Open Group Guide

Legacy Evolution to SOA

Open Group Guide

Legacy Evolution to SOA

ISBN:	1-937218-05-8
Document Number:	G122

Published by The Open Group, April 2012.

Comments relating to the material contained in this document may be submitted to:

The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom

or by electronic mail to:

ogspecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 400 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

- Capture, understand, and address current and emerging requirements, and establish policies and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Offer a comprehensive set of services to enhance the operational efficiency of consortia
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Open Group Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

Readers should note that updates – in the form of Corrigenda – may apply to any publication. This information is published at www.opengroup.org/corrigenda.

This Document

This document is an Open Group Guide to Legacy Evolution to SOA (L2SOA). It has been developed by the L2SOA project of the SOA Work Group, a Work Group of The Open Group.

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this document:

Authors

- Xinfeng Chen, Hewlett-Packard
- Sundararajan Ramanathan, Capgemini
- Mukund Srinivasan, Capgemini
- Joost Van Der Vlies, Hewlett-Packard (Project Chair)
- Tejpal (TJ) Virdi, The Boeing Company (Project Chair)

Reviewers

- Ramagopal Allampalli, Cognizant Technology Solutions
- Jim Basler, Oracle
- Carleen Christner, Hewlett-Packard
- Pascal Dussart, LoQutus
- Ed Harrington, Architecting-the-Enterprise
- Venkata Kona, Wipro
- Martin Neuhard, Hewlett-Packard
- Carol Wang, Hewlett-Packard

Referenced Documents

The following documents are referenced in this Guide:

[1] Gartner Glossary; refer to: www.gartner.com/technology/it-glossary/#11_0.

[2] Definition of SOA: The Open Group; available from: www.opengroup.org/soa/soa/def.htm#_Definition_of_SOA.

[3] Open Group Standard: SOA Reference Architecture (C119), published by The Open Group, December 2011; refer to: www.opengroup.org/bookstore/catalog/c119.htm.

[4] Open Group Standard: TOGAF Version 9, published by The Open Group; refer to: www.opengroup.org/togaf.

[5] Open Group Standard: The Open Group Service Integration Maturity Model (OSIMM) (C117), published by The Open Group, November 2011; refer to: www.opengroup.org/bookstore/catalog/c117.htm.

[6] Open Group Standard: SOA Governance Framework (C093), published by The Open Group, August 2009; refer to: www.opengroup.org/bookstore/catalog/c093.htm.

[7] White Paper: World-Class Enterprise Architecture (W102), published by The Open Group, April 2010; refer to: www.opengroup.org/bookstore/catalog/w102.htm.

[8] NCOIC Legacy Services Capability Pattern 20091211, v1.10.

[9] SOA Patterns: www.soapatterns.org.

[10] The Open Group Security Forum, a Forum of The Open Group: www.opengroup.org/security.

[11] IBM developerWorks: Design Strategies for Legacy System Involvement in SOA Solutions: www.ibm.com/developerworks/webservices/library/ws-soa-legacy.

[12] White Paper: Security Principles for Cloud and SOA (W119), published by The Open Group, December 2011; refer to: www.opengroup.org/bookstore/catalog/w119.htm.

[13] US Homeland Security: Build Security In; refer to: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/legacy/623-BSI.html

[14] US Homeland Security: Build Security In; refer to: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/legacy/624-BSI.html.

[15] Open Group Guide: Using TOGAF to Define and Govern Service-Oriented Architectures (G113), published by The Open Group, November 2011; refer to: www.opengroup.org/bookstore/catalog/g113.htm.

[16] Open Group Standard: Service-Oriented Architecture Ontology (C104), published by The Open Group, October 2010; refer to: www.opengroup.org/bookstore/catalog/c104.htm.

[17] The Carnegie Mellon Software Engineering Institute (SEI): Service Migration and Re-use Technique (SMART); refer to: www.sei.cmu.edu/library/abstracts/reports/08tn008.cfm.

1 Introduction

1.1 Objective

The Legacy Evolution to SOA (L2SOA) Guide, as an addition to Using TOGAF to Define and Govern Service-Oriented Architectures (TOGAF SOA Guide) [15], contributes to The Open Group vision of Boundaryless Information Flow by leveraging and fostering common understanding of L2SOA. Its main goal is to leverage the collective experiences of L2SOA practitioners to develop legacy evolution best practices and lessons learned to improve the success of L2SOA implementations. The content is therefore based on real projects.

The document describes the following:

- The background of this Guide, including current state problems and issues
- Key concepts, principles, and considerations related to L2SOA
- A high-level (general) approach to enable L2SOA
- Consolidated best practices, including metrics, architecture styles, technologies, governance, etc.
- Historical case studies providing insight into how some of the concepts described in the document are currently being applied

1.2 Overview

Legacy systems are the operations and knowledge backbone of organizations. They are difficult to replace or modernize and crucial to business survival. According to the Gartner Group, it is estimated that more than 70% of corporate data still resides on legacy systems.

These legacy systems are mature systems, created in an era where integration, flexibility, and company mergers were not as prominent as they are today.

Legacy systems have positive and negative qualities:

- They are reliable (securing daily, supporting primary operations), efficient (developed for a dedicated environment), suitable (implementing company-specific know-how), and valuable (distinguishes companies, company-specific business logic).
- They are also complex (many changes over time, almost no documentation), hard to modularize and integrate, and expensive to maintain and enhance.

Companies in all industries and of all sizes have expressed a growing need for migrating their existing application assets into new architectures guided by SOA concepts. The rationale is based on:

- Financial considerations (protecting investments, leveraging existing assets)
- Employee skill migration and improvement (rather than replacing people)
- Stability (retaining operational software benefits, maintaining required high uptime)
- Evolutionary renewal (enabling new and emerging technologies)

Service-Oriented Architecture (SOA) as a style enables a staged transition from a (partly) silo-based system landscape towards an integrated, componentized, and shared service environment. Legacy systems can be integrated and migrated into a modernized landscape using SOA. In this way a big bang modernization scenario is not necessary, but can be done in phases.

This document is based upon the experience of a set of L2SOA projects and organized as follows:

- Chapter 2: Common concerns and drivers for modernization.
- Chapter 3: Approach to enable L2SOA, aligned with the TOGAF Architecture Development Method (ADM).
- Chapter 4: Overview of a set of best practices in a variety of L2SOA evolution aspects.
- The document ends with a detailed description of two L2SOA cases, including recommendations to improve the specific journey, based upon this Guide.

1.3 Terminology

Legacy Application or System

Gartner defines a legacy application or system as: "*an information system that may be based on outdated technologies, but is critical to day-to-day operations. Replacing legacy applications and systems with systems based on new and different technologies is one of the information systems (IS) professional's most significant challenges. As enterprises upgrade or change their technologies, they must ensure compatibility with old systems and data formats that are still in use.*" [1]. This definition directly highlights the importance of these systems, but also the challenges for transformation towards new technologies.

The SOA Reference Architecture [3] describes legacy as: "*a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable applications. New applications should use alternative means of obtaining equivalent functionality.*".

Interestingly, this definition talks about retaining only for compatibility and limitations for portability, while Gartner focuses on the business-critical value. Both perspectives need to be taken into account.

SOA

The Open Group defines Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation [2]. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services. An architectural style is the combination of distinctive features in which architecture is performed or expressed.

The Open Group SOA Reference Architecture [3] provides layers and building blocks to enable SOA within an organization and guidelines for making architectural, design, and implementation decisions.

Legacy to SOA (L2SOA)

L2SOA is an approach to enable legacy systems to be integrated into a silo-transcending landscape and processes using SOA as the architectural style. This resulting goal of L2SOA is to reach a more adaptive and agile environment, providing an infrastructure for controlled transformation, re-use of important business functionality, etc.

1.4 Future Directions

The content of this Guide is input for the further detailing of the TOGAF SOA Guide [15], which will benefit from including L2SOA. Further detailing of aspects in this Guide into specific guidelines is possible depending on the need for it. There is no commitment to any particular additional content and other content not mentioned here may be added.

2 Motivation

2.1 Common Concerns and Drivers for Modernization of Legacy Applications

Modernization of legacy systems results in a variety of different projects depending on technologies used, the business they support, and new technologies and effort needed. If all of the modernization projects are taken into consideration, then common concerns and drivers for modernization may be identified.

2.1.1 Business Concerns

Business concerns include:

- Costs associated with maintaining and enhancing the legacy applications are escalating.
- There is slow time-to-market for new products because of the many changes in applications which take a long time.
- It is unclear just how one or more legacy applications directly relate to business problems and/or solutions.
- It is unknown what the complete set of dependencies is between all the systems, resulting in a high risk of downtime when implementing changes, which can lead to business (and sometimes reputation) loss.
- No sufficient business process metrics and analytics from legacy systems.
- Because of the abovementioned concerns, the technology deployment of legacy systems needs a risk management approach and mitigation, which is resource-intensive and creates tension.

These concerns are not meant to diminish the value of the systems they represent. Most banks, airlines, credit card companies, etc. still rely on these systems for their core business and processes. Mainframes are proven transaction systems, processing a large amount of data. What these concerns indicate is that maintenance costs are getting higher and higher, control over the application landscape and its dependencies is difficult, and the business needs to change faster than IT can support.

2.1.2 Functional Concerns

Functional concerns include:

- Applications are developed for silo domains or silo functions, while there is a drive to integrate functionality in enterprise-wide business processes.

- Data is not easily shared.
- Applications (and so functionality) are not easily re-usable. This also limits the support of new business concepts or functionalities.
- The same functionality may have been implemented in more than one application.
- Data is shared between applications mostly through replication (and therefore stored in more than one location).
- There is high risk of data inconsistency since one data entity can be modified in more than one application.
- There is no automation of business processes other than, for example, database triggers, hard-coded calls, etc.
- There is little or no documentation of business-critical systems available.
- There are considerations for product lifecycle (i.e., when does the product reach its end of serviceable life?).

2.1.3 Usability Concerns

Usability concerns include:

- There is no Single-Sign-On (SSO) support. The user has to log in to different applications, sometimes with different credentials, in order to participate in a business process.
- There is poor usability due to basic screens.
- There is inability to integrate with new user technologies (for example, social media).
- There is a need to stay current with technologies to provide intuitive user interactions.

2.1.4 Technical Concerns

Technical concerns include:

- Applications are tightly-coupled (integration is "hard-coded").
- Data is sometimes not stored in databases but in files instead. This makes it more difficult to both access information in the file structure and to scale the repository to support re-use in a service-oriented environment.
- Embedded logic (e.g., business rules, security, integration logic) in applications limits re-usability and increases difficulty of maintenance.
- Integrations are often point-to-point integration which makes it difficult to re-use.
- Old technology requires constant upgrading, but upgrading may no longer be possible due to deprecated technology (i.e., the version is no longer supported by the vendor) or worse, the vendor has ceased to exist.

- Application source code language may not support modern communication standards and protocols, therefore limits the ability for legacy applications to enhance and leverage.
- The Application Architecture itself is "out-of-date"; there is no component-based design for interoperability.
- Systems configurations are not easy to change; hence it impacts the business utility.
- There is an increase in the number of incidents and application downtime as a result of several of the listed concerns.

3 Approach to Enable L2SOA

Legacy systems come in all varieties, be it different languages, technologies, exotic vendors, etc. Therefore, not all transformation or enablement projects are the same. But when we look at these projects, we can identify some recurring high-level steps that are naturally present even when the actual content or activities within them are different.

The TOGAF Architecture Development Method (ADM) provides a process to facilitate and manage architectural development and transformation projects (Figure 1). Combined with the TOGAF SOA Guide [15], it provides the foundational process for L2SOA transformation engagements. This can be done as part of an enterprise architectural engagement, or as a separate architecture cycle which specifically focuses on L2SOA.

Using the ADM phases and the TOGAF SOA Guide, the important aspects to consider from an L2SOA point of view are outlined in this chapter. These aspects can be used together with the more extensive guidance in the TOGAF SOA Guide as a process for L2SOA architecture cycles.

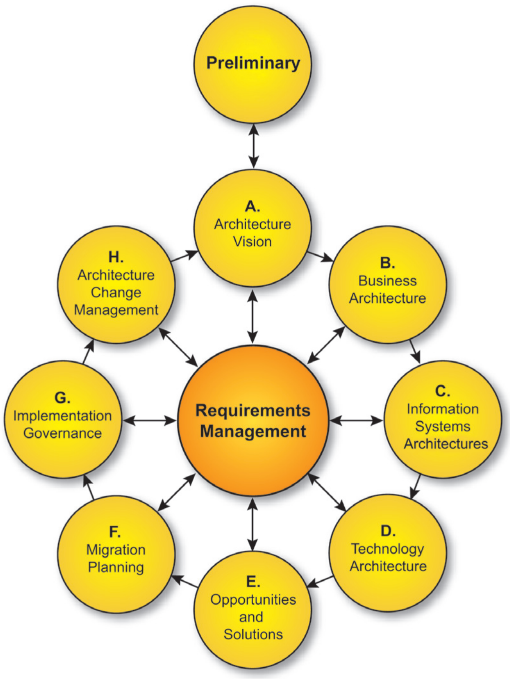
Preliminary
A. Architecture Vision
B. Business Architecture
C. Information Systems Architectures
D. Technology Architecture
E. Opportunities and Solutions
F. Migration Planning
G. Implementation Governance
H. Architecture Change Management
Requirements Management

Figure 1: The TOGAF ADM

3.1 The Preliminary Phase

Successful SOA depends in part on the readiness of the enterprise to become service-oriented. Peruse any existing SOA maturity assessment report to arrive at the SOA readiness of the organization and in particular for the legacy application(s) under consideration. One of the results would be an assessment of the "fit-for-SOA" state of the current legacy architecture, with gaps described from a maturity perspective. If there is no existing SOA maturity assessment the advice is to conduct one; for example, by using The Open Group Service Integration Maturity Model (OSIMM) [5], an Open Group Standard.

It is also important to note in this phase that the architecture governance and support strategy need to be confirmed. This will need to include the importance of legacy governance in transition stages from legacy to SOA.

With regards to the Architecture Repository and Architecture Building Blocks (ABBs), we suggest looking at The Open Group SOA Reference Architecture (SOA RA) [3] to initially populate the Architecture Repository. Note that the Operational Systems Layer in the SOA RA hosts the legacy systems, and the SOA RA describes "access services" which use service enablers and adapters to enable legacy functionality to be accessed as services.

When establishing the architecture team, make sure to include representatives from the legacy environments for collaboration on the transformation.

The TOGAF SOA Guide suggests establishing a Center of Excellence for SOA. This would be a good suggestion to examine in order to concentrate L2SOA expertise, foster re-use, and potentially eliminate both application and infrastructure redundancy.

3.2 Phase A: The Architecture Vision

A new architecture lifecycle starts with the Architecture Vision which is concerned with establishing the architecture project and obtaining approval to proceed.

One of the major drivers of an L2SOA initiative is cost reduction, so a return-on-investment analysis is important. A business case is a mechanism to support rationalization of the investment process and stakeholder buy-in. This is an essential step in order to obtain approval. Business planning and portfolio management efforts are to be performed by the business responsible for managing the business capabilities when transformation of their legacy business-critical systems is being performed. This needs to be addressed in the Architecture Vision.

In Phase A, a business-oriented decision (or principle) needs to be made to extend the life of legacy systems, or to focus on replacement. This is important for the Target Architectures and the Opportunities and Solutions phase (Phase E). The high-level description of the final architecture in the Architecture Vision will need to take this into account.

This phase also needs to lay the groundwork for non-functional requirements such as: creation/use of legacy modernization guidelines, identification and selection of legacy integration patterns and selected tools, maintainability, supportability, performance and security, extensibility requirements, data and information protection, and information assurance; for example, refer to work by The Open Group Security Forum, The Open Group RTES Forum, and The Open Group Trusted Technology Forum (OTTF), all Forums of The Open Group.

3.3 TOGAF 9 ADM (Phases B, C, and D)

In these phases, the as-is (current state) and to-be (target state) Business, Information Systems, and Technology Architectures are developed.

Current State Architecture

For L2SOA, the current state assessment of the application landscape needs to be done in sufficient detail to be able to identify the legacy systems and functionality decomposition and the integration to/with other application systems. The usage relation from legacy functionality to current state business services and business processes needs to be identified, to know which business will be impacted in a transformation.

The to-be migrated or modernized (e.g., SOA-enabled) legacy applications and their functionality need to be identified in more detail (logical application components, information system services), to prepare for gap analysis with the Target Architecture. Also, an analysis of the information elements managed within the legacy systems is important.

With regards to legacy systems, a first analysis can be done if systems can be retired, consolidated, re-factored (SOA enablement), or replaced. This will need to be detailed and finalized in the Opportunities and Solutions phase (Phase E).

Target State Architecture

Potential (groups of) services need to be identified which fulfill the required functional needs and flexibility. This starts from the Business Architecture, in which business services are identified. Information system services have to be identified to support the business services, using the approach, SOA principles, and views as described in the TOGAF SOA Guide [15]. This enables the identification of gaps between functionality provided by current legacy systems and what is needed from a Target Architecture perspective.

L2SOA needs to align with business and IT strategies, and therefore needs to consider a portfolio rationalization assessment as input for the target state architecture, as well as application portfolio management. The portfolio rationalization assessment provides input on systems that can be retired, consolidated, re-factored, or replaced. Application portfolio management provides input on tactical increments that are in-flight and planned.

Depending on principles like protecting return-on-investment and others, the Technology Architecture needs to contain technology for both the SOA infrastructure components, as well as legacy integration technology (e.g., providing specific adapters, emulators, source convertors, etc.). The Open Group SOA Reference Architecture (SOA RA) [3] is a starting point for that. Also evaluate the need for and type of tooling to support/enable legacy migration/modernization.

3.4 Phase E: Opportunities and Solutions

The Target Architecture will be divided into several Transition Architectures to gradually transform from legacy under architectural governance control. This phase consolidates the gap analysis from Phases B to D, identifies the Solution Building Blocks (SBBs) and the projects that will deliver them. It is here that the final decisions are made about which specific legacy systems will be transformed towards SOA style and in which priority (e.g., first quick-win with screen scraping, but further work on implementing a service bus on which the legacy systems will be connected).

To assess the impact of the SOA migration and to communicate the various aspects to and with stakeholders, it is advised to create an additional SOA migration planning view on the defined Transition Architectures (as preparation for the next phase). The view describes enterprise-wide impact for options considered (Transition Architectures with options/impact). The view critically examines options considered for migration paths and throws light into enterprise-wise impact, pros, cons, and work packages for each option with stakeholders weighing in on the best possible migration path given the opportunity and situation. The final target state options, as planned in the next phase, will contain strawman views of all the baseline applications carried forward, baseline applications burned down, and target applications introduced and carried during increments.

Specific vendors need to be chosen to provide tools for legacy integration, SOA infrastructure components, and to replace legacy systems (with, for example, a Software as a Service (SaaS) solution, if relevant). The TOGAF SOA Guide [15] details the various SOA-specific deliverables for this phase.

3.5 Phase F: Migration Planning

With L2SOA, the transformation is at least two-fold:

1. The design and build of an SOA infrastructure
2. The modernization of the legacy applications itself

The latter will not be able to succeed if the former does not have a minimum level of implementation; this is important for planning.

The planning can take into account various stages of SOA maturity and a phased approach to modernize the legacy systems. For example, first retire systems (quick cost reduction), then integrate legacy user interface in an existing portal, consolidate systems, then open up some transactions as web services, etc. Note that for the integration of legacy systems in an SOA infrastructure a proof-of-concept is advisable to get familiar with the integration technology.

Also, the governance model needs to be reviewed again, especially now that it is clear which legacy systems are affected, and therefore also the relevant business domains.

3.6 Phase G: Implementation Governance

It is very important to keep working with the relevant legacy system specialists, and have them participate in the architecture team.

Especially when the transformation is at the program level, it should be supported by a change program. Current operations and departments will be affected and need to be managed and guided for this change. Section 4.8 describes further organizational impacts and suggestions.

Section 4.9 discusses additional governance-related considerations and challenges, along with recommendations for overcoming the issues.

3.7 Phase H: Architecture Change Management

In this phase, changes from various inputs are taken into consideration regarding the effect on the architecture. Depending on the assessed type of change, a decision can be made to start a new architectural cycle for a (set of) legacy application(s) to transform towards an SOA (e.g., changes in the business environment, new technologies). It can also be decided to implement a change request without a cycle if the change is small and with no or minimal impact. A new cycle will take the existing defined architectural views and artifacts as input, together with the new requirements.

At a certain time it is advisable to again conduct an SOA maturity assessment, using The Open Group Service Integration Maturity Model (OSIMM) [5], an Open Group Standard. This manages and justifies the transformation process, for L2SOA evolution.

3.8 Additional Considerations

3.8.1 Where to Start

There are different starting points possible to identify which legacy applications to transform towards an SOA. For example:

- From a functional perspective, select the business services and processes and information system services (or define new services) which need more flexibility, re-usability, etc. and then relate the legacy systems that need to transform to their respective services.
- From a technology perspective, select the applications to modernize based on the age of the technology, availability of new technologies to service enable them, etc.

3.8.2 Testing

From experience with legacy modernizations, testing is a very important activity and sufficient time needs to be planned for it. Some systems have been rebuilt in a new language based upon newly written documentation by using and observing the legacy system. Some systems run as emulators on new hardware, etc. All types of testing are important on applications, infrastructure applications (e.g., emulators), and hardware. This includes performance testing.

It is advisable to conduct a proof-of-concept to verify the viability of the overall approach, including the types of test.

3.8.3 TOGAF Content Metamodel

Legacy systems and technology platforms are building blocks in the TOGAF Content Metamodel, just like all other applications and technology platforms. Functionality of existing legacy applications (physical application components) need to be modeled in logical application components and decomposed in information system services. This enables, on a logical level, a gap analysis with the Target Architecture. However, this needs to be done only to the level of detail/granularity that allows the architect to perform an effective gap analysis.

Architects may consider extending the TOGAF Metamodel with the SOA-specific concepts proposed in the TOGAF SOA Guide [15].

3.8.4 Service Migration and Re-Use Technique

The Carnegie Mellon Software Engineering Institute (SEI) has developed a Service Migration and Re-use Technique (SMART) which helps to make initial decisions about the feasibility of re-using legacy components as services within an SOA environment [17]. This technique can be of good value during the architecture cycle.

3.8.5 Cross-Cutting Concerns

Legacy modernization has a major impact on operation. Some new capabilities introduced in rebuilt applications may require adjustments in monitoring, network planning, and security. Service-Level Agreements (SLAs) should also be reviewed. Cross-domain applications may cause interesting conflicts among decentralized operations teams.

4 Best Practices

4.1 Overview

Service-Oriented Architecture (SOA) is an architectural style for creating business solutions to enable business agility. The goal is to continuously improve business results by SOA services that generally enable flexibility, maintainability, and cost-effectiveness more than legacy and non-SOA solutions. This section emphasizes the key foundational aspects and considerations that would effectively transform legacy solutions to SOA solutions.

4.2 Key SOA Principles

SOA defines a set of (generally agreed) key principles for services. In this section a few of them are described, including key aspects regarding legacy systems which need to be taken into account in an Legacy to SOA (L2SOA) evolution. Though TOGAF itself is not intended to be prescriptive, these key SOA principles are essential for the service enablement of legacy.

4.2.1 Well-Defined Service Contract

Service interactions must be well-defined with a widely supported standard (e.g., WSDL) describing details to assist the service requestor to invoke the service(s) required.

This principle helps to prevent the automatic generation of contracts out of not-so-well documented existing legacy APIs (more or less "code first"), which might be tempting at first.

Another aspect of this is to avoid performing direct legacy database-to-schema conversion, which introduces tight-coupling between message and database.

4.2.2 Define Services with Appropriate Granularity

Services must be designed for appropriate granularities that offer greater flexibility to service requestors without impacting the performance and security. Services granularity should make it easy for service requestors to assemble services to execute business scenarios. This is not always possible, especially for (multi-step) transaction-oriented (legacy) systems. The granularity level should be clearly described per service (e.g., which steps of functionality and invocations of other services or modules take place). For certain legacy systems this might imply first a (re)documentation project.

4.2.3 Loosely-Coupled Services

Service requesters must not have any knowledge about the technical details associated with a service implementation. Seamless integration and exposure of legacy systems into a flexible IT environment with the use of SOA abstracts the technology implementation from the use of it. As long as the implementation meets the specified Service-Level Agreement (SLA), the service requester does not need to have any knowledge about the technical part of the service implementation. Therefore, legacy systems can be migrated or replaced by other technology, without affecting the service and therefore the service requester.

This also relates to the principle of well-defined service contract, described earlier.

4.2.4 Design Services for Stateless

Services invocation must be independent of the state of other services and each service invocation has all the required information from one request to another. This is a challenge in the functionality of legacy systems, as functions can be intertwined with other functions (e.g., not well structured COBOL modules). This is acceptable as long as it occurs in the background and the principle can still be upheld for the specified service and its contract.

4.2.5 Ensure Services have Appropriate Security Enforcement Standards

Services must be designed with appropriate security policy enforcement mechanisms to ensure that only authorized requesters can successfully invoke them. When centralizing policies and identity management in SOA, at least a mapping needs to be done from individual systems security information to the centralized information to be able to implement this. It might not be possible to integrate legacy systems with a central security mechanism, so redundancy mechanisms need to be developed and maintained (including pre-call mapping of credentials).

4.2.6 Adopt SOA Ontology/Vocabulary Standard

To effectively facilitate SOA adoption and have required alignment between the business and IT communities, a common vocabulary standard must be utilized in services lifecycle management. Ontology defines the SOA concepts and semantics commonly understood by all stakeholders and enables effective communications. The Open Group SOA Ontology [16] facilitates SOA adoption.

4.3 Operational Considerations

4.3.1 Define Metrics (including Key Business Performance Indicators)

Metrics are essential for the continuous communication and justification of an L2SOA engagement. What starts with a baseline measurement (which might be put in a business case), will be measured at appropriate intervals and reported. Here a few metrics are suggested.

1. **Application Usage**: What is the actual usage of an application, before and after L2SOA? SOA can lead to a service which can be used by many more consumers than in the current state. This change can be measured if there is a baseline.

2. **Cost Reduction**: Define various cost factors; e.g., maintenance, license, energy, etc. The ratio between relevant cost factors also need to be taken in to consideration; e.g., the ratio in IT budget between the projects supporting business growth initiatives and the projects for (legacy) maintenance.

3. **Functional Re-use**: Define and measure functional re-use before and after the evolution of service-enabling business functionality. The metrics need to be measured and reported across business silos.
4. **Quality of Service**: Quality of Service (e.g., service up/downtime, message throughput) is part of Service-Level Agreement (SLA) monitoring and should include appropriate performance-related KPIs.
5. **Revenue-generated**: Creation of services can lead to additional revenue by exposing internal processes/functionality externally.
6. **Time-to-Market**: Using SOA to enable re-usable and standardized services to access legacy systems should reduce time-to-market of certain functionality.
7. **Security KPIs** (data protection-related KPIs): Especially concerning the key SOA principle: "Ensure services have appropriate security enforcement standards", this metric helps to monitor frequency and severity of security incidents after modernization. The Open Group Security Forum, a Forum of The Open Group, covers a range of security aspects including SOA [10].

4.4 Define Architecture using the TOGAF ADM

There are several distinct approaches that can be leveraged to approach the architecture definition of the transformation (evolution) of any legacy enterprise to service-oriented. The approaches in this Guide are based on the foundation of the TOGAF 9 [4] architecture framework and Architecture Development Method (ADM), providing the step-by-step approach, iterative processes, and types of building blocks needed for the evolution. Chapter 3 discusses this in detail.

To define the Architecture Building Blocks (ABBs) for the L2SOA evolution, modernization strategies are helpful. They define common strategies to use for a certain situation, and provide common building blocks for use in that context. The next section provides an overview of strategies for enabling L2SOA modernization.

4.5 Modernization Strategies

There are several strategies to enable the process of modernizing legacy applications using SOA. Enterprises need to select one or more strategies and evaluate against these strategies during the process. The key strategies are service enablement, language conversion, re-architect, and re-hosting of applications. The following sections detail each of them, describing the approach, problem solved/value-add, risks and mitigations, and architecture building blocks involved.

4.5.1 Service Enablement

Approach

This strategy enables the access of legacy applications through services hosted by integration platforms and/or service containers (comparable with NCOIC Integration Level 3 – Using Existing Interfaces [8]).

Two implementation models can be distinguished [11]:

1. **Take the Data to the Service**: Collate slowly changing information into a holding data store (a warehouse or data grid, for example), and act upon the data there. Extract, Transform, and Load (ETL) and object caching technologies can be used to make the data more readily available to the service and less dependent upon the legacy application.
2. **Take the Service to the Data**: For rapidly changing transactional data, the action must go to the physical source to manipulate it. The integration layer will use connector and adapter technology to make those accesses (for example, read and update).

These services can be categorized into presentation services (e.g., screen scraping), and task services (e.g., web service querying data).

Capabilities to consolidate multiple legacy assets are possible by creating services that invoke multiple legacy systems or data.

Problem Solved/Value-Add

- Re-use and leverage existing assets while protecting huge investments made in legacy applications.
- Improve value of core applications by using SOA methods and technologies to unlock value.
- Leverage legacy functionality in automated business processes spanning multiple systems and departments.

Risks and Mitigations

- Difficult to identify useful legacy functionality to be leveraged; mitigated by detailed business process analysis and mapping, and conducting (both static and run-time) code analysis may help to answer the question: "Which applications contain the designed functionality?".
- Poor documentation can lead to unknown functionality *versus* clear service interface.
- System transformations may be disruptive. Proper analysis is necessary to understand organizational, process, and technology impact.
- It is important to communicate the vision of service enablement to collaborate in enterprise-wide SOA program governance, which is not always easy.

Architecture Building Blocks Involved

- **Information System Architecture**: New information system services, providing legacy system functionality.
- **Technology Architecture**: Integration platform, connected to legacy systems.
- **Technology Architecture**: Service containers hosting the services.

4.5.2 Language Conversion

Approach

This approach aims at converting applications written on legacy languages into more supportable languages. The main goal of this conversion is to be able to convert to a language that provides SOA enablement by default (e.g., Java platform), so the legacy system functionality can be directly provided through services by the system itself. This only works if the conversion can be done automatically. Otherwise, a re-architecture (see next section) might be necessary in case the software structure is very complex and interwoven, or even a re-build might be necessary in case the previous strategies are not sufficient.

Problem Solved/Value-Add

- Reduce Total Cost of Ownership (TCO) of the applications – mainframe, hardware, software license fees, and learning costs.
- Reduce time-to-market with equal or better performance, availability, and scalability.
- Applications can be made adaptable to SOA implementation.

Risks and Mitigations

- Legacy functionality is not structured in modules, but has intertwined business logic which makes it impossible to provide structured autonomous services; mitigation is to re-architect (see section below).
- No supported functions in the latest version of the languages; mitigation could be to analyze transformation and convert to compatible modules providing equivalent functionality.
- Skill/training used for analyzing old platform/technology/languages might be scarce; mitigation could be to ensure access to key resources with legacy business and program skills.
- No automated conversion possible; only mitigation is to select another strategy.

Architecture Building Blocks Involved

- The logical components for the legacy system stay the same, but this might imply a new technology component for platform changes (language changes, platform might change).

- Additional information system services can be defined, which can be implemented using the SOA enablement options provided by the new language. This is comparable with the service enablement strategy as described in Section 4.5.1; the difference is that the system is able to provide the services by itself as a provider (which still can be mediated by an integration platform).

An overall important aspect is what to do with the bugs that are found: fix them or keep them. Fixing brings additional impact analysis, testing, and budget and planning discussions. Keeping them makes the unknown known and needs to be communicated.

4.5.3 Re-Architect

This approach aims at re-architecting the legacy system. It restructures the business logic towards a modularized design. This design enables the creating of autonomous services that can access the functional modules.

Two sub-strategies are identified below.

4.5.3.1 Re-Architect, but Use the Same Environment

Approach

In this strategy the mainframe source is first restructured into specific functional modules. Then these modules are opened up with adapters and integrated with a service mediation component, as described in Section 4.5.1. But, this means a change in the legacy system.

Problem Solved/Value-Add (in addition to Section 4.5.1)

- Re-use and leverage existing business functionality and intellectual property.

Risks and Mitigations (in addition to Section 4.5.1 and 4.5.2)

- Poor and old documentation especially on the system structure is a huge risk. Part of re-architecting will be spent on documenting the system as-is.
- Re-architecting might lead to programming changes (e.g., split functionalities and code). Needs additional testing, especially regression testing.

Architecture Building Blocks Involved (in addition to Section 4.5.1)

- Depending on the granularity of the ABBs in the specific architecture (e.g., one level deeper than the legacy component itself), re-architecting can lead to less or more ABBs.

4.5.3.2 Re-Architect, but Use a New Environment (e.g., Language, Platform)

Approach

This is a combination of restructuring in specific functional modules, combined with a language conversion. This is also called re-build, but in the case of L2SOA evolution re-architecture must be done first before re-building in a new language. Re-building is not the same as language conversion described in Section 4.5.2., because it is not an automated conversion due to the necessary re-architecture work.

Problem Solved/Value-Add (in addition to Section 4.5.2)

- Possible to fix old problems and introduce new functionalities.

Risks and Mitigations

- Budget discussion; need good arguments why one of the other approaches was not chosen.
- See previous sub-strategy.

Architecture Building Blocks Involved

- See previous sub-strategy.

4.5.4 Re-Hosting of Applications

Approach

This is the need to reduce the cost of the mainframe platform in cases where mainframes support the majority of business-critical applications to distributed and cloud computing models. Note that this strategy on its own does not transform the legacy system to SOA. It needs to be combined with one or more of the strategies described earlier.

Problem Solved/Value-Add

- Preserve business logic and protect existing investments in legacy applications.
- Improve their competitiveness by integrating legacy applications with new applications.
- Maintain the end user's experience on the front end of the application and yield cost savings.
- Reduce time-to-market with equal or better performance, availability, and scalability.

Risks and Mitigations

- Integration of multiple product vendors and tools; need a program to manage required integration changes as separate projects.
- Estimating capacity for re-hosting from one platform to another requires very detailed analysis.

Architecture Building Blocks Involved

- The information system building blocks will stay the same.

- The technology ABBs, especially the physical, will change to the new host type.

4.6 Pattern-Based Approach

The L2SOA enablement key architecture styles are elaborated further by leveraging Enterprise Integration Patterns (EIPs) which provide the capabilities to service enable legacy applications.

However, large enterprises having complex enterprise systems with various types of legacy applications often leverage a combination of approaches to provide the capabilities to modernize their enterprise solutions depending upon the architecture strategy that is employed.

The following patterns are mostly used (one or a combination):

Table 1: Enterprise Integration Patterns

Pattern	Description
Enterprise Integration (Service Bus)	An Enterprise Service Bus (ESB) represents an environment designed to foster sophisticated interconnectivity between services. It establishes an intermediate layer of processing that can help to overcome common problems associated with reliability, scalability, and communications disparity [9].
Queue-based Mediation	Queues enable guaranteed delivery through persistence of the data (messages) being sent. They operate in an asynchronous manner, which enables the complete decoupling between sender and receiver.
Service Enablement	Solution logic can be encapsulated by a service so that it is positioned as an enterprise resource capable of functioning beyond the boundary for which it is initially delivered. It is subject to additional design and governance considerations [9]. Important is the use of a standardized service contract, to reduce implementation coupling. This includes both providing legacy functionality as services, and adding mechanisms to legacy systems to consume services.
Screen Scraping	Screen-scraping software electronically "reads" and "puts" the information from a mainframe terminal screen. But, the service specification is totally dependent on the screen transaction definition.
Re-design	Instead of keeping the application as-is and using one of the patterns above, this pattern actually re-designs the application which provides opportunities to service enable the application by design (using re-architecture and/or language conversion strategies, as described in Section 4.5).

Based on these patterns and the statistical information gathered by research, the approaches can be categorized into matrices which provide indicators of the benefits and pitfalls of each approach. This provides a starting point to add more detailed multi-dimensional models that provide a more holistic view of the various approaches.

Based upon a set of L2SOA projects, it is possible to qualitatively analyze cost and complexity ranges per pattern.

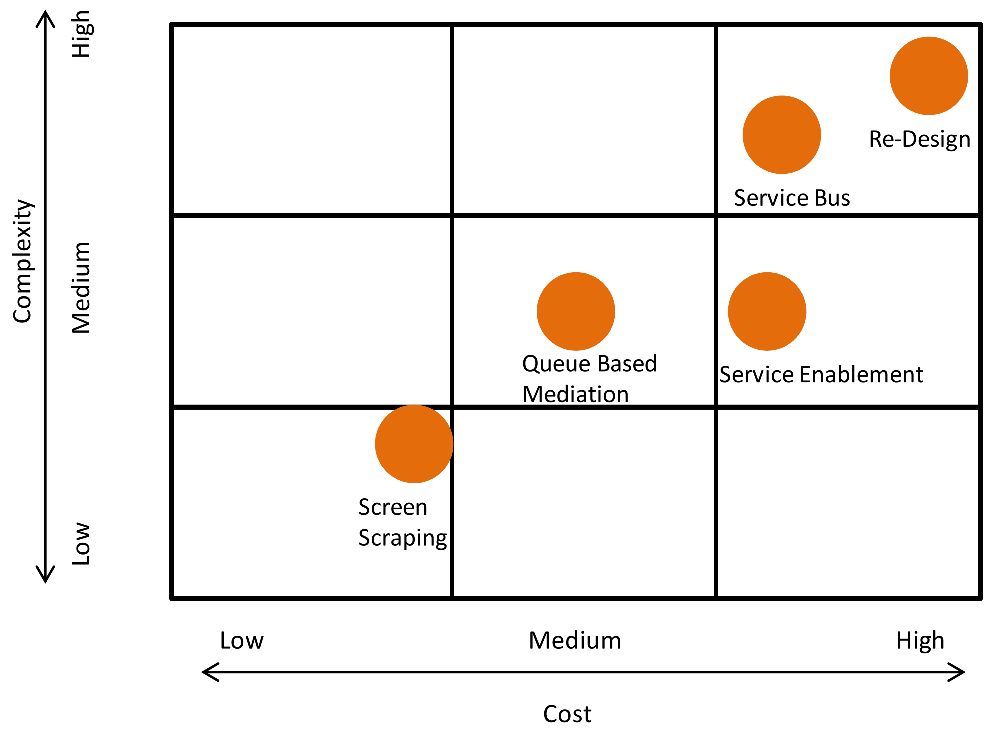

Figure 2: Integration Patterns Cost and Complexity

One thing that is immediately visible is the relation to return-on-investment. The directly SOA-related patterns have a high cost and medium-to-high complexity. But these patterns result in a solution/enterprise-level infrastructure and transformation that is beneficial to many more (legacy) systems than if we talk about screen scraping, typically delivered as a web service. A strong conclusion can be made that SOA enablement of legacy systems needs to be seen in the context of the enterprise as a whole and not as individual systems. In this manner, the high (infrastructure) costs can be spread across many service-enabled applications – both legacy and new. This can only be accomplished if the process is viewed from an Enterprise Architecture perspective.

4.7 Relationship with OSIMM

The OSIMM model [5] provides a holistic approach to the maturity of applications in an enterprise. Leveraging the model and the patterns that are described above can provide a view of the silo'ed legacy applications' maturity level, providing the organization with the information to progress by making an evolution from a legacy system to a service-oriented approach. For example, providing a queue-based mediation between an SOA application and a mainframe that currently resides in a silo that is targeted to be integrated. This could provide the organization with the capability to strategize enablement of a portfolio of legacy applications leveraging the

OSIMM. Chosen architectural styles and patterns can be mapped on the matrix fields between desired maturity level and dimensions (application, architecture, and information).

4.8 Organization and Process

Legacy evolution has impact on current organization and process:

- L2SOA is a transformation program and has to be approached and managed as such.
- Current IT operations and management teams undergo impact. A change program is needed to guide them.
- New technologies will enter the landscape and need to be learned.
- New roles will be added to create/support the modernization. This can be an opportunity for employees to gain new knowledge, but also a threat that they might not keep up or are not given the opportunity to re-learn (e.g., other people are hired).
- Current change request procedures need to be extended.
- SOA brings the concept of services implemented across business boundaries, both from a functional and technical perspective. While the legacy systems might currently support clearly defined dedicated business processes and even departments, this will certainly change. Service owners will need to be able to operate in that context.
- It is advisable to create a competence center where the (at least technical) knowledge of modernizing L2SOA is consolidated.
- How can organizations see the value and how do they "sell" it to other departments in the organization? Definition and measurement of return-on-investment of an L2SOA engagement is crucial. See Section 4.3.1 for metrics and Key Performance Indicators (KPIs) which can be used for this.

4.9 Governance

4.9.1 Introduction

In general, governance means establishing and enforcing how people and solutions work together to achieve organizational objectives. This focus on putting controls in place distinguishes governance from day-to-day management activities [6].

SOA governance should extend the organization's existing IT and Enterprise Architecture governance models to take into account the new SOA assets and SOA policies [6]. SOA, when done correctly, leads to services which connect and also transcend traditional business boundaries. Functionality and data in legacy systems might therefore be used by new consumers under new Service-Level Agreements (SLAs). SOA governance is essential for organizations that make pervasive use of SOA in their business and IT systems regardless of legacy system modernization efforts.

4.9.2 The Open Group SOA Governance Framework and Relation to Legacy Systems

The Open Group SOA Governance Framework [6] is based on the definitions of governance and SOA governance as stated in the previous section. Its goal is to enable organizations to define and deploy their own focused and customized SOA governance model. As a framework, it consists of an SOA Governance Reference Model (SGRM) which is utilized as a starting point, and an SOA Governance Vitality Method (SGVM) which aims at continuous improvement of the governance process.

The SGRM defines a set of principles, governing processes, artifacts, roles and responsibilities, and governance technologies. Legacy systems are explicitly taken into account in the SGRM, where there is a need to distinguish legacy from any other type of system.

Table 2 shows an overview of the defined touch points of the SOA Governance Framework with legacy systems.

Table 2: SOA Governance Reference Model Touch Points

Reference Model Part	Where	What
Process	Service Lifecycle Governance/ Service Realization Planning	Explicitly analyze whether the service should use existing legacy functionality.
Process	Service Lifecycle Governance/ Service Modeling	When creating the service implementation model, identify appropriate legacy integration patterns, if necessary. (*)
Checkpoint	Service Lifecycle Governance/ Approve Service Model	Ensure conformance to legacy modernization guidelines. (**)
Checkpoint	Service Lifecycle Governance/ Approve Service Implementation	Ensure conformance to legacy modernization guidelines. (**)
Artifact	Service Description	Service implementation design model needs to include legacy re-use patterns used and which legacy functionality is re-used. Also, in the general description of the implemented service, the used legacy functionality needs to be described.

(*) See also the patterns described in Section 4.6.

(**) This needs to be explicitly taken into account as an architectural deliverable.

4.9.3 Additional Considerations and Challenges, with Recommendations

Table 3: Additional Governance Considerations

Consideration/Challenge	Recommendation
Legacy systems are explicitly sized and not ready for an unknown number of users, as a result of opening up as a service.	Strict SLAs with maximum named and concurrent users defined. Explicitly test whether these numbers can be supported, because several "older" technologies have strictly defined boundaries in this.
Organization's sourcing policies to detail the architecture for the evolution from legacy results in changes to governance structures at portfolio, program, and third-party contractor levels.	Review current governance structure and extend it to cover the people (stakeholders, employees, etc.), technologies (legacy, SOA, etc.), financial implications, and processes impacted by the evolution [6][7].
The stability of the security in the legacy system can be impacted by exposing it as a service.	Security design and solid implementation covering the legacy system and the services architecture together is essential to make sure there are no "rogue" uses of services causing unexpected service usage.

4.10 Skills/Education/Professional Development

Employees who are used to job stability and somewhat resistant to adapt new technology often feel stress about their ability to transform to a new skill set required for evolved legacy application technology. In general, some people are uncomfortable with change and tend to resist change. Management should consider the following in planning for legacy applications transformations to SOA:

- Carefully assess the SOA implementation styles that best suit the organization while keeping the current expertise of employees. Plan to proceed gradually to adapt simple implementation models rather than those that are complex and require advanced skills.
- Identify the ideal path of least resistance that provides maximum return with minimum risk to SOA program adaptation.
- Organizations should account for the complexity and learning curve required in adopting SOA technologies and standards in their legacy applications transformation.
- Provide necessary training to progressively evolve employee skills required to effectively perform new roles.
- An effective governance process should be put in place to ensure that training and education is an ongoing process of skills development.
- Identify, involve, and cultivate agents of change early in the organization of a competency center to lead additional educational efforts. Would-be detractors may also be embraced at this stage to turn negativity of change into positive behavior.

4.11 Security

4.11.1 SOA and Security

SOA by definition is a loosely-coupled architecture, where applications are composed by connecting exposed service endpoints through a set of SOA enabling components. However, this open architecture along with its benefits also presents a complex set of requirements to secure information. Organizations that started with SOA originally used web services for simple point-to-point integration with basic security requirements, but are moving towards a more matured SOA model. To do this right they will need a solid and robust security model. Any security model should build on an organization's existing security infrastructure and hence it is not possible to come up with a one-size-fits-all model. This implies that the existing security infrastructure should be well documented, including the exact way in which security in legacy systems is implemented and integrated with its surroundings (if any).

Within SOA the general way of approaching security is to move security and policy checks out of the actual business logic into a layer above it, making it possible to integrate with a centralized policy and identity management solution including Single Sign On (SSO). In most cases this is not done or not possible on the data layer with the data layer platforms, as data is considered the core asset in the organization and therefore will have its own security layer. Ideally these are aligned with the centralized solutions mentioned earlier.

The Open Group Security for the Cloud and SOA project published a White Paper: Security Principles for Cloud and SOA [12]. It describes two sets of principles:

1. A set of 20 generic security principles common to all designs aiming to assure a secure IT architecture, ranging from "security by design" to "data protection lifecycle".
2. A specific set of principles regarding assuring security in architecting the cloud and the SOA environment, mainly focused on policies, data protection and privacy, and cloud-specific principles.

Those principles are also relevant for L2SOA engagements.

4.11.2 Legacy and Security

One of the first concerns with legacy systems is the risk of outdated technology and software, making it extremely vulnerable when the system is exposed to more parties and technologies than currently required. Assessing the state of the software is important.

Because of the direct link from system to terminal, encryption of data was not directly important. Encryption of transport and/or information itself needs to be considered when opening up the system with services.

Some legacy systems have already reached a maximum amount of named users in the years that they are used. This means that the people that use the system might have an ingenious manual administration of user names and passwords. This is a serious risk for integrating the systems with services using centralized security components.

Another concern is the way a session is defined in the legacy system. Introducing SOA can also introduce SSO. Depending on the implementation and platform chosen, a generic session is defined with session durations for an account before it needs to authenticate again. If it is not possible to send an "invalidate session" to a legacy system, other measures need to be taken to be sure that the session cannot be taken over by an account that is de-activated centrally.

US Homeland Security: Build Security In documents 623 and 624 [13] and [14] describe detailed guidance in assessing the security state of legacy systems, which is also useful in L2SOA engagements.

A Case Study: Automotive Company

A.1 Introduction

This case study provides a historical view of a journey that an enterprise has taken to evolve legacy applications to SOA. The objectives of the case study are to:

- Provide a sample of how enterprises are evolving their legacy application to SOA
- Identify gaps and learn from their journey to establish guidelines and best practices
- Describe the viewpoint on the current state for the evolution
- Provide recommendations to improve the journey by adapting the approach and best practices outlined above

A.2 Case Description

For such an initiative we cannot talk about a "project". In fact, it is a transformation program.

The Transformation Program

This program was divided into the following phases:

- **Assess current state**, including an application assessment (including what can be phased out, etc.): An assessment was done on around 50 legacy supply chain applications. This determined that: 30 applications could be replaced by common systems/Commercial Off-the-Shelf (COTS) systems; 15 applications could be retained; four (4) applications could be retired; and two (2) applications could be retired because other systems could take over the functionality. Retiring can show as a quick-win with an immediate cost saving. This was important input for the future state architecture.
- **Define the future state architecture**: The future state architecture divides the IT landscape into a presentation layer (able to web-enable mainframe screens, replace thick clients, introducing a portal), process layer (able to orchestrate processes), integration layer (Enterprise Service Bus (ESB) functionality, to replace the point-to-point coupling and to enable provisioning of web services), and the application layer (existing and new application functionality). This future state architecture enables automotive supply chain process support, in a way that is not possible in the current state.
- **Transformation planning**: This transformation is not only about replacing applications, but also about introducing a new future architectural landscape design. Careful planning is necessary to maintain control and keep the IT support ongoing for the business while

transforming the IT landscape. This also includes more detailed business case planning to support the right priorities.

- **Execution**: Execution follows a tight program and project management approach, with an architectural framework and processes in place.

Challenges to Evolve Legacy Applications to SOA

- Providing realistic return-on-investment data
- Lack of documentation
- Poor architectural design
- Lack of resources able to work on the outline
- Poor programming structure (lack of structured programming)
- Hardware where application is used (e.g., mainframe)

Business Concerns

High update costs:

- Applications are tightly-coupled (integration is "hard-coded").
- Old technology requires constant upgrading.
- Application source is mainly Programming Language 1 (PL/I), originating from the 1960s.
- The number of experts on this language is declining.
- Costs for maintaining the code are escalating.

Functional concerns:

- Applications are custom to a specific local country situation, while there is a drive to globally standardize business/manufacturing processes.
- Data is not easily shared.
- Data is not stored in a database but in (SAM) files.
- Poor usability.

Poor Business Process Management (BPM) support:

- Applications are not optimized to support best-in-class business processes.

About the Legacy to SOA Initiative

The ultimate vision of the Automotive Company is to reach the following situation:

- An adaptive and agile environment which can absorb changes.
- COTS and/or globally selected common systems deployed.
- Local customization minimized, only where local business differentiates significantly.
- Applications are driven by the processes, and not the other way around.
- Open architecture, which allows for re-use and timely sharing of data.
- Lower costs and reduced amount of maintenance.

Architectural Overview

The following diagrams show a high-level overview of the evolution from legacy systems point-to-point integrated, towards SOA.

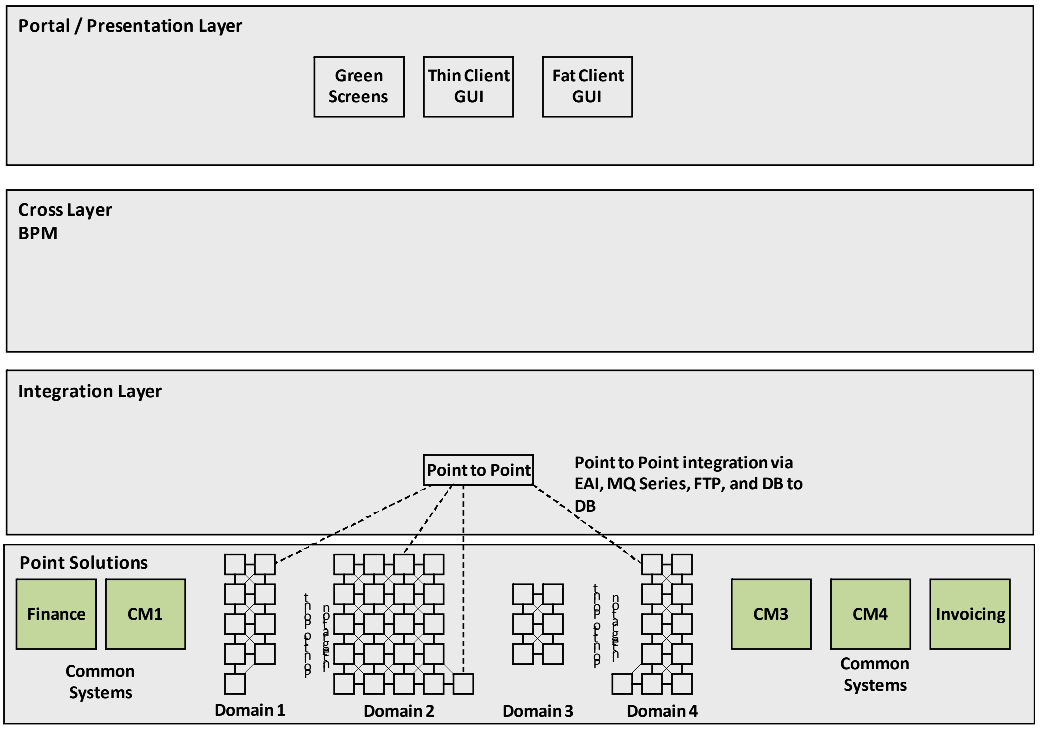

Figure 3: Case Study A – Supply Chain As-Is (Legacy)

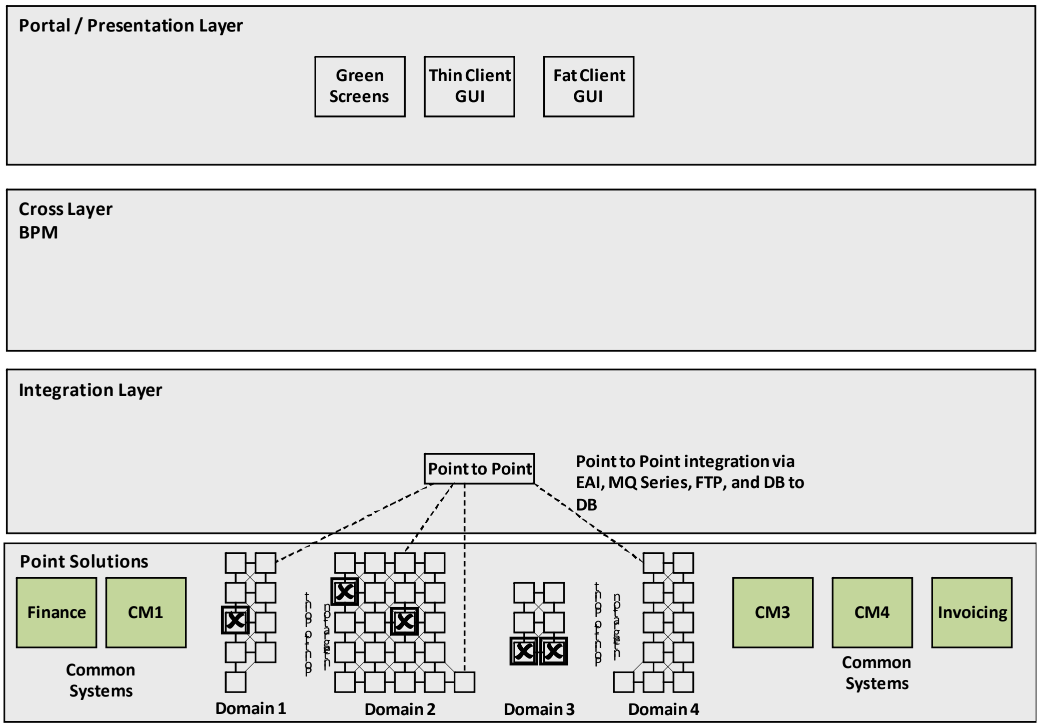

Figure 4: Case Study A – Retire Some Systems

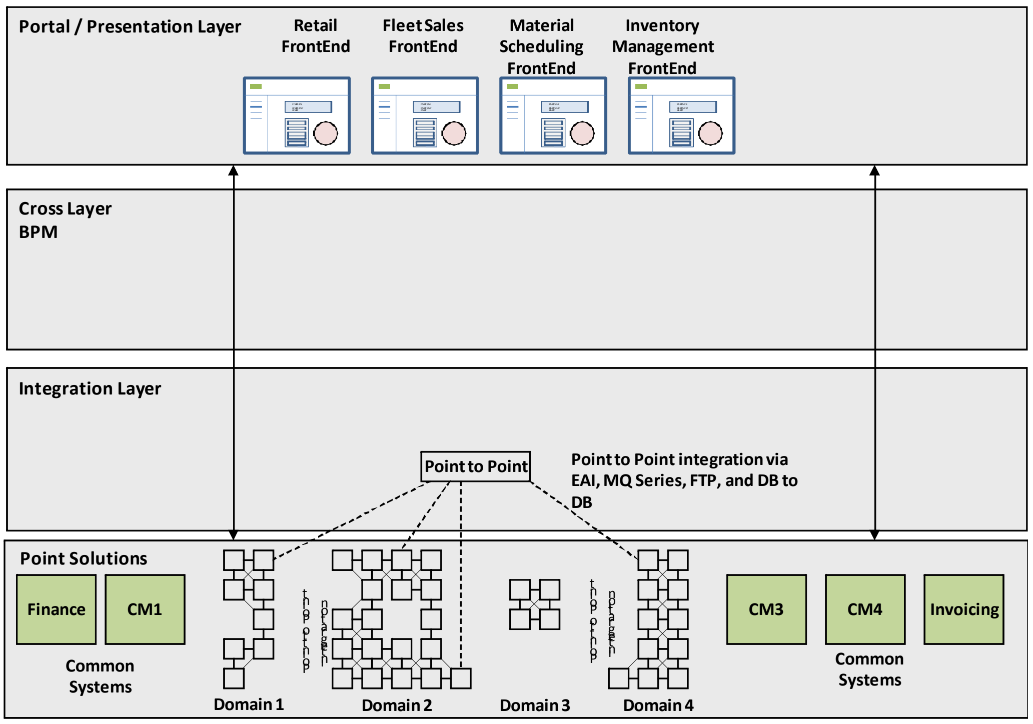

Figure 5: Case Study A – Portal Rollout

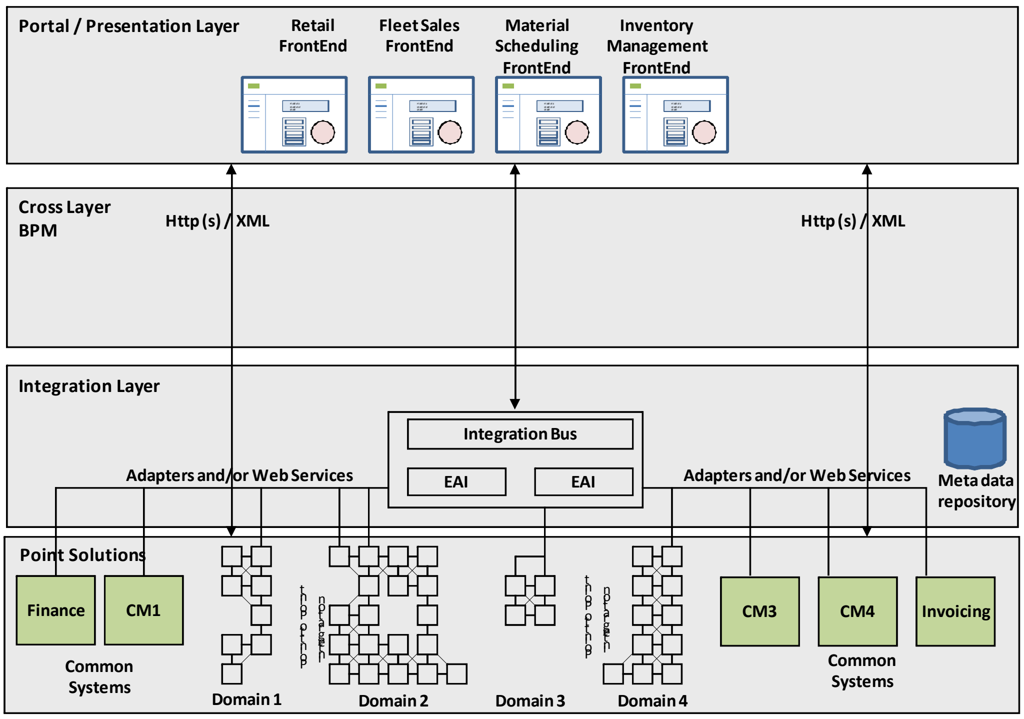

Figure 6: Case Study A – Integration Focus

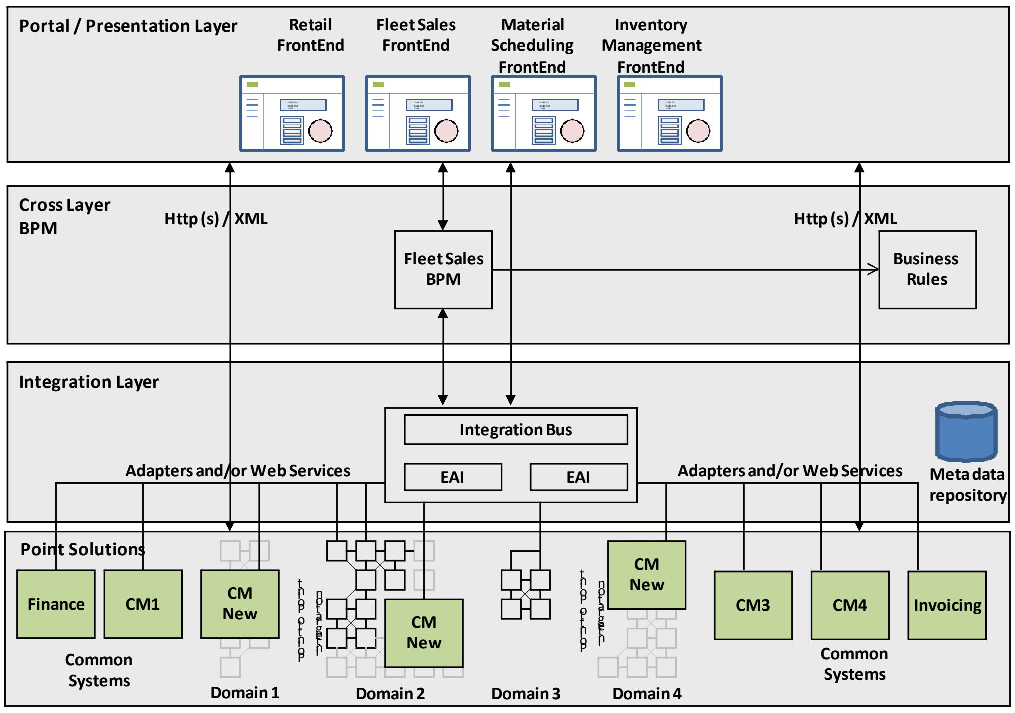

Figure 7: Case Study A – Business Process Focus

A.3 Lessons Learned

- L2SOA is a transformation program and has to be approached and managed as such.
- Quick-wins and cost reduction can help gain support.
- An application (portfolio) assessment should always be part of an L2SOA project/program.
- Transformation strategies and technology choices need to be made during transformation planning using a holistic approach to consideration of options.
- Use of a reference architecture.

A.4 Recommendations to Improve the Journey, based upon this Guide

- Governance strategy and processes were not taken into account, but need to be.
- There was a lack of resources which could work on the future architecture outline and transformation program, which poses a serious acceptance risk. This should be noted as a Critical Success Factor (CSF), and regarded as a showstopper.
- While there was a recognized challenge on providing realistic return-on-investment data, there were no base-lined metrics available. Those are necessary for the return-on-investment discussion and for initiating (approved) further transformation projects.
- The SOA focus in this case study seemed to focus on technical enablement of applications as Solution Building Blocks (SBB), and not on business capabilities and processes using a top-down approach. This is not necessarily a wrong approach, but a top-down approach focusing on business process optimization to support business drivers typically provides greater motivation for change. Many organizations use a combination of both approaches.
- These large transformations should be supported by a change program which manages changes towards operations, departments, and people.

B Case Study: Large Telecommunications Company

B.1 Introduction

This case study provides a historical view of a journey that an enterprise has taken to evolve legacy applications to SOA. The objectives of the case study are to:

- Provide a sample of how enterprises are evolving their legacy application to SOA
- Identify gaps and learn from their journey to establish guidelines and best practices
- Describe the viewpoint on the current state for the evolution
- Provide recommendations to improve the journey by adapting the approach and best practices outlined above

B.2 Case Description

About the L2SOA Initiative

This initiative was started with a pilot in 2008 and the first project in the second half of 2009. The objectives of this initiative comprise long-term improvements of the application landscape, particularly better agility, improved data quality, improved end-user experience (e.g., SSO), and sustainable cost reduction. It has selected Service-Oriented Architecture (SOA) as the underlying architecture framework to build upon. The telecommunications organization that is the subject of this case study selected a particular vendor's products as their strategic platform, including operations environment, database platform, and integration platform. In parallel to the decision on the platform product, mandatory architecture guidelines were also documented.

Note that this was a discrete, application-focused project with the intention of incrementally adopting some fundamental capabilities of SOA for rapid business benefits.

Business Concerns

High operating costs:

- The legacy environment consists of more than 40 applications. Most of the applications, including the application architecture and underlying technology infrastructures, are out-of-date.
- The same functionality may have to be implemented in more than one application, creating redundancy.

- Data is shared between applications mostly through replication, creating redundancy and the risk of non-actual data.
- Data is inconsistent since one data entity can be modified in more than one application.
- Business logic, including functions and rules, is "hard-coded" into applications and difficult to maintain and re-use.
- All integrations are point-to-point, creating a web of direct dependencies.
- Those redundancies and dependencies in the landscape make changes difficult, risky, and costly.

Slow time-to-market:

- As the result of mobile technology evolution, the mobile network must adjust to the new technology generation; for example, the next mobile generation Long-Term Evolution (LTE). This requires frequent changes in the existing applications and new development.
- Because of the IT constraints mentioned above, changes and new development became increasingly costly to the business.

Poor Business Process Management (BPM) support:

- Business processes are mostly not automated as such. Changes in process would lead to changes in code.

Poor user experience:

- There is no SSO support. The user has to log on to different applications using different credentials, in order to perform one task.

Project Description

The SOA initiative was managed as a program consisting of many projects. Examples of two of the core projects are:

- **Project I: SOA Framework**: The major goal of this project is the delivery of preparatory work that is necessary before business-driven IT projects can start. The preparatory work comprises SOA infrastructure (web services framework, registry, security, keys, development environment, and monitoring). It is crucial that the results of this project can be used by the organization and external vendors to deliver results in the future. Furthermore, this project must take into consideration that the current IT landscape of the organization is heterogeneous. Current applications are building upon a number of different types of technologies. This fact implies that this project must deliver results that are equally useful for all those technologies. For example, this project must enable third-party developers to deliver web services that can be consumed by applications that are delivered by other vendors.
- **Project II: Address Management**: The proposed project delivers a new database for storing geometry data, subsequently referred to as a GEO database. This new GEO database will be the central master for storing objects related to geographic locations

including addresses. The GEO database will be "spatially enabled" which means that it supports spatial data types and spatial functions as specified by the Open Geospatial Consortium, Inc. (OGC) and ISO. The main advantage of this new technology is the support of complex and time-consuming spatial operations. Comparable technologies are currently not in the field and are unsupported by the current database technology used. Additionally, the proposed project delivers a set of re-usable SOA services to access the objects which are stored in the GEO database or for the validation of addresses.

Roadmap to SOA: Address Management

This section illustrates the high-level migration plan, as well as the Transition Architectures communicated with relevant stakeholders.

The simplified Baseline Architecture is shown below. The address management system is built on a legacy database. It consists of nine distributed, redundant databases. The changes that occurred in one database are replicated through a central hub database. The client applications access the database through proprietary APIs.

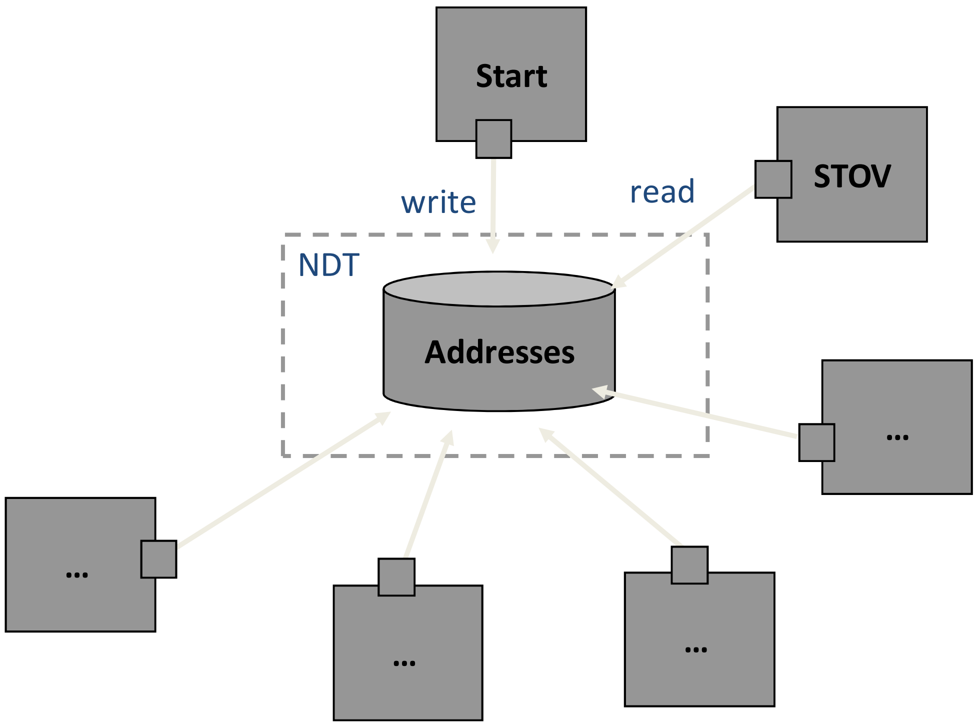

Figure 8: Baseline Architecture

The first work package aims to migrate data from the legacy database to the targeted new database system (see Figure 9).

Two important activities during the data migration are:

- **Implementing New Information Model**: The data model is redesigned according to the (new developed) information model NSS SID. This model is defined based on TM Forum

Shared Information and Data (SID), which provides the telecommunications industry with a common vocabulary and set of information and data definitions and relationships used in the definition of NGOSS architectures. In addition, the company has identified the need to further extend the SID model, so-called "SID refinement levels". The extensions include additional attributes and references to the existing data entities, as well as new data entities.

- **Data Validation and Data Cleansing:** The migrated data will be validated using predefined rules. This ensures the data quality for the new start.

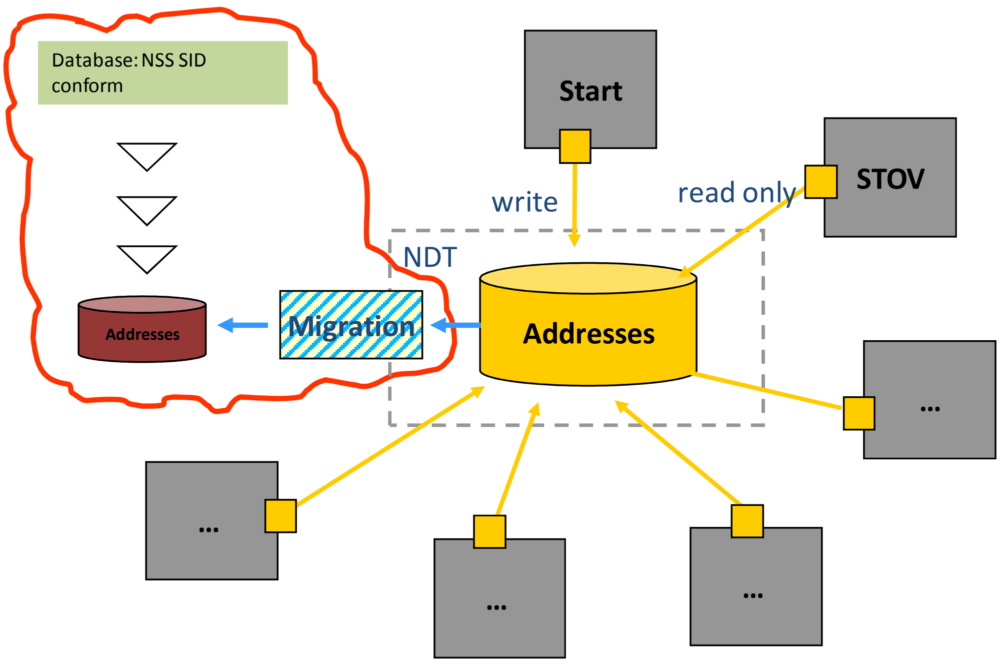

Figure 9: Step 1: Data Migration

In the second step, the new database becomes the master of address data. This is achieved by the following activities:

- All write access to the legacy address database is disabled; e.g., application "Start" (see Figure 10). This is done by database-level security.
- The application Start is "re-interfaced" with the new capability to make changes through web service calls (see Figure 11).
- Address Data Service ("BasicAddressService") provides CRUD operations that access the new data sources.
- The change data is synchronized to the legacy database through database-level replication.

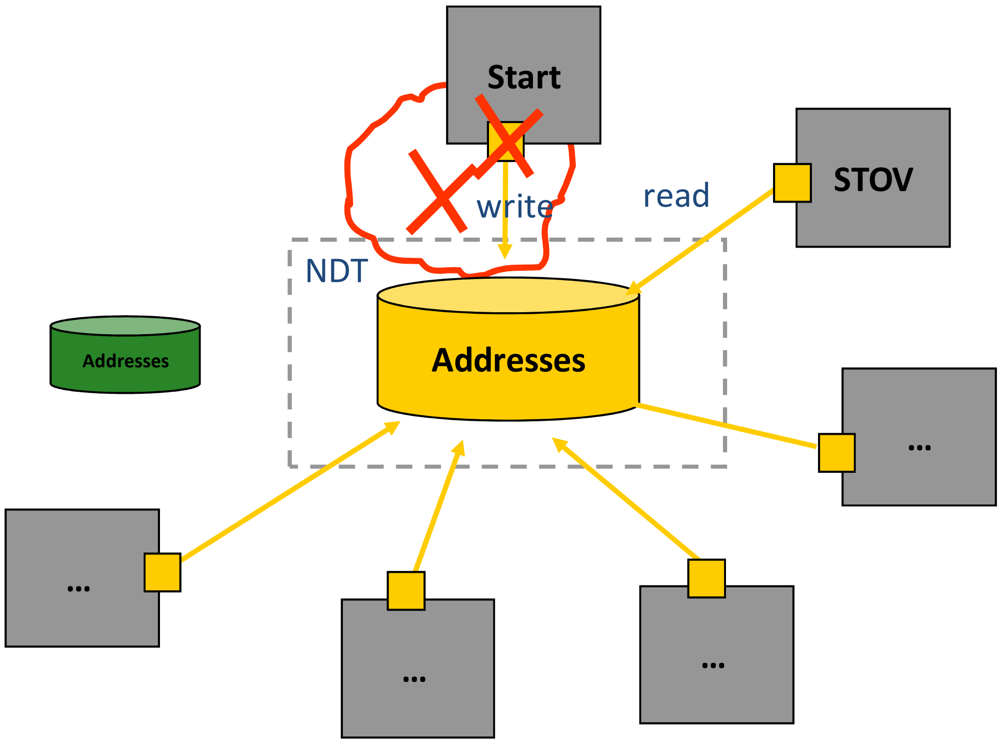

Figure 10: Step 2a: Shutdown of Write Access

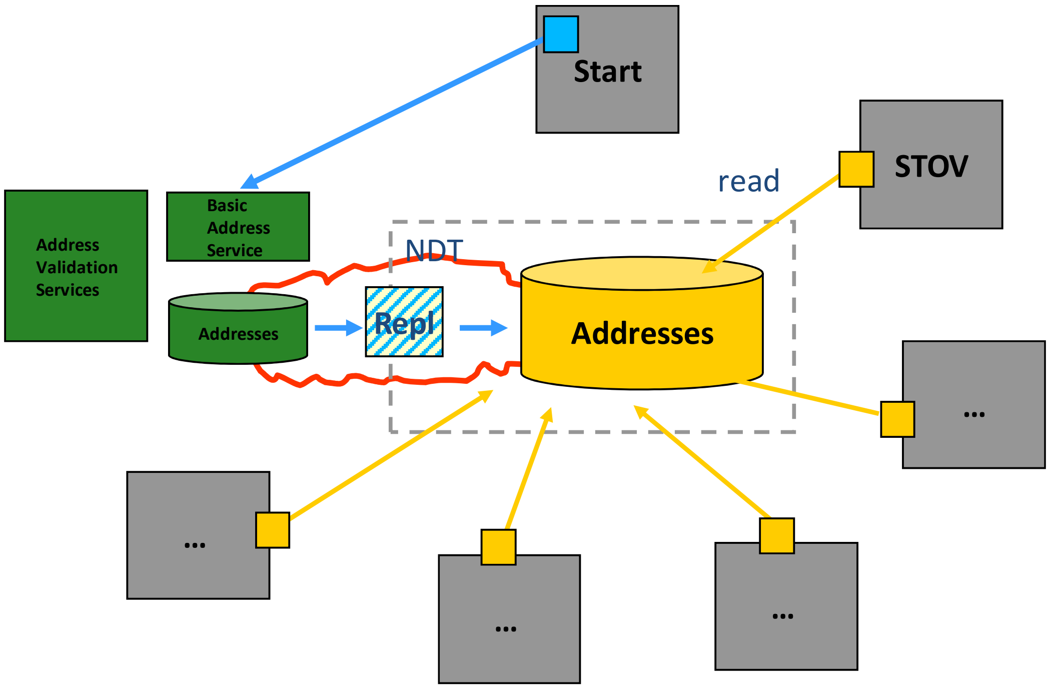

Figure 11: Step 2b: Data Replication

In the next step, all other applications are migrated one by one to access the new data source. Each migration contains the following activities:

- First, the access to the legacy data resource is shut down (see Figure 12).
- The application is then "re-interfaced" to retrieve data by consuming BasicAddressService (see Figure 13).

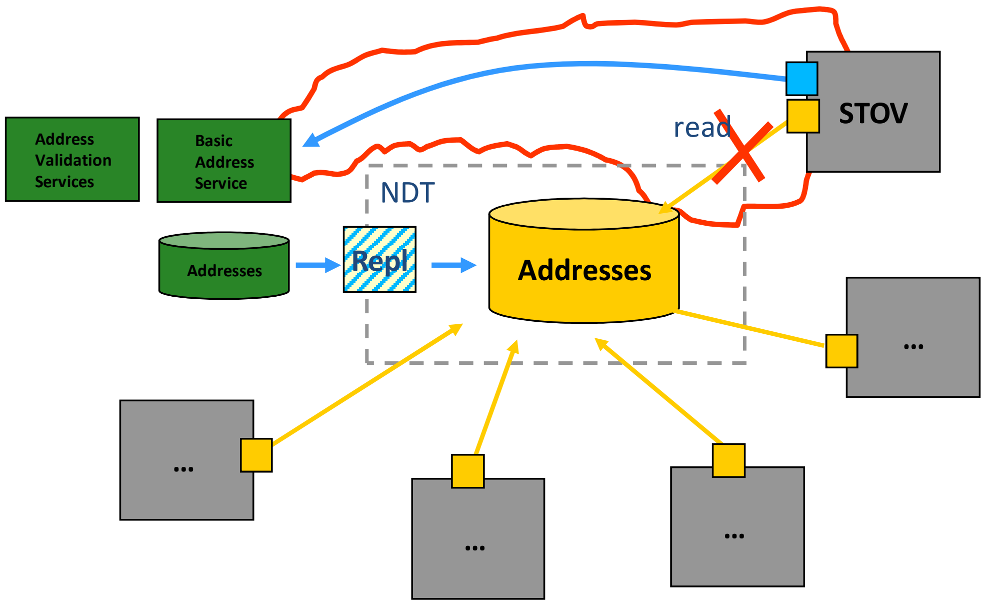

Figure 12: Step 3a: Migration of One Read-Only Application (STOV)

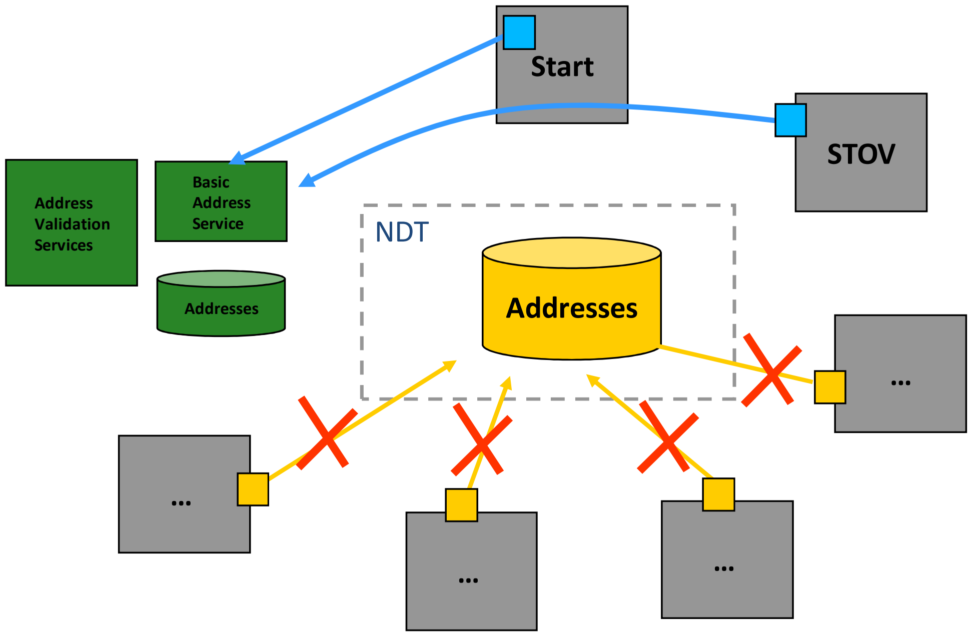

Figure 13: Step 3b. Migration of the Remaining Applications

Finally, after all applications are switched to new data source, the legacy database will be retired.

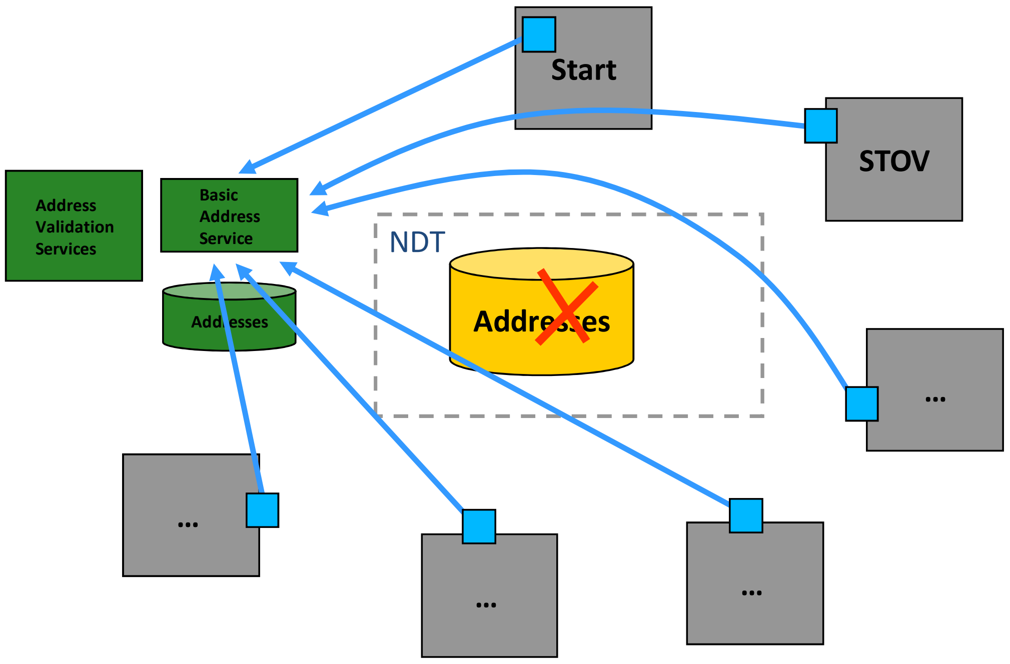

Figure 14: Retirement of Legacy Data Source

The following sections describe how the applications are “re-interfaced”.

B.3 "Re-Interface" on Consumer Legacy Applications: General Approach

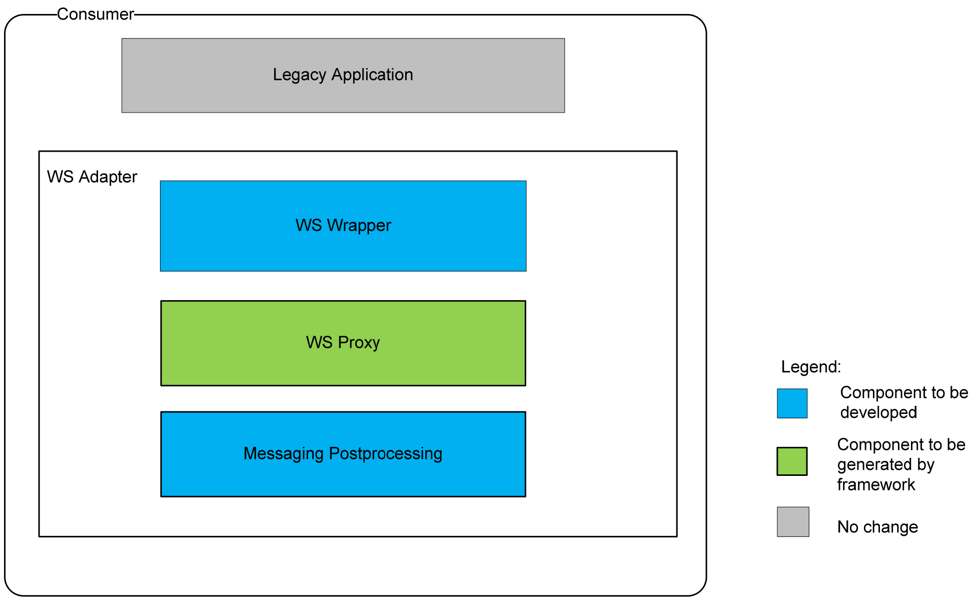

Figure 15: Consumer Layered Architecture

Figure 15 shows high-level layers of a web service consumer:

- The legacy application could be START or STOV that doesn't have built-in web service capability.
- The web service adapter is a component that exposes remote web service provider functionalities to legacy applications as if the provider is local to the legacy application. The main activities of an adapter involve protocol transformation and data transformation. It can be further decomposed into three sub-systems: WS wrapper, WS proxy, and message post-processing.

WS wrapper provides the following functionalities:

- **Data Transformation**: The input parameters will be transformed into NSS SID entity object instances. The result will be transformed back to the required format. In most cases, this part should be implemented programmatically. If the transformation logic is not stable, both proprietary data type and SID data type must be first transformed into XML documents. The transformation logic would be implemented through XSLT programming.
- **Endpoint Look-up**: The provider endpoint URL can be looked up in the UDDI Registry through direct routing.
- Invocation of operations provided by web service proxy.

WS proxy:

- WS proxy implements the WS*-Standard. It is sometimes also referred to as the Service Endpoint Interface (SEI). Basically, it contains the standard web sub-systems. In case of contract first approach, WS proxy will be generated by the WSDL compiler.

Message post-processing/SOAP interceptor: The post-processing refers to interceptors that can process SOAP before it is sent to the provider. The following standard interceptors are implemented:

- **Security**: Enables Kerberos-based SSO; enables web service security using SAML token.
- **Logging**: It logs the messages in the central log store for debugging purposes.
- **Message ID**: It adds a unique message ID to the SOAP header.

Figure 16 depicts possible interactions between all consumer site parties.

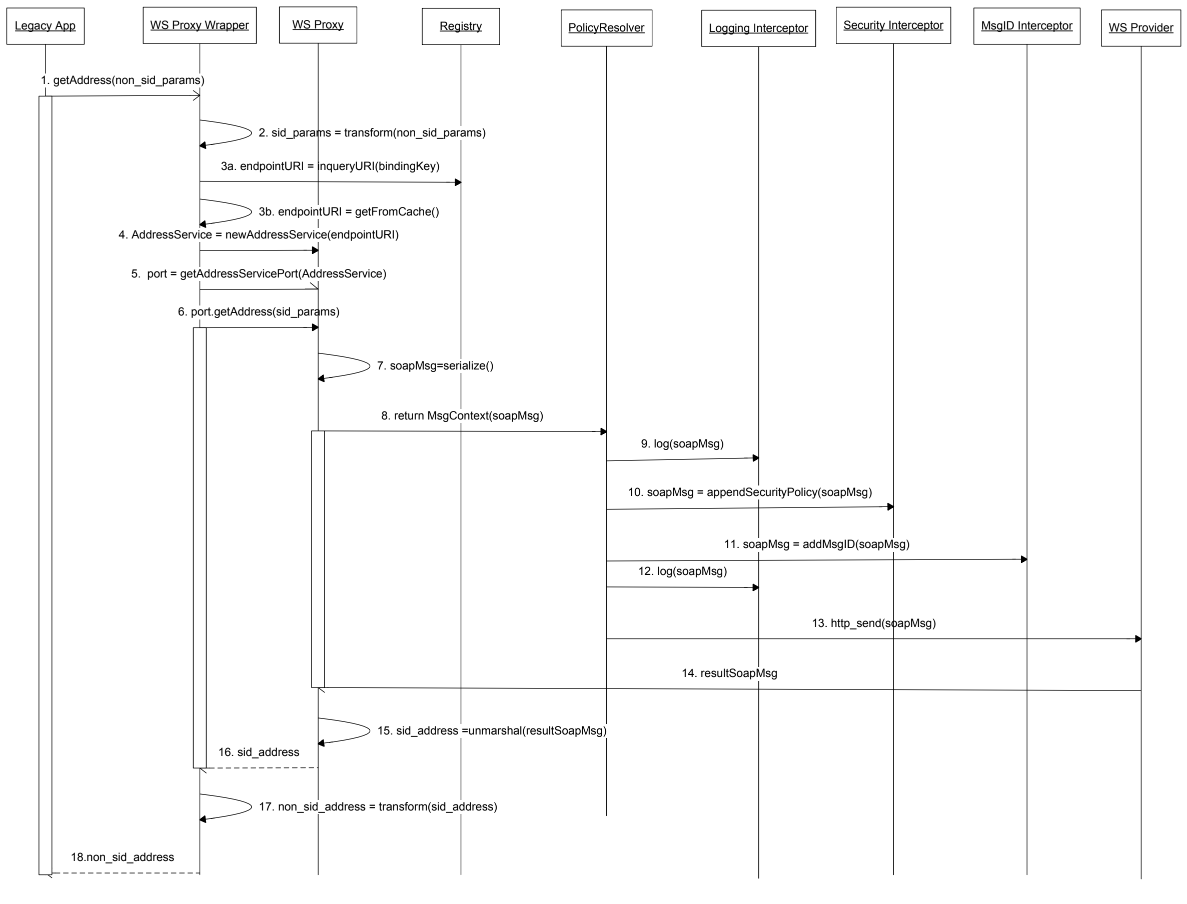
Legacy App
WS Proxy Wrapper
WS Proxy
Registry
PolicyResolver
Logging Interceptor
Security Interceptor
MsgID Interceptor
WS Provider
1. getAddress(non_sid_params)
2. sid_params = transform(non_sid_params)
3a. endpointURI = inqueryURI(bindingKey)
3b. endpointURI = getFromCache()
4. AddressService = newAddressService(endpointURI)
5. port = getAddressServicePort(AddressService)
6. port.getAddress(sid_params)
7. soapMsg=serialize()
8. return MsgContext(soapMsg)
9. log(soapMsg)
10. soapMsg = appendSecurityPolicy(soapMsg)
11. soapMsg = addMsgID(soapMsg)
12. log(soapMsg)
13. http_send(soapMsg)
14. resultSoapMsg
15. sid_address =unmarshal(resultSoapMsg)
16. sid_address
17. non_sid_address = transform(sid_address)
18.non_sid_address

Figure 16: Client-Side Interactions

B.4 "Re-Interface" on Consumer Legacy Applications: General Approach: Technology Platform-Specific Scenarios

This section demonstrates how the web service framework fits in the required (legacy) technology stacks (a non-Java and a Java stack).

Non-Java Stack

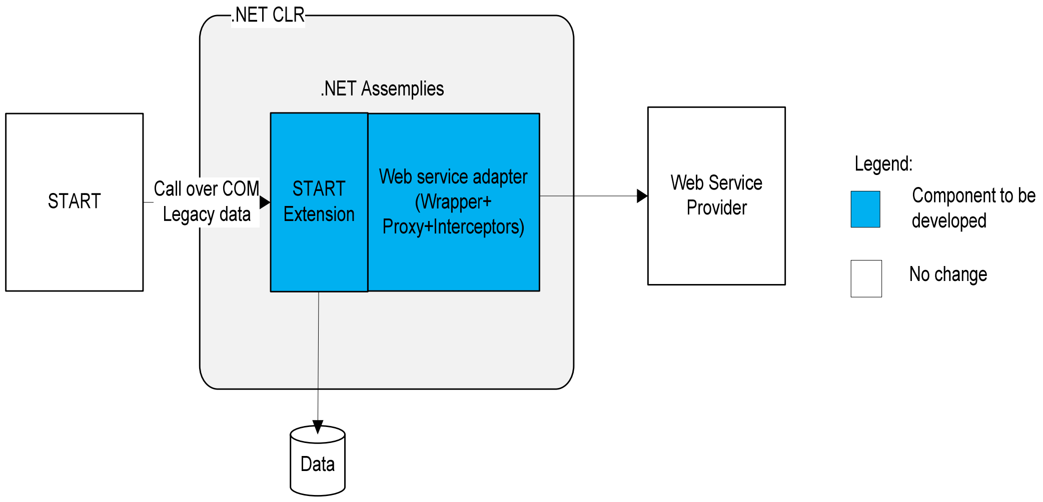

Figure 17: START as Web Service Consumer

Figure 17 shows an integration scenario with application START. The web service adapter will be implemented as .NET components/assemblies, since the START is built with a language version that does not support WS*. START will utilize a COM object that bridges the procedure call from START to web service adapter. In addition, START will provide an additional layer as controller that manages distributed transactions.

Java Stack

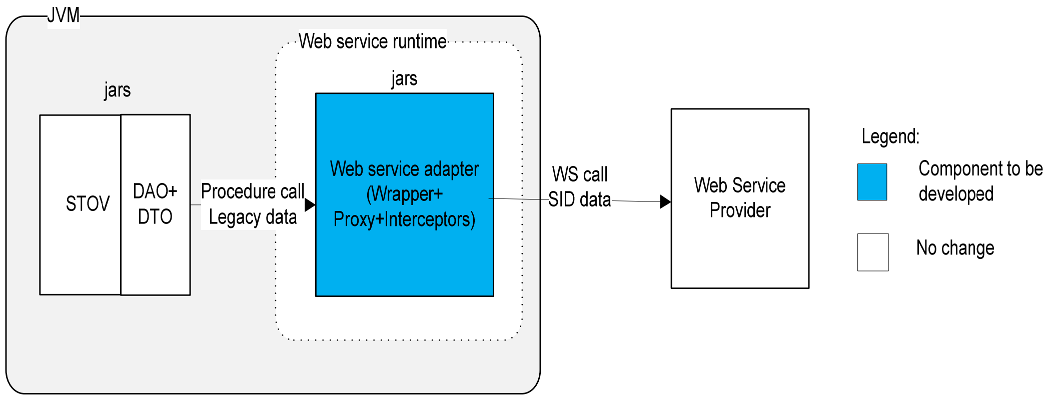

Figure 18: STOV as Web Service Consumer

STOV is a Java swing application that currently manages direct read access to the database though Data Transfer Objects (DTO) and Data Access Objects (DAO). Instead of implemented database access methods, STOV will call the web service adapter directly, and pass the DTO (in legacy format) as a parameter (see Figure 18).

Technical Platform

- Various Operating Systems
- Databases from different vendors
- Web Service Container
- Enterprise Service Bus
- Service Repository
- Service Registry
- Business Process Management
- Single Sign-On
- SOA Monitoring
- Software Configuration Management

Challenges (to date)

- **Requirement Management**: Because of paradigm change, the requirements are not clearly stated.
- **IT Operations**: These SOA projects will add (a lot of) new parts (both software and hardware) in the existing infrastructure. This requires tight collaboration with the existing operations team. The acceptance of the program within the operations team is low, so a change program is needed.
- **Skills**: New technology change requires new skills. Management is hesitating with the training, because they assume all development will be taken over off-shore anyway.
- **Technology**: Interoperability has to be ensured between different technology stacks. Also, performance may suffer because of network hops and XML overhead.

B.5 Lessons Learned

- An SOA framework (or platform) needs to be delivered and proofed before various IT transformation projects (scoped on an application) start.
- Tight collaboration between the existing operations team and the team that designs and installs the new SOA-based software and (additional) hardware is essential.

- When migrating data as part of the program (in SOA-related projects mostly consolidation towards data services), validation and cleaning is of utmost importance.
- Use of standardized (industry) information models is advisable.

B.6 Recommendations to Improve the Journey, based upon this Guide

- Instead of only focusing on SOA frameworks for specific solutions, also ensure that enterprise SOA principles are accounted for.
- The major business case was to decrease time-to-market. Define metrics, baseline them, and then measure after implementation. Some potential metrics are named in the case; e.g., frequency of changes in existing applications, frequency of new developments.
- Implementing SOA components like proxy, registry, policy manager, interceptors, etc. can be complex. It is important to execute a proof-of-concept addressing the riskiest areas of the solution architecture before starting a project.
- Managing change using a formal approach running parallel to SOA project execution is crucial in gaining acceptance of new technologies and processes used in projects.
- Various testing activities need to be planned as part of the project planning (and as early as possible), especially interoperability and performance testing.

Glossary

ABB	Architecture Building Block (TOGAF)
ADM	Architecture Development Method (TOGAF)
BPM	Business Process Management
COTS	Commercial Off-the-Shelf
CSF	Critical Success Factor
DAO	Data Access Object
DTO	Data Transfer Object
EIP	Enterprise Integration Patterns
ESB	Enterprise Service Bus
ETL	Extract, Transform, and Load
KPI	Key Performance Indicator
L2SOA	Legacy Evolution to SOA
LTE	Long-Term Evolution
OGC	Open Geospatial Consortium, Inc.
OSIMM	The Open Group Service Integration Maturity Model, an Open Group Standard
OTTF	The Open Group Trusted Technology Forum, a Forum of The Open Group
SaaS	Software as a Service
SBB	Solution Building Blocks (TOGAF)
SEI	Service Endpoint Interface
SGRM	SOA Governance Reference Model
SGVM	SOA Governance Vitality Method
SID	Shared Information and Data
SLA	Service-Level Agreement
SOA	Service-Oriented Architecture

SOA RA	The Open Group SOA Reference Architecture
SSO	Single-Sign-On
TCO	Total Cost of Ownership

Index