

SOA Source Book



THE *Open* GROUP

www.opengroup.org

SOA Source Book

Other publications by Van Haren Publishing

Van Haren Publishing (VHP) specializes in titles on Best Practices, methods and standards within four domains:

- IT management,
- Architecture (Enterprise and IT),
- Business management and
- Project management

These publications are grouped in series: *ITSM Library, Best Practice and IT Management Topics*. VHP is also publisher on behalf of leading companies and institutions: The Open Group, IPMA-NL, PMI-NL, CA, Getronics, Pink Elephant.

Topics are (per domain):

IT (Service) Management / IT Governance

ASL
BiSL
CATS
CMMI
COBIT
ISO 17799
ISO 27001
ISO/IEC 20000
ISPL
IT Service CMM
ITIL® V2
ITIL® V3
ITSM
MOF
MSF

Architecture (Enterprise and IT)

Archimate
GEA
TOGAF™

Business Management

EFQM
ISA-95
ISO 9000
ISO 9001:2000
SixSigma
SOX
SqEME®

Project/Programme/ Risk Management

A4 Project management
ICB / NCB
MINCE®
M_o_R®
MSP
PMBok
PRINCE2®

For the latest information on VHP publications, visit our website: www.vanharen.net.

SOA Source Book

THE *Open* GROUP
www.opengroup.org



Colofon

Title:	SOA Source Book
A publication of:	The Open Group
Publisher:	Van Haren Publishing, Zaltbommel, www.vanharen.net
ISBN:	978 90 8753 503 2
Print:	First edition, first impression, April 2009
Layout and design:	CO2 Premedia, Amersfoort-NL
Copyright:	© 2009, The Open Group

For any further enquiries about Van Haren Publishing, please send an e-mail to:
info@vanharen.net

Trademarks

TOGAF™ is a trademark and The Open Group® is a registered trademark of The Open Group in the United States and other countries.

All other brand, company, and product names are used for identification purposes only and may be trademarks that are the sole property of their respective owners.

Comments relating to the material contained in this document may be submitted by email to: ogspeccs@opengroup.org

Although this publication has been composed with most care, neither Author nor Editor nor Publisher can accept any liability for damage caused by possible errors and/or incompleteness in this publication.

No part of this publication may be reproduced in any form by print, photo print, microfilm or any other means without written permission by the publisher.

Contents

Chapter 1	Service-Oriented Architecture	1
1.1	What Is SOA?	1
1.1.1	Service-Oriented Architecture	1
1.1.2	SOA Architectural Style	2
1.2	SOA and Enterprise Architecture	2
1.2.1	Enterprise Architecture	3
1.2.2	SOA	4
1.2.3	Overview of SOA	5
1.2.4	Architectural Dimension of SOA	6
1.2.5	Mainstream SOA	7
1.3	SOA and Boundaryless Information Flow	8
1.3.1	The Problem	8
1.3.2	Boundaryless Information Flow through SOA	8
1.4	SOA Features and Benefits	10
1.4.1	Summary of Features and Benefits	11
1.4.2	Service	12
1.4.3	Service Re-Use	13
1.4.4	Messaging	14
1.4.5	Message Monitoring	14
1.4.6	Message Control	14
1.4.7	Message Transformation	15
1.4.8	Message Security	15
1.4.9	Complex Event Processing	15
1.4.10	Service Composition	16
1.4.11	Service Discovery	17
1.4.12	Asset Wrapping	18
1.4.13	Virtualization	19
1.4.14	Model-Driven Implementation	19
1.5	Maturity Model for SOA	20
1.5.1	Overview of the Model	20
1.5.2	Using the Model	20
1.5.3	Dimensions	22
1.5.4	Maturity Levels	23

Chapter 2	The SOA Reference Architecture	27
2.1	The Building Blocks of SOA	27
2.1.1	Services	28
2.1.2	Business Processes	28
2.1.3	Human Actors	29
2.1.4	Events	30
2.1.5	Service Descriptions, Contracts, and Policies	30
2.1.6	Service Compositions	31
2.1.7	Programs	32
2.1.8	Information Items, Data Items, and Data Stores	32
2.2	A High-Level Perspective of the SOA Reference Architecture	33
2.2.1	Overview	33
2.2.2	Operational Systems Layer	35
2.2.3	Service Components Layer	35
2.2.4	Services Layer	36
2.2.5	Business Processes Layer	36
2.2.6	Consumers Layer	37
2.2.7	Integration Layer	37
2.2.8	Quality of Service Layer	37
2.2.9	Information Layer	38
2.2.10	Governance Layer	38
2.3	Detailed Building Blocks of the SOA Reference Architecture	38
2.3.1	Composition	39
2.3.2	Messaging	40
2.3.3	Service Discovery	42
2.3.4	Asset Wrapping	43
2.3.5	Virtualization	43
2.3.6	Event Processing	44
2.4	Infrastructure for SOA	45
2.4.1	Service Repository	46
2.4.2	Messaging Program	46
2.4.3	Activity Monitor	48
2.4.4	PDPs and PEPs	49
2.4.5	Data Translator	49
2.4.6	Encryption Engine	50
2.4.7	Event Processor	51
2.4.8	Composition Engine	51
2.4.9	Service Registry	52

2.4.10	Service Components	53
2.4.11	Model-Implementation Environment	53
Chapter 3 Service-Oriented Infrastructure		55
3.1	Overview	55
3.2	SOI Reference Model.....	56
3.2.1	Business Requirements	56
3.2.2	Service-Level Requirements.....	57
3.2.3	BPM.....	57
3.2.4	IMF.....	57
3.2.5	Applications.....	57
3.2.6	Services Exposed to and Consumed by IT.....	57
3.2.7	Infrastructure Services.....	58
3.2.8	Physical Services	58
3.2.9	Virtualized Services.....	58
3.3	Infrastructure Services	58
3.3.1	Pure-Play Infrastructure Services.....	59
3.3.2	Cross-Application Infrastructure Services.....	60
3.3.3	Operational Infrastructure Services.....	61
3.4	Virtualized Services	61
Chapter 4 SOA and TOGAF		63
4.1	Using TOGAF for Enterprise SOA.....	63
4.1.1	Preliminary Phase	64
4.1.2	Architecture Requirements Management	66
4.1.3	Phase A: Architecture Vision.....	66
4.1.4	Phase B: Business Architecture	70
4.1.5	Phase C: Information Systems Architectures.....	71
4.1.6	Phase D: Technology Architecture	74
4.1.7	Phase E: Opportunities and Solutions	75
4.1.8	Phase F: Migration Planning.....	76
4.1.9	Phase G: Implementation Governance.....	76
4.1.10	Phase H: Architecture Change Management	76
4.2	Using TOGAF for SOA Solutions.....	77
4.2.1	Preliminary Phase	77
4.2.2	Architecture Requirements Management	78
4.2.3	Phase A: Architecture Vision.....	78
4.2.4	Phase B: Business Architecture	78

4.2.5	Phase C: Information Systems Architectures.....	79
4.2.6	Phase D: Technology Architecture.....	81
4.2.7	Phase E: Opportunities and Solutions.....	82
4.2.8	Phase F: Migration Planning.....	82
4.2.9	Phase G: Implementation Governance.....	82
4.2.10	Phase H: Architecture Change Management.....	82
4.3	Addressing Stakeholder Concerns in SOA.....	83
4.3.1	Stakeholders.....	83
4.3.2	Concerns and Models.....	84
4.4	Information Architecture for SOA.....	87
4.4.1	Importance of Information Architecture for SOA.....	88
4.4.2	Information Architecture for SOA using TOGAF.....	89
4.4.3	Preliminary Phase.....	90
4.4.4	Phase A: Architecture Vision.....	91
4.4.5	Phase B: Business Architecture.....	91
4.4.6	Phase C: Information Systems Architectures.....	92
4.4.7	Phase D: Technology Architecture.....	93
Chapter 5 SOA Governance.....		95
5.1	Introduction to SOA Governance.....	95
5.1.1	Definition of SOA Governance.....	96
5.1.2	Enterprise SOA Governance Models.....	97
5.1.3	SOA Governance Framework.....	98
5.2	SOA Governance Reference Model.....	99
5.2.1	Governance Principles.....	99
5.2.2	SOA Activities.....	100
5.2.3	Governing Processes.....	101
5.2.4	Roles and Responsibilities.....	102
5.3	SOA Governance Vitality Method.....	103
5.3.1	The Plan Phase.....	104
5.3.2	The Define Phase.....	104
5.3.3	The Implement Phase.....	104
5.3.4	The Monitor Phase.....	105
Index.....		107

List of Figures

Figure 1: Overview of a Service-Oriented Architecture.....	5
Figure 2: Boundaryless Information Flow.....	9
Figure 3: Information Silos.....	9
Figure 4: SOA for Boundaryless Information Flow	10
Figure 5: SOA Maturity Model Matrix	21
Figure 6: High-Level Perspective of the SOA Reference Architecture	34
Figure 7: Basic Model for Service Composition	39
Figure 8: Model for Scripted Service Composition.....	39
Figure 9: Basic Messaging Model	40
Figure 10: Detailed Messaging Model.....	41
Figure 11: Model for Service Discovery	42
Figure 12: Model for Asset Wrapping	43
Figure 13: Model for Virtualization.....	44
Figure 14: Model for Event Processing.....	45
Figure 15: Service-Oriented Infrastructure Reference Model.....	56
Figure 16: Example ODBC Service Stack.....	60
Figure 17: SOA Governance Relationships	96
Figure 18: SOA Governance Aspects	97
Figure 19: SOA Governance Framework.....	99
Figure 20: SOA Governance Vitality Method	103

List of Tables

Table 1: SOA Features, Benefits, and Infrastructure.....	12
Table 2: SOA Concerns, Stakeholders, and Models.....	87

Preface

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX® certification.

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

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at www.opengroup.org/certification.

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

This Document

The Open Group's *SOA Source Book* is a collection of source material for use by enterprise architects working with Service-Oriented Architecture (SOA).

It consists of material that has been considered and in part developed by The Open Group SOA Working Group¹. The SOA Working Group is engaged in a work program to produce definitions, analyses, recommendations, reference

¹ Refer to www.opengroup.org/projects/soa.

models, and standards to assist business and information technology professionals within and outside of The Open Group to understand and adopt SOA. The Source Book does not represent the final output of that work program, which will be published as a collection of Open Group Standards and Guides. It is an interim publication, and its content will not necessarily be reflected in the final output.

The material reflects input from a large number of people from a wide range of Open Group member companies, including product vendors, consultancies, and users of SOA. In some cases, these people have brought concepts developed, not just by themselves, but by groups of people within their organizations. The input has been refined and further developed through discussion within the Working Group. The value in the result is due to the ideas and efforts of the Working Group members.

The material is now published in its current form to make that value available to the wider architecture community.

Chapter 1 discusses SOA in relation to enterprises, and describes how to evaluate SOA features in business terms.

Chapter 2 presents The Open Group SOA Reference Architecture.

Chapter 3 describes how to apply the principle of service-orientation to infrastructure.

Chapter 4 explains how to use TOGAF - the comprehensive architecture framework developed and maintained by The Open Group - for SOA.

Chapter 5 describes SOA governance, and provides an initial explanation of how to define and maintain an SOA governance regimen for an enterprise.

Trademarks

Boundaryless Information Flow™ and TOGAF™ are trademarks and Making Standards Work®, The Open Group®, UNIX®, and the “X” device are registered trademarks of The Open Group in the United States and other countries.

Model Driven Architecture® and MDA® are registered trademarks, and Business Process Modeling Notation™, BPMN™, MOF™, and Unified Modeling Language™ are trademarks of the Object Management Group, Inc. in the United States and/or other countries.

OASIS® is a registered trademark, and Security Assertion Markup Language™ and SAML™ are trademarks of OASIS.

W3C® is a registered trademark (registered in numerous countries), and XML™ and XSL™ are trademarks of the World-Wide Web Consortium (W3C); marks of W3C are registered and held by its host institutions MIT, ERCIM, and Keio.

The Open Group acknowledges that there may be other brand, company, and product names used in this document that may be covered by trademark protection and advises the reader to verify them independently.

Acknowledgements

The Open Group gratefully acknowledges the following people in contributing, either directly or indirectly, to the SOA Source Book.

The material in the book is derived from the work of the Definition of SOA, SOA Reference Architecture, SOA/TOGAF Practical Guide, SOA Governance, Service-Oriented Infrastructure, and SOA Ontology projects of The Open Group SOA Working Group, from work done by The Open Group Service Integration Maturity Model project, and also from work done jointly by the SOA Working Group and the Semantic Interoperability Working Group.

The co-chairs of the SOA Working Group, Tony Carrato (IBM) and Mats Gejnevall (Capgemini), together with former co-chair Chris Greenslade (CLARS), member Jorge Diaz (IBM), and Forum Director Chris Harding (The Open Group), comprise the Working Group's Steering Committee. They are responsible for the overall direction of the Working Group and contribute greatly to the quality of its work. In addition, Tony Carrato took a particular interest in guiding the development of the Source Book. Chris Harding was primary author.

The Definition of SOA project was led by Dave Hornford (Hornford Associates).

The SOA Reference Architecture project is led by Ali Arsanjani (IBM) and Nikhil Kumar (ApTSi). Ali Arsanjani made a particular contribution in providing the base document for the work, having led its development within IBM.

The SOA/TOGAF Practical Guide project is led currently by Awel Dico (Bank of Montreal) and Dave Hornford, and was led formerly also by Steve Bennett (BEA Systems).

The SOA Governance project is led currently by Mats Gejnevall and Jorge Diaz, and was led formerly also by Andrew Hatley (IBM), Tony Carrato, and Steve Bennett. In addition, Bill Brown (IBM) provided a substantial part of its base material.

The Service-Oriented Infrastructure project is led currently by Hemesh Yadav (Wachovia), E.G. Nadhan (HP), and Michael Salsburg (Unisys), and was led formerly also by Mark England (HP) and Frank Kroon (formerly Capgemini, now HP). E.G. Nadhan authored the Service-Oriented Infrastructure section of the Source Book.

The SOA Ontology project is led by Chris Harding.

The Open Group Service Integration Maturity Model project is led by Andras Szakal (IBM). He was responsible for providing the project's base document, which resulted from work led by Ali Arsanjani within IBM.

The contribution of the Semantic Interoperability Working Group was led by Arnold Van Overeem (CapGemini) and Ron Schuldt (Lockheed Martin).

Many of the people mentioned above also made contributions to projects of which they were not officers.

The following Working Group members, who were not project or working group officers, made particular contributions to one or more projects: Stuart Boardman (CGI), Kathy Carusone (MIT Lincoln Laboratory), Dave Chapelle (BEA Systems), Bill Estrem (Metaplexity), Ed Harrington (Model-Driven Solutions), Harry Hendrickx (Capgemini), Heather Kreger (IBM), Bob Laird (IBM), Srikanth Inaganti (Wipro), Shreyas Kamat (Infosys), Rich Valentine (Unisys), and Bobbi Young (Unisys).

Finally, over 300 other people have been involved in the SOA Working Group. It is not possible to mention them all individually, but their collective contribution is important.

Referenced Documents

The following documents are referenced in this Source Book:

- The Boundaryless Organization: Breaking the Chains of Organizational Structure, by Ron Ashkenas, Dave Ulrich, Todd Jick, & Steve Kerr; ISBN 0-7879-5943-X.
- Control Objectives for Information and related Technology (COBIT), Version 4.1, available from ISACA; refer to www.isaca.org.
- Interoperable Enterprise Business Scenario (K022), published by The Open Group; refer to www.opengroup.org/bookstore/catalog/k022.htm.
- ISO/IEC 2382-1:1993, Information Technology – Vocabulary – Part 1: Fundamental Terms.
- OECD Corporate Governance Principles, 2004, available from the Organization for Economic Cooperation and Development; refer to www.oecd.org.
- TOGAF; refer to www.opengroup.org/togaf.
- The SOA Solution Stack: A Reference Architecture for Designing SOA Solutions, IBM Corporation.
- The following standards defined by OASIS; refer to www.oasis-open.org:
 - Business Process Execution Language (BPEL)
 - Security Assertion Markup Language (SAML)
 - Universal Description Discovery and Integration (UDDI)
 - Web Services Reliable Messaging (WS-ReliableMessaging)
 - Web Services Security (WS-Security)
 - Web Services Security Policy (WS-Security-Policy)
 - eXtensible Access Control Markup Language (XACML)
- The following standards, defined by the Object Management Group (OMG); refer to www.omg.org:
 - Business Process Modeling Notation (BPMN)
 - Meta Object Facility (MOF)
 - Unified Modeling Language (UML)
- The following standards, defined by the World-Wide Web Consortium (W3C); refer to www.w3.org:
 - Simple Object Access Protocol (SOAP)
 - Web Ontology Language (OWL)
 - Web Services Description Language (WSDL)

- Web Services Policy Framework (WS-Policy)
- eXtensible Markup Language (XML)
- eXtensible Stylesheet Language (XSL) Transformations (XSLT)

Chapter 1

Service-Oriented Architecture

This section discusses Service-Oriented Architecture (SOA) in relation to enterprises, and describes how to evaluate SOA features in business terms. It contains:

- A definition of SOA
- An analysis of the role of SOA in relation to enterprise architecture
- An explanation of how SOA can enable an enterprise to achieve Boundaryless Information Flow
- A description of the features of SOA and the business benefits that they provide
- An SOA maturity model that facilitates the assessment of an organization's current and desired future states in service integration and flexibility, and helps the organization to determine its architectural strategy for adopting service-orientation

1.1 What Is SOA?

This definition of SOA was produced by the SOA Definition team of The Open Group SOA Working Group.

1.1.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an *architectural style* that supports *service-orientation*.

Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

A service:

- Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
- Is self-contained

- *May be* composed of other services
- Is a “black box” to consumers of the service

1.1.2 SOA Architectural Style

An *architectural style* is the combination of distinctive features in which architecture is performed or expressed.

The SOA architectural style has the following distinctive features:

- It is based on the design of the services – which mirror real-world business activities – comprising the enterprise (or inter-enterprise) business processes.
- Service representation utilizes business descriptions to provide context (i.e., business process, goal, rule, policy, service interface, and service component) and implements services using service orchestration.
- It places unique requirements on the infrastructure – it is recommended that implementations use open standards to realize interoperability and location transparency.
- Implementations are environment-specific – they are constrained or enabled by context and must be described within that context.
- It requires strong governance of service representation and implementation.
- It requires a “Litmus Test”, which determines a “good service”.

1.2 SOA and Enterprise Architecture

SOA provoked hot debate when it burst onto the scene in 2005. Its advocates said that it would replace traditional information technology (IT) architecture. The traditionalists replied that SOA was nothing new; just a rehash of old (but good) ideas about encapsulation and loose coupling.

There is some truth in both of these positions. But in the main they are both wrong. Although SOA does include earlier architectural ideas, it is a distinct style which marks a major step forward. And, to obtain maximum benefit from SOA, an enterprise needs traditional architectural disciplines and methods.

1.2.1 Enterprise Architecture

Why does an enterprise need an SOA – or an architecture of any other kind?

The directing function of an enterprise – the board of directors of a commercial company, or the top-level management of a division or government department, for example – sets objectives for the enterprise, and decides how it should operate in order to achieve them. A clearly articulated architecture describes the desired enterprise organization and manner of operation. By doing so, it provides:

- A definition of the changes that should be implemented to achieve this organization
- A basis for control and governance of its ongoing operation

An enterprise architecture also provides a third benefit. Enterprises change over time. They combine and split, as in commercial mergers and spin-offs, or government department reorganizations. It is easier to combine an enterprise with another, or to split it into component parts, when it has a clearly-defined architecture. This brings significant cost savings, and can increase the value of a commercial enterprise.

Enterprise architecture in its widest sense includes much more than IT. It covers business operations, finance, people, and buildings in addition to technology, and it covers technologies other than IT, such as for manufacturing or transport. The enterprise architect must understand these areas, at least well enough to supervise architects that specialize in them. The IT architect must be able to work in teams with such specialists.

The SOA Source Book focuses on the IT component of enterprise architecture. This is concerned with the strategic development of an enterprise's IT. It looks at the whole of the enterprise, not just a particular system, and it looks at the long-term evolution of the IT, not just at what should be installed today.

The quality of an enterprise's IT architecture can have a major impact on its business performance. Since the 1950s, commercial and government organizations have become increasingly dependent on IT for the conduct of their everyday operations, and that trend looks likely to continue. Companies that use IT effectively prosper. The best of the once-derided .com companies

(“When will they ever make a profit?”) became household names. Companies with poor IT fall behind their competition, or fail.

Because of its importance to the overall business, enterprise IT architecture has become a profession. No company would think of undertaking the development of a major building without engaging a buildings architect with a professional status that provides a guarantee of competency. Similarly, companies undertaking the development of major IT systems look for professional enterprise IT architects. Their status as professionals indicates that they understand, and have a track record of applying, the best IT architecture methods and techniques.

1.2.2 SOA

An enterprise architect looks at the overall construction of the enterprise. SOA is a particular construction technique that can be used to build enterprise IT.

A particular technique can have a major impact on the overall construction. The introduction of steel-frame techniques in the latter part of the 19th century revolutionized buildings architecture. It made possible the skyscrapers of the 1920s, and the even larger buildings that we have today.

SOA could have a similar impact on IT architecture. It does not increase the size of IT systems, but it does increase their interoperability.

With SOA, the IT systems perform services that are defined and described in the context of the enterprise’s business activities. Each service is identified, and what it does is clearly set out in the form of a contract. This principle enables use of techniques such as service composition, discovery, message-based communication, and model-driven implementation, which give fast development of effective and flexible solutions. They are important features of SOA. Their benefits – especially that of enterprise agility – are the most frequently quoted reasons for SOA adoption.

But it is the replacement of large, monolithic applications that have tiny interoperability interfaces, grudgingly provided and not guaranteed, by smaller, modular services that have interface descriptions and contracts, that is the most fundamental effect of SOA. This is the basis for the huge increase

in IT system interoperability that SOA can bring, not only within enterprises, but also between enterprises.

1.2.3 Overview of SOA

The principle of service-orientation can apply throughout the enterprise architecture, but is most commonly applied to the organization of the software that supports the enterprise's business operations. With SOA, this software is organized as a set of software services. The services are supported by an infrastructure that, together with the services, improves information flow within the enterprise and between the enterprise and external enterprises.

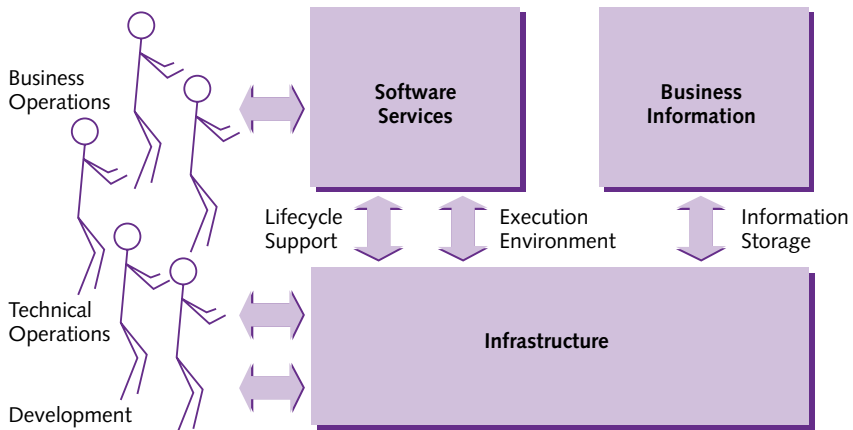


Figure 1: Overview of a Service-Oriented Architecture

The software services are used by the enterprise's business operations. This frequently involves a human-computer interface, often implemented as a web interface using portals, etc., but it may also involve other interfaces, such as machine interfaces for process control.

Specific sets of business processes, services, and interfaces are created in the context of a supporting infrastructure as service-based *solutions*. Each solution solves a particular business problem.

The business operations themselves may be organized on the service-oriented principle. Indeed, there are many people who believe that the greatest benefits of SOA are obtained when it is applied to the business architecture.

The infrastructure provides the execution environment for the software services. This includes the basic operating system and networking, and also includes specific support for software services, such as message passing and service discovery. The infrastructure is managed via human-computer interfaces by technical staff who are responsible for all aspects of operating the enterprise's IT, including its availability, performance, and security.

A major benefit of SOA is that it delivers enterprise agility, by enabling rapid development and modification of the software that supports the business processes. The infrastructure can provide for this by including facilities such as business-oriented scripting languages and model-driven implementation tools. These facilities support not only the creation of new software services, but also the modification and replacement of existing ones: the whole service lifecycle. They are used via human-computer interfaces by development staff.

The infrastructure also provides for storage of enterprise information. SOA can enable easier flow of information within and between enterprises. The information is not locked up in specific services, as it often is in the so-called "silo" applications of earlier architecture styles, but is available to all the software services that need it.

Service-orientation may extend to the design of the infrastructure, and many people advocate this, but it is not essential to service-oriented software architecture.

1.2.4 Architectural Dimension of SOA

It takes far greater knowledge and skill to erect a skyscraper than to build a house. The buildings architect must make complex stress calculations based on an understanding of the properties of the materials involved. Training and experience are essential for success.

Knowledge and skill are also needed for success with SOA. The IT architect must specify the right tools and infrastructure, create the basis for the identification of modular services, and ensure that appropriate implementation governance is in place. Good judgment in these matters is crucial.

Also, just as steel-frame construction is not appropriate for every building, SOA is not necessarily the right approach to solving every IT problem. The IT architect must know when, as well as how, to use SOA.

SOA can be a big investment. Its tools and infrastructure cost money, but that is only one part of what is needed. Development and operation staff must have special skills to create and use SOA, and the overall organization structure and culture must be right if the full benefits of SOA are to be achieved. Staff development and organizational change is often the larger part of the investment. Such an investment can only be justified in the light of a long-term strategy for the enterprise as a whole.

Many enterprises have undertaken small-scale SOA developments as part of a learning process. This is an excellent way for them to introduce SOA, but they often find it hard to extend beyond the initial pilot. Developers complain that they cannot justify the infrastructure that they need. Of course not! Expensive infrastructure cannot be justified on the basis of small projects and, in any case, looking for business justification for technical spend is putting the cart before the horse. The business need should come before the technical solution. SOA should be used where – and only where – it is the best way to meet that need.

This is where enterprise architecture comes in. Enterprise architecture creates long-term IT strategy in the light of business possibilities and needs. Inclusion in such a strategy is the only good justification for large-scale SOA.

1.2.5 Mainstream SOA

SOA is no longer a new toy. It is an established style that architects understand and can use.

The architect does not start by assuming SOA, but considers service-orientation and its associated techniques in the light of the business strategy. Sometimes, the technical possibilities can change that strategy, but the business needs and possibilities are still the main driving force. The architect finishes by specifying a particular combination of SOA techniques because it best realizes the possibilities and meets the needs.

This is the normal architectural approach to IT strategy. SOA and enterprise architecture may have seemed different in the beginning, but SOA is now part of the enterprise architecture mainstream.

1.3 SOA and Boundaryless Information Flow

Why is SOA important to The Open Group?

The Open Group's vision is Boundaryless Information Flow. It has long been a principle of enterprise organization that permeable boundaries between departments, organizational levels, enterprises, and nations deliver productivity and enterprise agility. This was established in the 1980s by pioneers such as Jack Welch of GE (see *The Boundaryless Organization: Breaking the Chains of Organizational Structure*). But traditional IT architectures hinder this! The need for Boundaryless Information Flow – provided by IT architectures that enable information to flow freely across the permeable organizational boundaries – was identified by The Open Group and described in its Interoperable Enterprise Business Scenario. The Open Group took on the mission of driving the creation of Boundaryless Information Flow.

1.3.1 The Problem

Enterprise architecture is the key to achieving Boundaryless Information Flow. The problem, as described in the Interoperable Enterprise Business Scenario, is that enterprises need the kind of architecture shown in Figure 2, in which the business processes are supported by systems that can exchange information freely.

Too often, however, they are faced with a situation where each business process has its own system which has its own particular interfaces and information formats, and is a so-called “information silo”, as shown in Figure 3.

1.3.2 Boundaryless Information Flow through SOA

With SOA, the applications are replaced by services that interact with each other. Typically, interactions take place by exchange of messages via an Enterprise Services Bus (ESB) within the enterprise, or across the web in the case of external services, although other forms of interaction, even direct

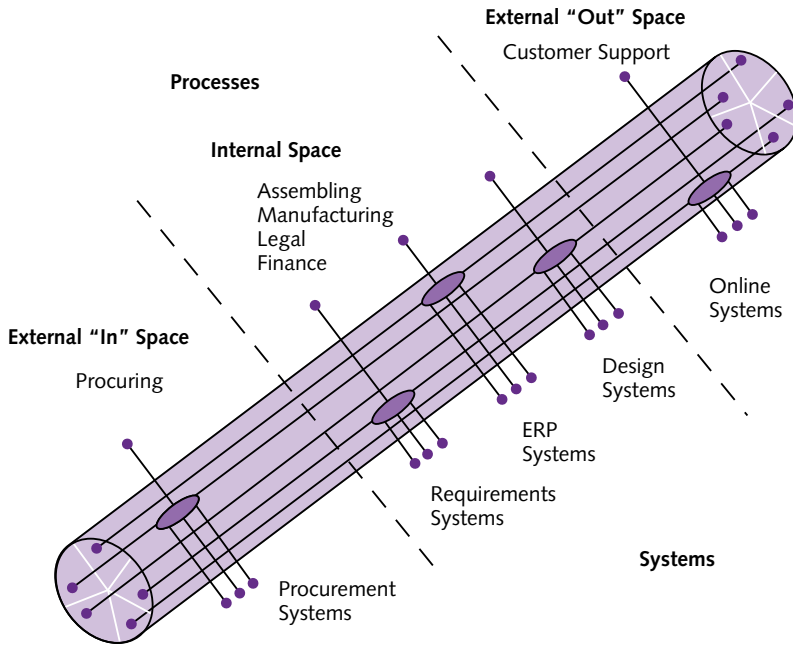


Figure 2: Boundaryless Information Flow

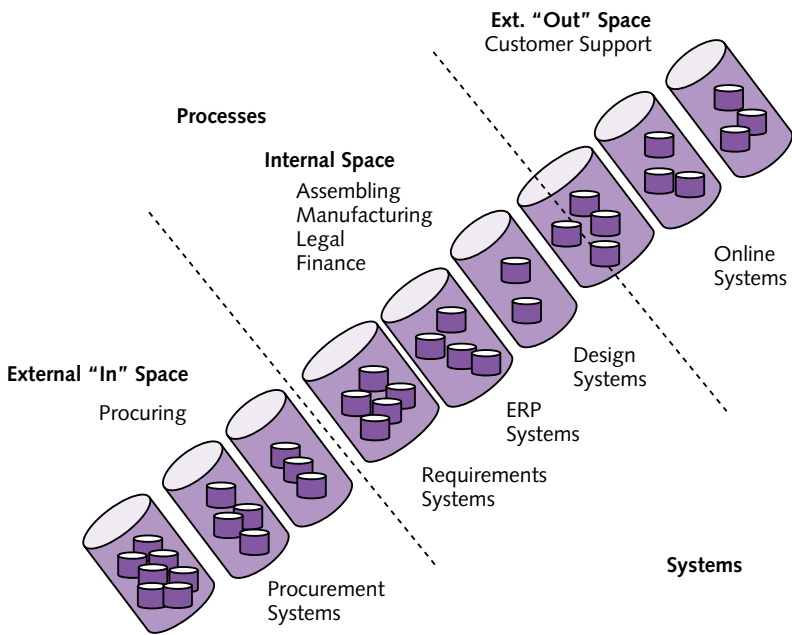


Figure 3: Information Silos

invocation of one service by another (so-called “hard-wiring”) may be used. This style of architecture can be the basis of Boundaryless Information Flow, as illustrated in Figure 4.

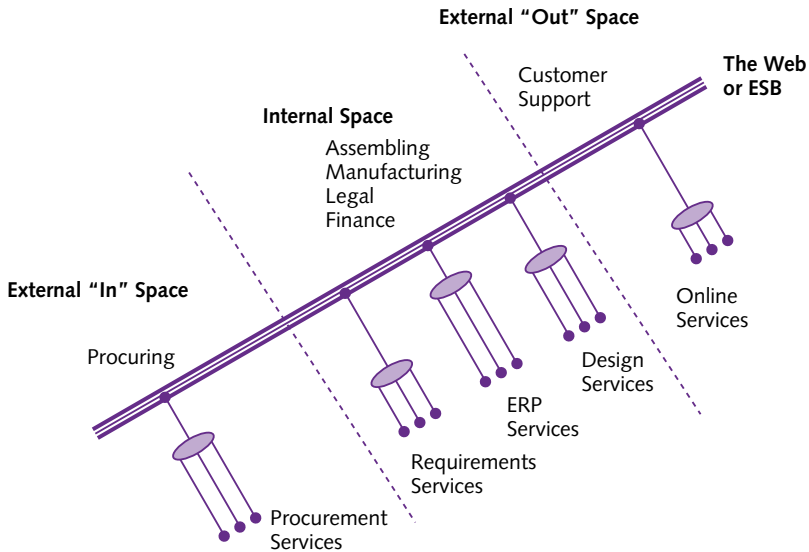


Figure 4: SOA for Boundaryless Information Flow

It is because of the potential for SOA to deliver Boundaryless Information Flow that SOA is critically important to The Open Group.

1.4 SOA Features and Benefits

SOA starts with a simple idea – the concept of *service*. This makes it possible to introduce other ideas, such as *service bus*, *service composition*, and *service virtualization*, each of which can be applied to the architecture of an enterprise to deliver benefits. As an architect, it is your job to evaluate the needs of your enterprise, and the costs of the different potential solutions, to determine which of these ideas should be applied, and how they should be applied, in your SOA.

An architect should always probe into the information given, about both requirements and solutions, to reach a level of understanding that goes deeper than the buzzwords. For example, it is often said that “SOA delivers

enterprise agility”. What does “agility” mean for your enterprise? Is it the ability to re-combine existing functions to meet changing customer requirements? Is it the ability to develop new functions rapidly? Is it the ability to scale operations to meet different levels of demand? Within the broad concept of SOA, there are three very different ideas that can help you meet these different agility requirements: service composition, model-driven development, and service virtualization. You can build all of these ideas into your SOA, but they each require different – and expensive – supporting infrastructure. You must choose your solution to fit the requirements.

This section will help you to match the features of SOA to the needs of your enterprise, so that you can determine the kind of SOA that is appropriate.

1.4.1 Summary of Features and Benefits

Table 1 shows the main features and benefits of SOA, together with the infrastructure needed to support them.

Feature	Benefits	Supporting Infrastructure
Service	Improved information flow Ability to expose internal functionality Organizational flexibility	
Service Re-use	Lower software development and management costs	Service repository
Messaging	Configuration flexibility	Messaging service
Message Monitoring	Business intelligence Performance measurement Security attack detection	Activity monitor
Message Control	Application of management policy Application of security policy	PDPs and PEPs
Message Transformation	Data translation	Data translator
Message Security	Data confidentiality and integrity	Encryption engine
Complex Event Processing	Simplification of software structure Ability to adapt quickly to different external environments Improved manageability and security	Event processor
Service Composition	Ability to develop new function combinations rapidly	Composition engine

Feature	Benefits	Supporting Infrastructure
Service Discovery	Ability to optimize performance, functionality, and cost Easier introduction of system upgrades	Service registry
Asset Wrapping	Ability to integrate existing assets	
Virtualization	Improved reliability Ability to scale operations to meet different demand levels	
Model-driven Implementation	Ability to develop new functions rapidly	Model-implementation environment

Table 1: SOA Features, Benefits, and Infrastructure

1.4.2 Service

Service is the essential concept of SOA.

It is not originally a technical concept. The idea of a service was developed in the world of business. Look in any “Yellow Pages” directory, and you will find categories such as “courier services”, “garage services”, and “roofing services”. For each of these, some person or company (the service provider) is offering to do something – carry goods and messages, look after vehicles, install and repair building roofs – that will benefit other people or companies (the service consumers). The providers offer to contract with the consumers to do these things, so that the consumers know in advance what they will get for their money.

The idea has been adopted by technologists. They have established the concept of a *software service*. A software service is performed by a software program. It produces effects that have value to the people or organizations that are its consumers. It has a provider – a person or organization that takes responsibility for running the program to produce those effects. And there is an implicit or explicit contract between the provider and the consumers that the program will produce the effects that the consumers expect.

Software services can be provided over the Internet and the world-wide web. In some countries, for example, the government provides a service by which taxpayers can complete and submit their tax returns via the web. Here,

the service has a human interface. Services provided over the web can also have software interfaces. For example, there are commercially-available web services that provide real-time stock quote information in a form where it can be analyzed by the consumers' software. Software services can similarly be provided over enterprises' internal networks, and a service performed by one program can be used by another program running on the same computer system. It is the organization of an enterprise's software as software services that are provided internally in this way, and also externally, that is the essential characteristic of SOA.

The use of services provides major benefits:

- In contrast to the use of large applications, which tend to be “information silos” that cannot readily exchange information with each other, the use of finer-grained software services gives freer information flow within and between enterprises. Integrating major applications is often expensive. SOA can save integration costs.
- Organizing internal software as services makes it easier to expose its functionality externally. This leads to increased visibility that can have business value as, for example, when a logistics company makes the tracking of shipments visible to its customers, increasing customer satisfaction and reducing the costly overhead of status enquiries.
- Business processes are often dependent on their supporting software. It can be hard to change large, monolithic programs. This can make it difficult to change the business processes to meet new requirements (arising, for example, from changes in legislation) or to take advantage of new business opportunities. A service-based software architecture is easier to change – it has greater organizational flexibility, enabling it to avoid penalties and reap commercial advantage. (This is one of the ways in which SOA can make an enterprise more “agile”.)

The service concept also makes possible further features of SOA. These can provide additional benefits, as explained in the rest of this section.

1.4.3 Service Re-Use

Clear service descriptions are a starting point for service re-use, which can provide another major benefit of SOA:

- Using existing software modules rather than writing new ones means lower development and testing costs and – in many cases an even greater saving – lower maintenance costs.

1.4.4 Messaging

You can have an SOA in which software services invoke each other directly; for example, by programming-language function calls. But, in many SOAs, the software services always invoke each other by exchanging messages, even where they are executing on the same processor. This might seem to be an additional overhead but, if the services are loosely-coupled (as they should be), then the number of message exchanges is relatively small, and the overhead is reasonably low.

Consistent use of messaging provides a key benefit:

- Services can very easily be moved between computer systems within the enterprise, and it is reasonably easy to use externally-provided services to replace internal ones, and *vice versa*. Which services handle which messages can be changed rapidly to meet changing business needs, or to tune performance. In short, messaging provides significant configuration flexibility.

Having a central mechanism by which all messages are exchanged facilitates monitoring, control, transformation, and security of messages.

1.4.5 Message Monitoring

Message monitoring can provide three key benefits:

- Monitoring message streams between business activities, and analyzing them to obtain information about those activities, is known as *business activity monitoring*. It can be a valuable source of business intelligence.
- Monitoring message volumes and response times is a valuable source of performance measurement. Service contracts often include performance clauses. Performance measurement enables service designers to put realistic clauses into the contracts, and enables systems managers to verify that those clauses are being met.
- Monitoring messages and message volumes can provide security attack detection, including detection of denial-of-service attacks as well as of attacks in particular messages.

1.4.6 Message Control

Message control can provide:

- Application of management policy; for example, by restricting a service consumer to a contracted service volume, or giving priority to certain kinds of message

- Application of security policy; for example, by controlling access to certain services, or rejecting messages that could damage the enterprise systems or the enterprise itself (e.g., messages containing viruses that could destroy data)

1.4.7 Message Transformation

Message transformation can provide:

- Data translation – the conversion of data from one format to another through automated field mapping.

Data conversion by specially-written software is expensive. The use of generic data translators can bring significant cost saving.

1.4.8 Message Security

Message security can include:

- Data confidentiality through encryption of messages
- Data integrity through addition of cryptographic integrity-check fields

Security is a complex area that is of crucial importance to enterprises. The ability to encrypt and apply integrity checking to messages in transit can be a valuable component of an overall security strategy.

1.4.9 Complex Event Processing

As well as being invoked by their consumers, services can respond to events from other sources. For example, a financial information service might respond to stock-price changes, or a manufacturing production-control service might respond to production process events, such as changes in temperature of the materials being processed.

In many cases, action is taken when a pattern of events is recognized, rather in response to individual events. A financial information service might notify the user when a volume of trades is exceeded rather than in response to a single trade. A production-control service might take measurements from a number of sensors and take action when the average exceeds a limit. This aggregation of simple events to generate complex events is known as Complex Event Processing (CEP).

In SOA, CEP is often used, not only for external events, but also to detect patterns in the flow of messages between services. When used in this way, it becomes an extension of message monitoring.

CEP is often linked with business activity monitoring. For example, detection of a particular pattern in sales transaction messages could provide advance warning of difficulties for the production process. In some industries, such as banking, detection of particular patterns may indicate fraudulent activity, or assist with regulatory compliance.

CEP can also be used with performance measurement and security attack detection. For example, where a service contract specifies an average level of performance, CEP used in conjunction with performance measurement could generate contract exception events. CEP might also be used to generate security events for unusual message volumes or patterns.

CEP provides the following benefits:

- Simplification of software structure – by removing functionality that is not business-related from the business software services
- Ability to adapt quickly to different external environments – by concentrating in one place the logic that relates environmental events to business events
- Improved manageability and security – when used with performance measurement and security event detection

1.4.10 Service Composition

Service composition is the putting together of a number of simple services to make a more complex one. For example, a “product sale” web service could be composed of simpler “product selection”, “shopping cart review”, “payment method selection”, “credit card payment”, and “invoice payment” services.

Service composition provides a key benefit:

- Ability to develop new function combinations rapidly

For example, if it is decided that the product sale service should cater for a new method of payment – “Internet cash” – this can be done by developing a new “Internet cash payment” service, and including it in the composition.

So far, this sounds to be little different from other software modularization techniques, from machine-code subroutines through to Java objects. Indeed, in an SOA that does not include messaging, service composition will be implemented by some such technique. But in many SOAs composition is implemented by services sending messages to invoke other services, and this technique gives much greater flexibility.

Two styles of composition are often distinguished:

- Orchestration, in which one of the services schedules and directs the others. If the above example was designed as an orchestration, there would be a direction service that would invoke in sequence the product selection, shopping cart review, payment method selection, and, depending on the selection result, credit card payment or invoice payment services.
- Choreography, in which the composed services interact and cooperate without the aid of a directing service. If the above example was designed as a choreography, there would be no directing service: the product selection service would invoke the shopping cart review service, the shopping cart review service would invoke the payment method selection service, and the payment method selection service would invoke the credit card payment or invoice payment service.

1.4.11 Service Discovery

When a program uses a software service, the identity of that service can be explicitly given in the program code. For example, where services are implemented as Java objects, their methods can be invoked by name by user programs. Where messaging is used, the destinations of the messages can be explicitly named at programming time. This is called *hard-wiring* of service connections.

Hard-wiring is a simple approach, but it has limitations. A different and much more flexible approach is service discovery. In this approach, the identity of the target service is not known at programming time, but is discovered at run time. The user program finds target services that meet its requirements, and chooses one of them.

The benefits of service discovery are:

- Ability to optimize performance, functionality, and cost – by selecting component services by these criteria
- Easier introduction of system upgrades – an upgraded service can be made available for selection in parallel with the one that it replaces, which can then be withdrawn

1.4.12 Asset Wrapping

The IT assets of an enterprise can often be considered as actors that perform services. A CPU performs an information processing service; a filestore performs an information storage service; and so on. This includes software as well as hardware assets. A database management system performs a database management service; an accounts package performs a financial information processing service.

An important feature of SOA is the recognition that these assets perform services, and the development of software façades that provide access to these assets and have interfaces that are in the same form as the interfaces to other software services of the enterprise. This is called *asset wrapping*. From a component-based software engineering point of view, the assets and the façade are components that are assembled to form a software service. The software services formed in this way can be used in service composition, have registry entries, and be dynamically discovered, in the same way as other services.

When an enterprise adopts an SOA, asset wrapping is typically applied to existing application software packages. This provides a significant benefit:

- Ability to integrate existing assets – which means that the value of an enterprise's existing assets is preserved, the cost of developing or acquiring replacements is avoided, and there is a smooth migration path from the old architecture to the new one

With the advent of SOA, some application vendors have begun to offer versions of their products in which the product capabilities are exposed as services. The acquisition of such a version is clearly a convenient way for an enterprise to achieve the “wrapping” of an application asset.

1.4.13 Virtualization

A façade can present to the consumer a virtual asset that does not correspond to the real underlying assets. This technique is called *virtualization*.

Virtualization can be used to enable programs that were written to use one asset to be executed with a different asset. For example, there are so-called “hypervisors” that can provide different operating system environments to programs running on a single CPU. But in the context of SOA it is more commonly used to create virtual assets that are functionally similar to the underlying assets. This can deliver two benefits:

- Improved reliability – through redundant operation of the underlying assets, so that one can take over when another fails or is withdrawn for maintenance
- Ability to scale operations to meet different demand levels – through dynamically increasing or reducing the number of underlying assets that support a real asset, as demand rises and falls

These benefits are particularly important when the principles of SOA are applied to enterprise infrastructure. While SOA is most commonly thought of as a way of architecting an enterprise’s application software, it can also be used at the infrastructure level, to create a Service-Oriented Infrastructure (SOI). Taken to the limit, this can provide a form of grid computing. The use of virtual assets that are made available over the Internet has become known as *cloud computing*.

1.4.14 Model-Driven Implementation

Model-driven implementation refers to the automatic realization of a system or application from an abstract model. Where the model starts at a high level of architectural abstraction, it is usually referred to as Model-Driven Architecture (MDA).

SOA lends itself particularly well to model-driven implementation, because it is based on a high-level software module concept (the service) for which there are good definition and interface standards.

Model-driven implementation provides:

- The ability to develop new functions rapidly – an important form of agility

In SOA, model-driven implementation can be applied to service compositions as well as to software services.

1.5 Maturity Model for SOA

As organizations move towards SOA and the use of services as the basis of their future IT architectures, they encounter the need to assess where they are in the migration path, and how to achieve greater benefits to support their businesses and systems.

This maturity model facilitates the assessment of an organization's current and desired future states in service integration and flexibility. It helps the organization to determine its architectural strategy for adopting service-orientation. It can be applied to a complete enterprise, or to an enterprise segment, such as one defined by one or more lines of business.

The model was provided by IBM as an input to work on The Open Group Services Integration Maturity Model (OSIMM).

1.5.1 Overview of the Model

The model defines seven dimensions across seven maturity levels.

The seven dimensions represent different views of the organization: the *Business, Organization, Methods, Application, Architecture, Information, and Infrastructure* views.

The seven maturity levels are *Silo, Integrated, Componentized, Services, Composite Services, Virtualized Services, and Dynamically Re-Configurable Services*.

The complete matrix of dimensions and levels is shown below.

1.5.2 Using the Model

You can use the generic model described here as a reference for the development of specific baseline and target models of an organization's SOA maturity. You can then perform a gap analysis to determine what is needed to move from the baseline to the target, and create a project roadmap for the transformation of the organization to the target SOA maturity level.








	Silo	Integrated	Componentized	Services	Composite Services	Virtualized Services	Dynamically Re-configurable Services
							
Business	Isolated Business Line-driven	Business Process Integration	Componentized Business	Componentized Business Offers Services	Processes through Service Composition	Geographical-independent Service Centers	Mix-and-match Business and Context-aware Capabilities
Organization	<i>Ad hoc</i> LOB IT Strategy & Governance	<i>Ad hoc</i> Enterprise IT Strategy & Governance	Common Governance Processes	Emerging SOA Governance	SOA and IT Governance Alignment	SOA and IT Infrastructure Governance Alignment	Governance through Policy
Methods	Structured Analysis & Design	Object-oriented Modeling	Component-based Development	Service-oriented Modeling	Service-oriented Modeling	Service-oriented Modeling for Infra (CDSP)	Business Grammar-oriented Modeling
Applications	Modules	Objects	Components	Services	Process Integration via Services	Process Integration via Services	Dynamic Assembly; Context-aware Invocation
Architecture	Monolithic Architecture	Layered Architecture	Component Architecture	Emerging SOA	SOA	Grid-enabled SOA	Dynamically Re-configurable Architecture
Information	Application-specific	LOB or Enterprise-specific	Canonical Models	Information as a Service	Enterprise Business Data Dictionary and Repository	Virtualized Data Services	Semantic Data Vocabularies
Infrastructure	LOB Platform-specific	Enterprise Standards	Common Re-usable Infrastructure	Project-based SOA Environment	Common SOA Environment	Virtual SOA Environment; S&R	Dynamic Sense, Decide & Respond
	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7

Figure 5: SOA Maturity Model Matrix

Note that this will provide a description of the target and a statement of the activities that need to be undertaken that are at a relatively high level. Further and more detailed analysis will be needed to produce the full target architecture description and implementation roadmap. In an architectural engagement, you would typically use the model at an early stage; for example, in the Preliminary Phase or Phase A of TOGAF. The full architecture and roadmap would not be produced until a later stage (for example, TOGAF Phases D and E).

You develop the specific baseline or target model by assessing the maturity of the organization in each of the seven dimensions. The assessment in each dimension results in a summary textual description and a number (1-7) identifying the maturity level. You can then aggregate the assessments through the dimensions to show the overall state of the organization.

You carry out the assessment by interviewing key staff from the organization, using a set of questions about the characteristics of the organization in each dimension. From the answers to these questions, you produce the summary maturity description and level identification. Architecture practices and consultancies will typically have standard sets of questions that they customize for each assessed organization to take account of particular circumstances and requirements.

1.5.3 Dimensions

An organization's SOA or desired SOA scope can be assessed across the following dimensions.

Business

The Business dimension is focused around the business architecture, the organization's current practices and policies around the business architecture, how business processes are designed, structured, implemented, and executed, how costs of IT capabilities are allocated throughout the enterprise, and how well the IT capabilities support flexibility of the business, business agility, and business Service-Level Agreement (SLA) management. Because the business dimension includes IT strategy, it includes a high-level quantifiable monetary-value justification for moving from one maturity level to a higher maturity level.

Organization

The Organization dimension is focused on the structuring and design of organizations and resulting measures of organizational effectiveness in the context of an SOA; for example, SOA governance. This includes the types and extent of skills, training, and education that are available within the organization, the existence of a formal governance process to keep IT activities and capabilities aligned with the needs of the whole business, how IT management is organized, and how costs are allocated.

Methods

The Methods dimension is focused on the methods and processes employed by the organization for its IT and business transformation, and the organization's maturity around the Software Development Lifecycle, such as the use of requirements management, estimation techniques, project

management, quality assurance processes, design methodologies and techniques, and tools for designing solutions.

Application

The Application dimension is focused on application style, structuring of the application and functional decomposition, re-usability, flexibility, reliability and extensibility of the applications, understanding and uniform use of best practices and patterns, whether multiple applications have been created to serve different lines of business with essentially the same functionality, and the availability of enterprise schema and object models.

Architecture

The Architecture dimension is focused on topology, data characteristics, business information model, integration techniques, enterprise architecture decisions, standards and policies, web services adoption level, experience in SOA implementation, SOA compliance criteria, and typical artifacts produced.

Information

The Information dimension is focused on the information modeling aspects, access to enterprise data, abstraction of the data access from the functional aspects, data transformation, service and process definition, handling of identifiers, security credentials, knowledge management, and content management.

Infrastructure

The Infrastructure dimension is focused on the organization's infrastructure capability, service management, IT operations, IT management and IT administration, how SLAs are met, how monitoring is performed, and what types of integration platforms are provided.

1.5.4 Maturity Levels

Each of the seven maturity levels reflects a possible abstract state of an organization in terms of its maturity in the integration of its services (business and/or IT). The following are typical aggregate descriptions for these levels.

Silo

Individual parts of the organization are developing their own software independently, with no integration of data, processes, standards, or technologies. This severely limits the ability of the organization to implement business processes that require co-operation between the different parts, and the IT systems cannot be integrated without significant manual intervention, such as re-keying and re-interpretation of data.

Integrated

Technologies have been put in place to communicate between the silos, and to integrate the data and interconnections. The construction of an IT system that integrates across different parts of the organization becomes possible. However, integration does not extend to common standards in data or business processes. Therefore, connecting two systems requires a possibly complex conversion of the data, operations, and protocols that they use. Each such connection may require bespoke code and adapters, leading to a proliferation of software that is difficult to manage and complex to code. It is not therefore easy to develop new business processes.

Componentized

The IT systems in the silos have been analyzed and broken down into component parts, with a framework in which they can be developed into new configurations and systems. There may also be some limited analysis of the business functionality into components. Although components interact through defined interfaces, they are not loosely-coupled, which limits interoperability between systems in different parts of the organization, or different organizations within the business eco-system. This makes it hard to develop cross-organization business processes.

Services

Composite applications can now be built from loosely-coupled business services. The way that services may be invoked is based upon open standards and independent of the underlying application technology. The services have an IT infrastructure that supports them with suitable protocols, security mechanisms, data transformation, and service management capabilities, even across different organizations within the eco-system, and may be managed by assigning responsibilities for SLAs to relevant parts of the organization. However, the flow of control within a composite application

is still defined by bespoke programming, rather than by a declarative flow language. The business functionality has been analyzed in detail and is broken down into business services residing within a business architecture that ensures that business services will interoperate at the business level. In addition, it is possible to define the services via a specification language that unambiguously describes the operations performed by each service, enabling the construction of a catalog of services. The combination of IT and business service architectures permits the construction of systems based upon these services, operating right across the organizations in the eco-system. However, at this stage the composition of services is still performed by developers writing bespoke code, thus limiting the agility of the development of new business processes.

Composite Services

It is now possible to construct a business process for a set of interacting services, not just by bespoke development, but by the use of a composition language to define the flow of information and control through the individual services. This permits the assembly of business services into composite business processes, which may be short-running or long-running, without significant construction of code. Thus, the design and development of business services is agile, and may be performed by developers under the close guidance of business analysts.

Virtualized Services

The business and IT services are now provided through a level of indirection. The service consumer does not invoke the service directly, but through a *virtual service*. The infrastructure performs the work of converting the virtual service invocation into an invocation of the real service, and may as part of this conversion change the address, network, protocol, data, and synchronization pattern of the invocation. Such conversions may be complex services in their own right; for example, transforming data from one data model to another. The virtual service thereby becomes more loosely-coupled from the infrastructure on which it is running, permitting more opportunities for the composition of business services. This is in contrast to the lower levels of service maturity where the service is more closely coupled to the infrastructure. Although virtualization has been used in non-SOA systems, this level extends the concept (and advantages) of virtualization to business services.

Dynamically Re-Configurable Services

Prior to this level, the business process assembly, although agile, is performed at design time by developers (under the guidance of business analysis and product managers) using suitable tooling. Now this assembly may be performed at run time, either assisted by the business analysts via suitable tooling, or by the system itself. This requires the ability to access a repository of services and to query this repository via the characteristics of the required services. In its simplest form, these characteristics may have been defined in advance, restricting the system to selecting and locating specific instances of services.

Chapter 2

The SOA Reference Architecture

This chapter describes the Service-Oriented Architecture (SOA) Reference Architecture, which is a significant underlying logical structure for the development and assessment of architectures designed and built using a combination of traditional and service-oriented computing principles and concepts. It contains the following sections:

- The Building Blocks of SOA (Section 2.1), which describes a set of architecture building blocks that represent the key elements of SOA
- A High-level Perspective of the SOA Reference Architecture (Section 2.2), which gives an overview of the nine layers of the reference architecture, with examples and rationale describing the main responsibilities of the layers and their primary building blocks
- Detailed Building Blocks of the SOA Reference Architecture (Section 2.3), which presents detailed models that show how some of the features of SOA can be implemented using the reference architecture
- Infrastructure for SOA (Section 2.4), which describes architecture building blocks that correspond to infrastructure products that are available today to support service-oriented applications

2.1 The Building Blocks of SOA

Architects develop models to show different aspects of the systems that are to be created or modified in accordance with their architectures. These models are constructed of *building blocks*, which are abstractions of the system components.

There are many different kinds of architecture and solution building block. You can abstract as a building block any part of a system that you want to think about. This section describes the building blocks that are commonly used for SOA. Some of them are also used for other styles of IT architecture.

They are described at a high level of abstraction; for example, *service* and *program*. An architectural model would typically show less abstract building

blocks – for example, *payment service* or *messaging program* – that fall into categories defined by the building blocks described here.

2.1.1 Services

Service is of course the most important SOA concept:

A *service* is a repeatable activity that has a specified outcome.

A service has a provider, can have one or more consumers, and produces effects that are of value to its consumers.

Providers and consumers see services from different points of view. To a consumer, a service is a black box. Two services are the same to a consumer if, given the same inputs, they produce the same effects. To a provider, a service is a means of exposing capabilities. Two services are different to a provider if they have different mechanisms for doing this, even though they produce the same effects. Architects talk to providers and to consumers, and must be able to see services from both points of view.

Consumers use services in their business processes (see Section 2.1.2). These business processes are mechanisms by which the consumers may themselves provide larger services. Such services, whether provided to consumers outside the enterprise or within it, are called *business services*.

From a provider's perspective, a service can be performed by people, by technology, or a combination of people using technology. Services that are performed by software programs (see Section 2.1.7) are the key components of an SOA. They are called *software services*. In any particular SOA, there is often a set of software services that are identified for processes such as discovery and composition. They make up the architecture's service portfolio, and they are called *portfolio services*.

2.1.2 Business Processes

A *business process* of an enterprise is an activity that is related to the enterprise's business mission and that is conducted in a defined, repeatable way.

People often take part in business processes. It is sometimes tempting to think of a business process as “a process in which people take part”, but it is

the relation to the business mission, rather than who or what takes part in it, that characterizes a business process.

Modeling business processes is an important element of enterprise architecture development. The Business Process Modeling Notation (BPMN) defined by the OMG is a formal notation for describing business processes that is becoming increasingly accepted.

When business processes are modeled, the people that take part in them are referred to as *human actors* (see Section 2.1.3).

Software programs can take part in business processes also. The software services of an SOA exist to support the enterprise's business processes. This relation can and should be symbiotic. Analysis of the business processes is the main way of identifying software services. On the other hand, the existence of the right software services enables new business processes to be developed, to meet new business opportunities.

2.1.3 Human Actors

An *actor* is someone or something that does something. Actors of various kinds can be abstracted as building blocks.

A *human actor* is a person that does something in relation to an architected system.

Human actors appear when business processes are modeled, and also in models showing other aspects of the system, such as management and security.

A model generally shows an abstraction of a person, rather than a real person: "president", rather than "George Washington". The word "role" is sometimes used, rather than "actor", because of this.

Other kinds of actor may be encountered in IT architecture. A *technology actor* is an actor that is a machine or other piece of technology. It could be hardware, software, or both. A program is one particular kind of technology actor; a data store (see Section 2.1.8) is another. An *organization actor* is a

system whose components can be people, technology items, and other things, and that is regarded as a single actor.

2.1.4 Events

Events to which business processes or services respond can be building blocks too.

An *event* is something that happens.

An event is not necessarily associated with any particular business process or service.

Like human actors, events appear when business processes are modeled, and also in models showing other aspects of the system, such as management and security.

2.1.5 Service Descriptions, Contracts, and Policies

An important feature of services in SOA is that they have descriptions that state clearly what they do and how to interact with them.

A *description* is an information item (see Section 2.1.8) that is represented in words, possibly accompanied by supporting material such as graphics.

A *service description* is a description of a service.

A service description can be represented in informal text but, for many purposes, it is better to use a structured description language. The Web Services Description Language (WSDL) defined by the W3C is widely used for this purpose.

A *contract* is an agreement between two or more actors – the parties to the contract. The term is most commonly used for a written agreement, or one that is enforceable at law, but it can be applied more widely.

A *service contract* is a contract between the provider of a service and one or more of its consumers.

A service contract may be an implicit agreement that the service will conform to its description, or it may be a more formal agreement, which could be recorded in a signed internal enterprise document, or be a legal contract executed between enterprises.

A service contract covers functionality (what effects the service produces), and often also covers service quality. Service quality typically includes the service's performance and security characteristics.

A *policy* is a course of action that a person or organization intends to follow, or intends that another actor should follow.

A *service policy* is a course of action that a service provider intends to follow in providing a service, or intends that the service consumers should follow.

For example, an enterprise might have a policy that certain services are provided only to internal consumers, or a web service provider might have an "acceptable use policy" that states what consumers of its service may and may not do.

Service description, contract, and policy building blocks appear in models that show how services are consumed. In SOA, this is a very important aspect of system implementation and operation.

2.1.6 Service Compositions

A *composition* is a collection of things that are put together to form a single thing.

Services can be composed of other services. Business processes can be composed of services and other business processes.

A *service composition* is a collection of services that are put together to form a single service.

Service composition is a provider's concept. It relates not to what a service does, but to how it is performed.

Service compositions appear in models that show how business processes are supported by services. They may also appear where functionality that is not related to the business mission is implemented using a service-oriented approach, in a model of an SOI, for example.

2.1.7 Programs

A software *program* is a set of instructions for a computer to perform a specific task.

Software programs relevant to SOA include programs that contain instructions to perform services. When such a program is executing, we generally say that it (rather than the computer that carries out its instructions) performs the service.

A software service is different from the program that performs it, just as services performed by people are different from those people.

A program that performs a software service should also be distinguished from the service provider. The service provider is the person or organization that takes responsibility for the service, and enters into contracts with its consumers.

Other programs relevant to SOA include programs that provide infrastructure for the services, and application programs.

Program building blocks appear in models that show how services are performed, models that show how services are integrated with each other and with other system components, models that show how the architected system processes data, and in models showing other system aspects, such as performance, management, security, and governance.

2.1.8 Information Items, Data Items, and Data Stores

IT architecture is of course very much concerned with information, and building blocks related to information are used for SOA, as for other architectural styles.

The concept of information is a very general one.

An *information item* is a thing that is known about some other thing.

An information item may be simple or may be complex, composed of simpler information items.

Information items appear in models of business processes.

Data can be defined as “a re-interpretable representation of information in a formalized manner suitable for communication, interpretation, or processing” (see ISO/IEC 2382-1:1993.)

A *data item* is a representation of an information item.

A *data store* is a technology actor that stores data items.

Data items and data stores are important architectural building blocks. Like programs, they appear in models that show how services are performed, models that show how services are integrated with each other and with other system components, models that show how the architected system processes data, and in models showing other system aspects, such as performance, management, security, and governance.

2.2 A High-Level Perspective of the SOA Reference Architecture

This high-level perspective shows the conceptual building blocks of an SOA solution, and how they relate to each other. It can be used as a basis for specific solution models, and also for models of larger SOA systems, including those of enterprise SOA.

The perspective is derived from “*The SOA Solution Stack: A Reference Architecture for Designing SOA Solutions*” submitted by IBM as input to the Reference Architecture project of The Open Group SOA Working Group. The original reference architecture is described in an IBM developerWorks article.

2.2.1 Overview

The reference architecture classifies SOA-related building blocks into nine layers, as shown below.

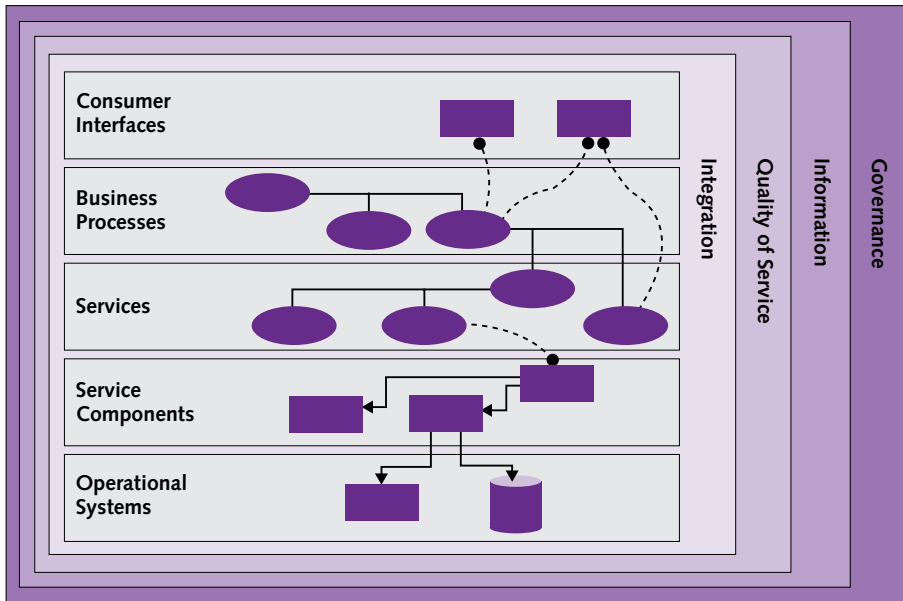


Figure 6: High-Level Perspective of the SOA Reference Architecture

The first five layers contain building blocks whose purposes relate to business functionality. They support each other in a hierarchy, although its layering is not strict. In order, from bottom to top, they contain building blocks that are:

- Existing application assets and other programs (the Operational Systems layer)
- Software components that help to perform services and may leverage existing assets (the Service Components layer)
- The services that are in the service portfolio, and hence available for use in solutions, including through discovery and composition (the Services layer)
- Business processes, and service compositions that they use, including orchestrations and choreographies (the Business Processes layer)
- The people and external systems that participate in the business processes, and their interfaces to the services (the Consumers layer)

The remaining four layers support the layers related to business functionality, but do not support each other in a strictly layered hierarchy. They contain building blocks whose purposes relate to:

- Integration of other building blocks (the Integration layer)
- Quality aspects of system operation (the Quality of Service layer)

- Information (the Information layer)
- Governance (the Governance layer)

2.2.2 Operational Systems Layer

The building blocks in this layer are programs and data of the operational systems of the enterprise. For example, a bank might have building blocks such as “customer relations management”, “customer database”, “internal accounting”, and “settlement” in this layer. They include:

- Applications and data stores with functionality required to deliver the service functionality in the Services layer
- Infrastructure programs such as operating systems, database management systems, and run-time environments

2.2.3 Service Components Layer

This layer contains programs, other than the programs in the Operational Systems layer, that help to perform services.

The asset wrapping and virtualization features of SOA are supported by building blocks in this layer.

The building blocks in this layer include:

- Programs that “wrap” the programs in the Operational Systems layer to create services
- Programs that are written to perform services and deliver the service functionality themselves
- Groups of such programs

The Service Components layer enables IT flexibility by strengthening decoupling in the system. Decoupling is achieved by hiding volatile implementation details from consumers.

As well as insulating consumers from the service implementation, a service component provides a point at which compliance with the service contract can be monitored or enforced.

2.2.4 Services Layer

This layer contains the portfolio services. Each service conforms to a specification that provides sufficient detail to enable a consumer to invoke the functions exposed by the service provider.

This is the central layer of the model. Its building blocks support the basic service feature of SOA.

Examples of portfolio services, in a banking context, might be “identify eligible customer account”, “validate transfer”, “submit transfer”, “move funds”, and “complete transfer”.

Portfolio services can be composed of other portfolio services. For example, “move funds” might be composed of other portfolio services including “move funds from source” and “move funds into destination”.

In an enterprise architecture development, as opposed to a solution design, you are likely to be dealing with groups of related services rather than individual services. An example of such a group might be “funds transfer services”.

The building blocks in this layer include:

- The portfolio services themselves
- Compositions in which portfolio services are composed of other portfolio services
- Groups of services and compositions covering functional areas
- Data created or used by the portfolio services
- Service descriptions, contracts, and policies

2.2.5 Business Processes Layer

This layer contains the business processes. An example might be “transfer funds”.

Business processes can be composed of other business processes and of portfolio services. For example, “transfer funds” might be composed of other business processes including “create transfer” and “process transfer”.

The lowest layer typically includes business processes that are composed of portfolio services. For example, “create transfer” might be composed of portfolio services including “submit transfer” and “move funds”.

The building blocks in this layer include:

- The business processes themselves
- Compositions in which business processes are composed of other business processes and of portfolio services
- Information created or used by the business processes

2.2.6 Consumers Layer

This layer contains the users of the system and the programs by which they interface to the portfolio services. Examples might be “customer” and “online banking portal”.

Building blocks in this layer include:

- People, organizations, and programs that take part in the business processes (the consumers)
- Interface programs that present information to and accept information from the consumers, such as channels, portals, other human-computer interface programs, format converters, and interface configuration programs
- Data used by the interface programs, such as user profiles and interface configurations

2.2.7 Integration Layer

This layer contains building blocks whose function is to enable integration of and communication between other building blocks. It gives the ability to decouple service providers and consumers, which adds flexibility to the architecture.

The messaging, message transformation, complex event processing, service composition, and service discovery features of SOA are supported by building blocks in this layer.

2.2.8 Quality of Service Layer

This layer contains building blocks whose functions are concerned with monitoring and management of the quality of service of the architected system, including its performance, security, and manageability.

The message monitoring, message control, and message security features of SOA are supported by building blocks in this layer.

The layer also includes building blocks such as performance managers, security managers, and configuration managers.

2.2.9 Information Layer

This layer contains building blocks whose functions are concerned with the transformation and management of data.

The message transformation feature of SOA is supported by building blocks in this layer.

The layer also includes building blocks such as:

- Information models
- Vocabularies
- Data models
- Data representation models
- Programs that expose data as services
- Data search engines
- Data mining engines
- Document management systems

2.2.10 Governance Layer

This layer contains building blocks whose function is concerned with implementation and operational governance.

The layer includes building blocks such as:

- Governance rules and procedures
- Services and programs that support the application of the rules and the operation of the procedures

2.3 Detailed Building Blocks of the SOA Reference Architecture

This section contains detailed models showing how some of the features of SOA can be implemented using the SOA Reference Architecture.

The building blocks shown in these models fall into the categories described under Building Blocks of SOA (Section 2.1), but are at a lower level of abstraction. Some of them are described further under Infrastructure for SOA (Section 2.4). They are all building blocks of the SOA Reference Architecture.

2.3.1 Composition

This is the basic model for service composition.

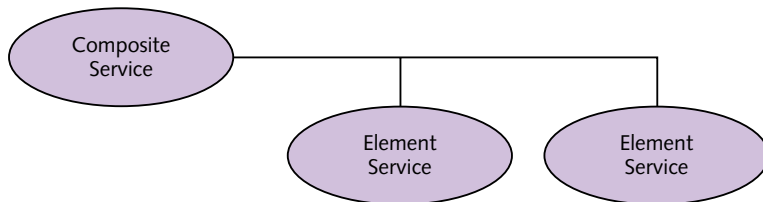


Figure 7: Basic Model for Service Composition

The composite service is created by putting together the element services. This may simply be a matter of performing the element services one after another, or there may be more complex interactions between the services that affect the order in which they are performed.

In this latter case, the performance of the services may be controlled by *scripts*. The model for scripted service composition is shown below.

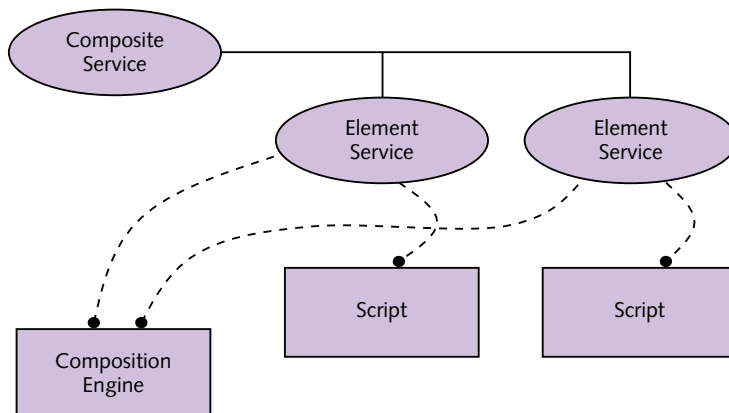


Figure 8: Model for Scripted Service Composition

In this model, one or more of the element services is controlled by a script, executed by a composition engine (see Section 2.4.8). The script is written in a language (such as the Web Services Business Process Execution Language (BPEL) defined by OASIS) that includes constructs for interacting with other services.

(This interaction is often via messages, and composition engines are often integrated with messaging programs.)

In a common orchestration scenario, only one of the element services is controlled by a script. This service directs the other services by invoking them in the appropriate sequence.

In a common choreography scenario, all of the element services are controlled by scripts, and these scripts are written so as to synchronize the performance of the element services.

2.3.2 Messaging

This is the basic messaging model.

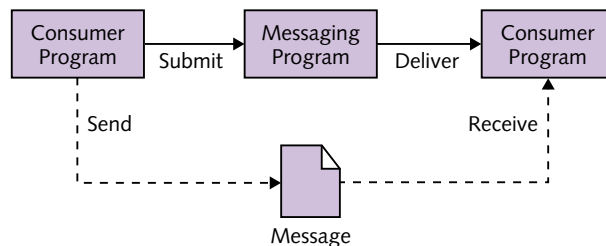


Figure 9: Basic Messaging Model

Programs can exchange messages using a messaging service that is performed by a messaging program. One program submits a message to the messaging program, which delivers the message to a second program. The message is sent by the first program, and received by the second. These programs are both consumers of the messaging service.

The consumer programs that send and receive the messages can be programs that perform services. In this case, we also say that the services are sending and receiving the messages.

Use of a single program for communication between services makes it easy to process the information as it is exchanged, by adding other programs onto the messaging program. Several SOA features, such as message monitoring and data translation, are commonly implemented in this way.

A detailed messaging model, including several such programs, is shown below.

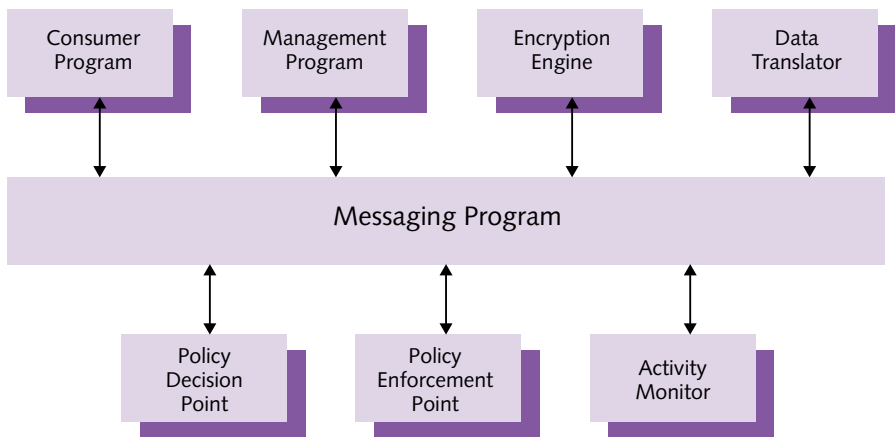


Figure 10: Detailed Messaging Model

The *consumer programs* are as in the Basic Messaging Model (Figure 9).

The *management programs* perform functions such as setting up and changing routing tables, configuring the infrastructure building blocks that are attached to the bus, and managing logs and audit trails.

The programs other than the consumer programs and the management programs combine to perform the messaging service. They all realize infrastructure building blocks:

- An *encryption engine* (see Section 2.4.6) encrypts and decrypts data, and applies and verifies integrity checks based on encryption technology.
- A *data translator* (see Section 2.4.5) converts data from one representation to another.
- A *policy decision point* (see Section 2.4.4) decides, in accordance with a policy, whether resource access requests should be granted.

- A *policy enforcement point* (see Section 2.4.4) implements decisions to grant or deny resource access requests.
- An *activity monitor* (see Section 2.4.3) monitors messaging activity.

The programs shown in the Detailed Messaging Model (Figure 10) support a rich set of capabilities. Not all of these capabilities are required for a viable messaging service, and not all of these programs need be present.

2.3.3 Service Discovery

This is the model for service discovery.

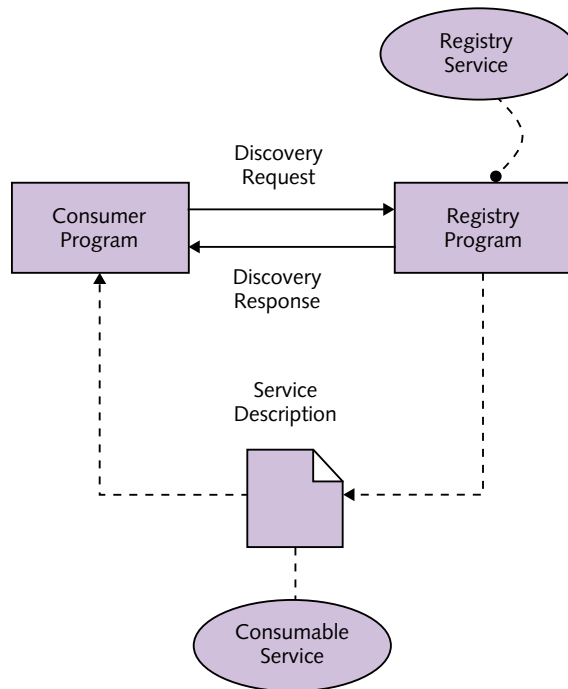


Figure 11: Model for Service Discovery

A program (the *consumer program*) wishes to use a service. It makes a discovery request to a program that performs a *registry service* (see Section 2.4.9), specifying the characteristics of the service that it wishes to use. The *registry program* replies with a *discovery response*, giving descriptions of services that meet the specification (the *consumable services*). The consumer program can then use one (or more) of the consumable services.

The consumer program can be a program that performs a service. It is a consumer of the registry service, and is also a consumer of whichever of the consumable services it uses.

2.3.4 Asset Wrapping

This is the model for asset wrapping.

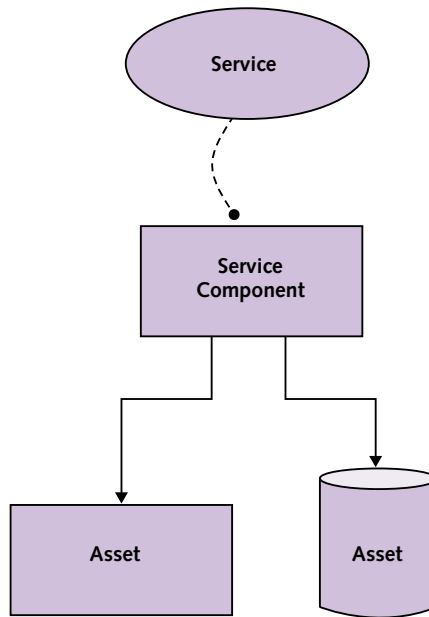


Figure 12: Model for Asset Wrapping

The *service component* (see Section 1.4.10) in this model is a program that implements one or more of the interfaces of a service and invokes existing assets in order to deliver the required functionality. It need not itself have much functionality, and is sometimes described as a façade. The service is performed by its service components and the assets that they invoke.

2.3.5 Virtualization

This is the model for virtualization.

This model can be used to define the internal structure of a service component in order to create a virtualized service.

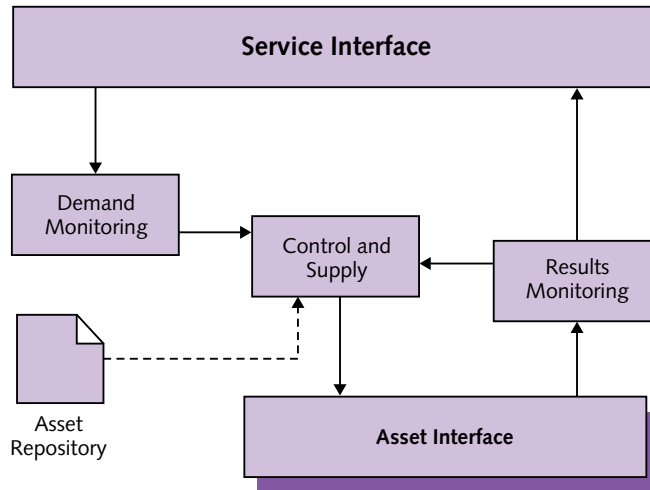


Figure 13: Model for Virtualization

The *service interface* program implements the interface that is used by consumers of the service, in accordance with the service contract. The *asset interface* programs invoke the underlying assets whose capabilities are exposed by the service. The *control and supply* program invokes the underlying assets via the asset interface programs, using appropriate numbers of assets with regard to the level of demand and the ability of currently-used assets to respond. Details of available assets are held in the *asset repository*, which is maintained as assets are commissioned and decommissioned to show which are in service. The *demand monitoring* program monitors the level of demand for the service, and the *results monitoring* program assesses the response.

2.3.6 Event Processing

This is the model for event processing.

The *event processor* (see Section 2.3.6) detects *input events* with associated information, and generates *output events* with associated information. A single output event may correspond to multiple input events, and the information associated with an output event is derived from all the information associated with the corresponding input events.

A common scenario is that the input events are messages on a message bus, and the output events are handled by analysis and presentation programs within an activity monitor.

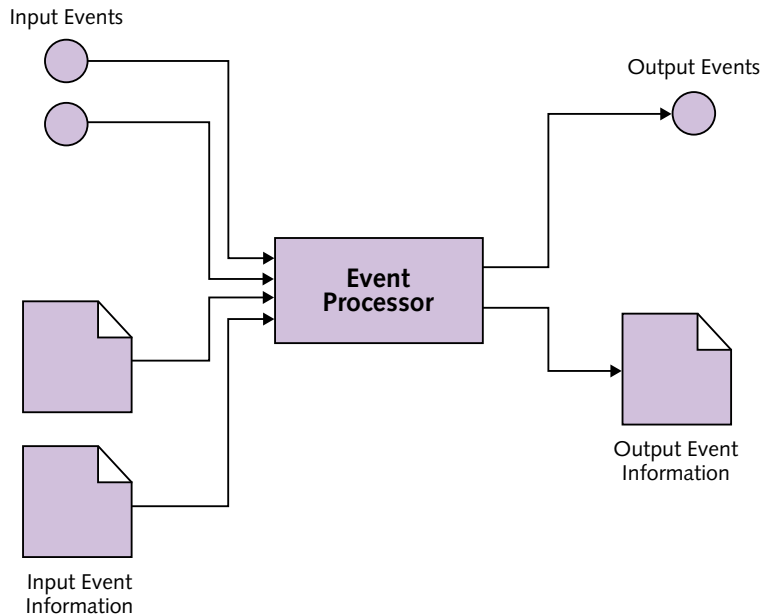


Figure 14: Model for Event Processing

2.4 Infrastructure for SOA

The infrastructure for SOA is implemented within the context of the SOA Reference Architecture. SOA places unique requirements on enterprise IT infrastructure. Good choice of infrastructure products, and their effective integration, is crucial for SOA implementation.

An effective architecture makes this possible:

- The architecture should have building blocks to which vendors can match their products, so that products can be compared like-for-like.
- The building blocks should be defined by open standards for interoperability.
- The architecture should be expressed in terms of a standard reference architecture, so that it can easily be communicated to developers and integrators.

This section of the SOA Reference Architecture will help you to achieve these goals. It describes a number of SOA infrastructure building blocks that correspond to products that are available today. It identifies the key standards to which those products can conform. And it is based on the conceptual building blocks of SOA (Section 2.1), the high-level SOA model (Section

2.2), and the detailed models for SOA features (Section 2.3) described in this Source Book.

2.4.1 Service Repository

Re-use of software services is a very important aspect of SOA for many enterprises. They often introduce governance mechanisms to ensure that services are developed with re-use in mind, and that the possibility of re-using existing services is explored before new ones are written. But, for re-use to be possible at all, the services must be clearly described, and their descriptions must be readily available to developers.

The simplest way of making service descriptions available is to publish them as documents. A more sophisticated mechanism, which has advantages where there are large numbers of services to keep track of, is a *service repository*, which enables you to maintain and search a database of service descriptions.

In a large enterprise you may have a very large number of services, and you will need a common vocabulary and data model, as well as some sort of knowledge management system, as the basis for your service repository.

Registries used in service discovery (see Section 1.4.11) can be used as service repositories. There are a number of specialist service registry products that are commercially available, but they tend not to provide good support for searches conducted by people, and are useful at run time, rather than at design time.

The WSDL defined by the W3C is a commonly-accepted standard for service descriptions. There is no commonly-accepted standard for repository management and search operations.

2.4.2 Messaging Program

Between enterprises, services typically exchange messages via the web. Within an enterprise, they are often exchanged using an Enterprise Services Bus (ESB). The term ESB can mean many different things; the term *messaging program* used here describes a program that has essential ESB functionality. Commercial ESB products should include this functionality, but often also

have additional features, such as those of other building blocks of the detailed messaging model that are described later in this section.

A messaging program transports messages between other programs. It performs the messaging service shown in the Basic Messaging Model (Figure 9), and may interface to management programs, encryption engines, data translators, policy decision points, policy enforcement points, and activity monitors as described in the Detailed Messaging Model (Figure 10).

A messaging program must support the following functions:

- Acceptance of submitted messages
- Message delivery
- Message routing

A messaging program may optionally support the following additional functions:

- Routing configuration
- Configuration of attached encryption engines, data translators, policy decision points, policy enforcement points, and activity monitors
- Logging
- Auditing
- Assurance of delivery
- Protocol transformation (between different message submission and delivery interface formats)

A messaging program has one principal interface: message submission and delivery. The Simple Object Access Protocol (SOAP) defined by the W3C is a commonly-accepted standard for this. In use, SOAP is layered on top of more basic communication protocols, such as the Hypertext Transfer Protocol (HTTP).

SOAP is the core message transmission standard. W3C also defines related standards for addressing, resource representation, and message transmission optimization.

SOAP and its related standards were defined in the first instance for web services. They are also supported, at least to some extent, by many messaging program products.

SOAP does not provide assurance of message delivery. Another protocol, Web Services Reliable Messaging (WS-ReliableMessaging) defined by OASIS, includes mechanisms that enable messages to be transferred reliably in the presence of software component, system, or network failures. Its specification includes a SOAP binding, so that it can be layered on top of SOAP.

SOAP is a standard that enables interoperability between messaging systems. For portability across messaging systems, software services should use a standard interface dependent on the programming language, such as the Java Messaging Service (JMS).

The Web Services Security (WS-Security) standards defined by OASIS are commonly-accepted standards that provide for message encryption and encapsulation of security tokens. A messaging program that interfaces to an encryption engine may support WS-Security for encryption, and a message bus that interfaces to policy decision points and policy enforcement points may support WS-Security for security tokens.

A message bus that interfaces to management programs, encryption engines, data translators, policy decision points, policy enforcement points, or activity monitors can communicate with them using the same interface as for message submission and delivery, but is more likely to use different interfaces that are designed to support the particular programs concerned.

2.4.3 Activity Monitor

An *activity monitor* is a program that interfaces to a messaging program as described in the Detailed Messaging Model (Figure 10), and monitors the messaging activity.

An activity monitor has three principal interfaces:

- Message bus
- Results
- Configuration

SOAP is a commonly-accepted standard for the interface to the message bus.

There are no commonly-accepted standards for output of the results of the monitoring, or for configuration of the monitor.

2.4.4 PDPs and PEPs

A *policy decision point* (PDP) is a program that decides, in accordance with a policy, whether resource access requests should be granted.

A *policy enforcement point* (PEP) is a program that implements decisions to grant or deny resource access requests.

PDPs and PEPs co-operate as part of an overall policy framework. Such a framework can be used to enforce policies of various kinds, including security and performance management.

PDPs and PEPs can be attached to messaging programs, as shown in the Detailed Messaging Model (Figure 10). When used in this way, they can control acceptance of messages for transport over the bus, and can control delivery of messages to services.

PEPs can also be embedded within the services themselves.

There are a number of standards that are relevant to PDPs and PEPs, including the Web Services Policy Framework (WS-Policy) defined by the W3C, and the following standards defined by OASIS: the Web Services Security Policy (WS-Security-Policy), the eXtensible Access Control Markup Language (XACML), and the Security Assertion Markup Language (SAML).

2.4.5 Data Translator

A *data translator* is a program that converts data from one representation to another. It can interface to a message bus, as shown in the Detailed Messaging Model (Figure 10).

A data translator has three principal interfaces:

- Input data format
- Output data format
- Translation specification

The eXtensible Markup Language (XML) defined by the W3C is a commonly-accepted generic standard for the input and output data formats. Document Type Definitions (DTDs) and XML schemas are used to define specific XML data formats.

The eXtensible Stylesheet Language (XSL) Transformations (XSLT) standard defined by the W3C is a commonly-accepted standard for specifying translations between data with different XML DTDs or schemas. However, it is not sufficiently powerful to satisfy all significant translation requirements.

Where the data translator interfaces to a message bus, SOAP (with XML layered on top) is a commonly-accepted standard for transport of the input and output data.

2.4.6 Encryption Engine

An *encryption engine* is a program that encrypts and decrypts data, and applies and verifies integrity checks based on encryption technology.

As shown in the Detailed Messaging Model (Figure 10), an encryption engine can be connected to a messaging program. In this situation it encrypts or applies integrity checks to messages, either for transmission over the bus or for transmission from the bus over the Internet, and decrypts or verifies the integrity of messages prior to delivery to services using the bus.

An encryption engine has five principal interfaces:

- For submission of data for encryption, and return of the encrypted data
- For submission of data for decryption, and return of the decrypted data
- For application of integrity checks
- For verification of integrity checks
- For configuration and management (including encryption key management)

WS-Security includes extensions to SOAP that provide message integrity and confidentiality. They also provide the ability to send security tokens in SOAP messages, and this feature enables higher-level security mechanisms.

Where an encryption engine is connected to a message bus, or sends and receives data via the Internet, WS-Security is a commonly-accepted standard for submission of data for encryption and decryption, and for application and verification of integrity checks. However, product-specific interfaces are often used instead.

There is no commonly-accepted standard for the configuration and management interface.

2.4.7 Event Processor

An *event processor* is a program that processes input events and combines them to generate output events, as described in the Model for Event Processing (Figure 14).

An event processor has three principal interfaces:

- Input event
- Output event
- Configuration

Input events and output events can be SOAP messages. SOAP is, however, not necessarily an appropriate standard for all event interfaces.

There is no commonly-accepted standard for the configuration interface.

2.4.8 Composition Engine

A *composition engine* is a program that executes scripts that control services and describe interactions between them. It is used to perform one or more of the services in a composition, as described in the Model for Scripted Service Composition (Figure 8).

A composition engine has two principal interfaces:

- With the scripts
- With the services with which it interacts

The Web Services BPEL defined by OASIS is a widely-accepted standard language for describing service compositions. It is based on XML and integrated with WSDL. It can be used by an enterprise to define compositions of internal and external services. The composed services can include both short-term and long-running transactions.

SOAP is a commonly-accepted standard for the interface between a composition engine and the services with which it interacts.

2.4.9 Service Registry

A *service registry* is an organized collection of service descriptions, maintained by a *registry service* that returns the descriptions that match specifications in submitted enquiries, as described in the Model for Service Discovery (Figure 11).

A registry service has two principle interfaces:

- For management of the service descriptions that it maintains
- For enquiries and responses

The Universal Description Discovery and Integration (UDDI) standards published by OASIS are the most commonly-accepted standards for both of these interfaces. They apply to web-based registries that expose information about businesses or other entities.

The original UDDI concept was of public registries for businesses and services, and several well-known companies provided public UDDI nodes. However, in January 2006 those companies announced the closure of their nodes. UDDI is now mostly used as an internal registry standard within corporations.

The UDDI standards define interfaces, based on SOAP and XML, for publication and inquiry of registry contents. Publication and inquiry are services, and the UDDI standards include descriptions of these services in WSDL. Also, it is possible to map UDDI service descriptions to WSDL service descriptions. This means that UDDI can be used together with WSDL in an SOA.

A significant limitation of UDDI, which has made service discovery less useful than had been hoped, is the difficulty of matching service descriptions to consumer needs. A person reading a “Yellow Pages” directory understands that a building contractor may provide roofing services, and looks under “Building Contractors” as well as under “Roofing Services” when a roof needs repair. Software programs cannot yet make this kind of connection in a general way. There is, however, another possible approach, which is not yet established, but which promises better search capabilities. This is the use of the Web Ontology Language (OWL) defined by the W3C for service descriptions published as part of the Semantic Web. A group of researchers is developing an OWL web service ontology known as OWL-S, which includes

a core set of mark-up language constructs for describing the properties and capabilities of web services, and is designed to be used with WSDL.

2.4.10 Service Components

A *service component* is a program that plays a part in performing a service. Service components include programs that implement service interfaces using underlying assets, as in the Model for Asset Wrapping (Figure 12).

These service components often employ container-based technologies, such as Enterprise JavaBeans (EJB) technology.

Service components are a limiting influence on performance. The technology selected as the basis for service components must be carefully considered from this point of view.

Service component building blocks that implement service interfaces must conform to the same interface standards as the services. In an SOA that uses messaging, these typically include WSDL and SOAP.

2.4.11 Model-Implementation Environment

A *model-implementation environment* is a set of programs that support the creation of models for services or service compositions, and the implementation of programs and scripts that perform the services and compositions directly from the models, without substantial human intervention.

A model-implementation environment has two principal interfaces:

- Model-description language
- Implementation meta-model

The Unified Modeling Language (UML) defined by the OMG is a standard modeling language that is widely used by the architecture community. The OMG also defines a Meta Object Facility (MOF) which provides an object-oriented abstract modeling framework that comprehends UML and is the basis of its model-driven approach.

OWL is also sometimes used for describing models of systems and applications, instead of, or together with, the OMG standards.

