This activity may not be required if the solution is entirely new and there is no data sharing with existing or external components.

Activity	Core?	Scope	Impact of not having this activity	Reasons for not needing this activity
SOMA realization phase				
Select information service pattern	optional	information services	• Risk of selecting inappropriate products and designs for information services that will not meet requirements • Risk of implementing services inconsistently and thereby failing to achieve reuse and increasing maintenance costs	• Project scope does not include development of new services – only existing, purchased, or outsourced services will be used • SOA solution does not include information services • Options may be limited due to project scope, available budget, and/or client's IT strategy that may exclude a pattern.

8. Appendix

8.1 References

[1] IBM – “Service-Oriented Modeling and Architecture” – Technique Paper – Version 2.4, December 15, 2005

[2] IBM – “Data Modeling Best Practices – Methods, techniques and approaches” – Document version 1.1, 16-October-2005, Anthony Giordano - Global Business Services – Business Intelligence Practice

[3] IBM – “SOA Solution Stack 2.0 – CommunityPaper Version 2.0”, November 7, 2006, Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, Kishore Channabasavaiah

[4] IBM Attributes & Capabilities Study, 2005; Client Interviews 2004; IBM CFO Study, 2006

[5] Companion white paper to this technique paper

[6] Global Business Services - Business Intelligence Practice: Data Quality Best Practises, Methods Techniques and Approaches. Document Version 1.0, 01 October 2005

[7] Checklist for “SOAbility” of data: Real-life data constraints to semantic inter-operability, presented at IBM Academy of Technology Conference, 2006, “SOA: The Information Architecture Perspective”, Claude Blouin

[8] IBM Cross Industry Data Model

[9] John Zachman “Enterprise Information Architecture”

[10] "Enterprise Information Modeling" Technique Paper

[11] “Information service patterns, Part 1: Data federation pattern” – developerWorks Article – July 28, 2007

[12] “Information service patterns, Part 2: Data consolidation pattern” – developerWorks Article – December 5, 2006

[13] “Information service patterns, Part 3: Data cleansing pattern” – developerWorks Article – April 6, 2007

8.2 Index

8.3 List of Figures

