# SERVICE-ORIENTED ARCHITECTURE: KEYS TO SUCCESSFUL ADOPTION AND IMPLEMENTATION

**Qusay F. Hassan**

**Faculty of Computers and Information Systems, Mansoura University, Mansoura, Egypt**

***ABSTRACT*:** Since the emergence of the SOA model, many organizations –both public and private– sought out to adopt it in order to acquire the desired agility level they seek to efficiently meet their needs. However, SOA adoption is not a trivial task given the number of challenges that should be overcome in order to reap the inherent benefits. This paper addresses the top challenges that might prevent adopters from achieving a successful implementation of SOA in their organizations.

***KEYWORDS:*** SOA, SOA Adoption Challenges and Solutions.

## INTRODUCTION

SOA is a promising paradigm that presents a number of benefits to its adopters including [Has09, Kob05]:

- **Reusability:** Technical components and business functionalities are abstracted, after removing redundancies and inconsistencies, in a form that allows them to be used again and again by different systems and business units.
- **Data Sharing:** Underlying data could be shared between different systems, by wrapping data sources with joint data service.
- **Location/Platform Independence:** A greater interoperability is enabled between different systems and business partners. This is achieved by allowing access to services regardless of their physical locations or used platforms.
- **Business Alignment:** Since the "service" is originally a business term, SOA enables a better alignment between IT and business professionals.

The aforementioned benefits would result in shorter time-to-market, fewer bugs, increased productivity, and cost savings when building, integrating, or maintaining software systems. However, the ability to realize these benefits strongly depends on properly addressing the challenges that SOA adopters might face. These challenges range from an inconsistent prior understanding of SOA through implementation obstacles to incomplete management strategies. Lacking awareness of these challenges could place SOA implementations at risk and could lead to complete implementation failure among its adopters. In the subsequent sections bellow, the paper will discuss the key adoption challenges and introduce some approaches that organizations can follow to ensure a successful SOA implementation.

## 1. UNDERSTANDING

The meaning of SOA is interpreted differently from one person to another [***07a]:

- **A product:** Organizations usually think of SOA as a product that can be purchased from software vendors. This belief stems from the fact that labels of SOA are placed on the software vendors' websites and in their brochures.
- **A synonym to XML Web Services:** Web Services is simply a modern technology that allows access to remote objects. Conversely, SOA is a design methodology that aims to build systems in a way that makes them easy to reuse and integrate. People unknowingly use the two terms interchangeably because most SOA-based implementations are built with Web Services.
- **A goal:** SOA is not a goal in and of itself. Rather, it is a way to build software in terms of abstract, callable components known as services. These services can be flexibly used/re-used whenever needed.

SOA in simple terms is a design model that tends to package existing or new functionalities as a collection of accessible services. These services can communicate with each other to pass information, and/or coordinate business workflows while abstracting technical details.

A service is a well-defined, self-contained block that is composed of a set of operations and components, built in a way that lets them be dynamically integrated to cover technical and/or business needs. A service can perform a simple functionality as that of calculating a loan interest rate, or a complete business workflow such as granting a loan.

Each service has three constituent elements:

- **Contract:** Provides informal specifications of the purpose, functionality, constraints and usage of the service. It also contains a formal definition based on a description language such as IDL or WSDL that provides information about the programing

language, middleware, network protocols, and runtime environment.

- **Interface:** Acts as a stub/proxy class that exposes available operations.
- **Implementation:** Contains the physical implementation of the service logic which may be encapsulated internally within the service itself or provided by external components. The separation between the service interface and the implementation offers a "clean" model that enables designers/developers to change the underlying logic without affecting the callers.

Work in SOA is done collaboratively by three distinct parties [Pap03]:

- Provider: Creates, publishes, and maintains services.
- Broker: Enables access to the registered services. The service registration process refers to publishing services (either inside the organization or externally via the Internet) with information that enables clients to discover and bind to them.

Client: Looks for services that meet its specifications, and follows the instructions to test and use them. The interaction between clients and services normally deploy the request/reply model, where a service receive requests from clients and forward back the returned results.

Different services can be combined together to form a composite service, again while hiding underlying complexities. However, in real SOA models, each service must not depend on the state of other services. That is, calls between individual services must not be embedded inside them, but rather, the composition and "glue" logic must remain outside of the services. Orchestration tools or a custom code can be used to associate separate services together either in new composite services or directly in client applications.

It is clear that this perspective is quite different from traditional methods that depend on creating a myriad of (mostly redundant) software systems for business silos. These systems are usually created to meet today's needs without considering unforeseen requirements [KBS04]. Modifying (or the integrating with) these systems is a burden that IT executives face in organizations, making them less agile in meeting the business needs. For instance, a legacy system is usually composed of statefull, tightly-coupled objects that do not separate between operations, interfaces, and business processes leading to spaghetti architectures that are hard to understand and time-consuming to modify. Conversely, leveraging SOA to turn software systems into a set of tested infrastructure common services, that would be built upon to meet business fluctuations, could promote organizational agility and performance [RH08].

In spite of what preceded, SOA is not a panacea that can solve all the organization's problems. Hence, adopters should know exactly why they want to migrate to SOA and what to expect from that switch. This is crucial because in many situations it might be considered a wrong move. Some of the cases which SOA might not be suitable for include [L+05]:

- Organizations that cannot afford to abandon their investments in existing applications and systems to redevelop them from scratch as services.
- The ability to convert the non-SOA based systems to SOA-based ones is one of the core benefits of the model; however, the technical specifications of these systems such as age, architecture, technologies, and documentation, can complicate this process.
- Lack of information about existing systems.
- High costs, risks, and efforts linked with the migration process.
- A big gap might exist between the state of the existing systems and the future state of the targeted services.
- Lack of a clear vision and a migration strategy.

In such cases, leveraging SOA may not be the way to go and may lead to some undesired results. Thus, knowing exactly why, when and how to use SOA is critical for a successful adoption.

Organizations can decide whether to adopt SOA or not by reading about others' prior experience. This would provide initiatives with the necessary details about the conditions for using SOA as well as a deeper understanding of distinct implementation challenges, obstacles and best solutions. After that, initiatives can start the adoption journey by migrating a small, non-critical application. Following a "think big, start small" approach will give adopters the chance to better understand the value of SOA while minimizing the risks.

Another helpful recommendation is to plan an adoption roadmap. This roadmap should involve: understanding the business domain; gathering information about current processes, applications, integrations, security, data and governance; and reducing the SOA gap readiness [B+05]. Such roadmap might also include an impact analysis to forecast the extent of change to the existing resources affected by SOA, transition plans, and approximate estimates to the future growth of services.

## 2. KNOWLEDGE

Most software specialists are knowledgeable about traditional terms and paradigms, such as functions, classes, modules, libraries/components, and Object-Oriented Programming. This knowledge allows software specialists to accomplish their regular tasks efficiently. However, these skills alone are not enough

when dealing with SOA since it comes with its own terms and methodologies.
To enable people to professionally deal with SOA, they should first learn about the basics. This should include the aspects of SOA lifecycle which includes analysis, modeling, design, implementation and testing [T+**]. Table 1 briefly lists the main differences between these aspects in SOA and traditional paradigms.

**Table 1. Traditional paradigm vs. SOA paradigm**

| | Traditional Paradigm | SOA Paradigm |
|---|---|---|
| Analysis | • Requirements are given in a natural language to system analysts who convert them to technical specifications in order to enable developers/designers to understand them. | • Domain analysis is solely done by providers, enabling application builders to focus only on finding and combining services that meet business/technical specifications. |
| Modeling | • Stakeholders use various models such as use cases diagrams, sequence diagrams and flowcharts to represent requirements, and specifications. However, keeping these models updated to reflect technical changes is challenging. | • Models are represented in a machine-readable form, associated with policies and specifications that enable service builders to automatically translate them into an executable code. |
| Design | • Relationships between the underlying objects are statically defined so they cannot be changed once created.<br>• UML and word processors are usually used to create system models. | • Bindings and relationships are dynamically defined at runtime instead of static definition at design/development time. Hence, new services can be dynamically created using existing ones.<br>• Sophisticated tools and languages such as MS.NET, BizTalk, WebSphere, Oracle SOA Suite, and BPEL are used to design services and workflows. |
| Implementation | • Development is performed by a single (virtual or physical) party that creates functions, classes, modules and libraries.<br>• Object-oriented languages such as java, c++, c# are used to develop system constructs. | • Development is divided between a service provider and an application builder. A service provider writes and exclusively owns the code of the offered services, whereas an application builder develops client applications that make use of the offered services. This separation enables application builders to focus on business logic while leaving the technical details to service providers.<br>• Open standards such as XML, WSDL, SOAP, XSD and XSLT are usually used to build and call services. |
| Testing | • Testing is usually performed by testers in the same organization.<br>• Validation and verification (V&V) is performed based on the source code and functional specifications.<br>• Test cases are defined by developers/testers, usually from within the same organization.<br>• Test scripts are defined by developers/testers. | • Testing is divided between a service provider, a broker and a client, with little or no interaction between them.<br>• Service provider tests services based on the functional specification and source code, and it then creates test cases for other parties.<br>• Service Providers give test cases to brokers and clients, and therefore services can be tested before their registration and usage, respectively.<br>• Test scripts can be automatically generated on the spot by both brokers and clients during the V&V process based on service metadata and specifications. |

In addition to educating software specialists about the basics of SOA, decision makers should learn about SOA so they can know in which scenarios it can be leveraged. In this regard, software and business schools should offer courses and curriculums about SOA [T+06]. This would permit IT and business professionals to gain the missing knowledge and skills that would allow them to apply SOA methods in situations where doing so make sense.

## 3. GOVERNANCE

Governance is probably the most important part in the entire adoption lifecycle. The broad definition of SOA governance refers to the processes, policies, principles and best practices that an organization applies to ensure the successful implementation of SOA. The purpose of having a solid governance framework is to efficiently control different aspects related to services, such as people, technologies, business and quality of service, and thus, delivering value to the organization [***07b]. Without efficient governance, SOA implementation could be chaotic: the total adoption process could easily fail. Governance strategies should be planned and defined before the implementation phase, not afterwards. This can be achieved by constructing an internal governing committee to manage all aspects of services. Governance aspects are like a puzzle that is composed of various components, including:

- Identifying SOA stakeholders including those who will participate in the governance body.
- Clearly defining the responsibilities of the identified stakeholders.
- Identifying and specifying services that add value to business.
- Listing the type and granularity of services.
- Defining the different versions and uses for each service.
- Choosing communication models and message patterns.
- Choosing technologies that will be used to realize the services.
- Designing formal specifications and description documents.
- Reviewing the addition and deletion of services.
- Creating V&V and test models.
- Establishing policies for fixes and updates.
- Planning monitoring/audit strategies.

## 4. BUDGET

One of the benefits of using SOA is the reduction of costs. However, this reduction is difficult to accomplish during the initial stages of the implementation [Rai09]. Adopters can expect expending significant resources to procure the necessary hardware and software tools, train staff, and convert legacy systems. These stages might last months or even years until they reach a maturity point.

Again, initiatives should not follow a "big-bang" approach to realize a successful implementation. Rather, they should use iterative and incremental implementations especially when constrained by the availability of resources. This approach would give adopters the chance to enjoy a hands-on experience to SOA while minimizing expenditures.

In fact, the successful adoption of SOA is threatened by the lack of awareness of the funding that should be dedicated during the initial phases. This inattention to the fiscal aspects could lead adopters to think that SOA is simply a bad methodology that causes them to spend too much money rather than saving overall costs. To alleviate these concerns, it is important for businesses to conduct a thorough cost/benefit analysis before beginning the design phase, so that costs are well understood. Furthermore, as benefits are broadly shared, it is important to identify potential stakeholders early in the design process so that costs are equitably shared.

## 5. TECHNOLOGY

In principle, different technologies can be used to create needed services. This includes Message Queues, COM, Jini and Web Services. Each of these technologies has its own strengths and limitations; thus, each of them has its own comparative advantages that best fit specific scenarios and requirements.

The question for adopters when thinking about using a specific technology is: To what extent is that technology supported by software vendors? Is that technology vendor-based or standards-based? Using vendor-based technologies will definitely prohibit loose-coupling between service providers and clients. The ultimate goal of this loose-coupling is to allow clients to use the services offered by the providers no matter what kind of technologies were utilized to build them. Furthermore, loose-coupling enables clients to change the used services and even the providers with a little effect, if any at all, on their applications.

Generally, adopters are highly encouraged to utilize standards-based solutions to be able to gain the optimum benefits offered by SOA. However, adopters should be careful because the support for the standard-based approaches varies from one vendor to another and from one tool to another. The extension of support to standards in products depends on different circumstances such as implementation challenges and marketing issues. In some cases, a vendor misinterprets standards to an extent that it just offers vendor-based solutions disguised as standards-based ones. Preserving loose-coupling and interoperability between service providers and clients in such cases might be problematic due to inconsistencies in the data types/formats, for example. To avoid such scenarios, both providers and clients are advised to read the original specifications, and current implementations of the adopted standards in the offered solutions. Understanding these specifications would enable SOA parties to abstract services and calls with wrappers that assure the use of original standards only instead of masked vendor-based offerings.

Examples to standards that could be used in SOA-based projects include CORBA, RSS, REST, and XML Web Services.

XML represents the head of the standards-based technologies pyramid available in the software field for realizing SOA applications. The credit for favoring the use of XML Web Services in SOA implementations goes to its benefits, which include:

- It is easy to learn and use by both providers and clients.
- Its broad support from key software vendors.
- Its affordability vis-à-vis proprietary technologies.
- Syntax is machine-readable, and therefore it enables different nodes to easily interpret the transmitted data.
- It is modular by nature, making it possible for implementers to encapsulate their logic into separate operations and services.
- It is "composable" allowing implementers to aggregate different services together.

Implementers should always keep in mind that regardless of the abovementioned advantages, XML is not the answer to all problems; it is just a rich tool that will fit some requirements. For instance, some modern SOA-based implementations use REST style to responsively return (representation) information in different data format such as HTML, MIME, and plain-text. Some other implementations use JSON-RPC as a light-weight, text-based standard to enable bidirectional communications and data interchange between clients and services.

## 6. SECURITY

Security contains a number of questions that need to be addressed before proceeding with the implementation. Some of these questions are:

- Which clients possess the rights to access available services?
- Do all clients have similar roles and rights?
- How will secrecy of sensitive information be assured when providers and clients belong to different organizations?
- Should services offer one view for the underlying information to its clients?
- What is the suitable transport for the bidirectional transmission of messages between services and clients (and possibly other intermediaries)?
- Which format is the data going to be transmitted in between nodes? Is it encrypted or not?

Answering these questions depends a great deal on the business requirements as well as the technologies utilized to achieve the offered services.

From a business perspective, it might be necessary to access one service with different security roles according to the sensitivity of the information offered. For example, the manager's role might require access to sensitive information, such as salary reports while the secretary's role is to only generate reports for the available employees with their hire dates. Moreover, transmission of sensitive information such as SSN and credit card numbers must be encrypted, while it is optional for non-sensitive data such as employee names and hire dates.

From a technical perspective, the used technologies should be able to cover business requirements in an efficient manner. To achieve this goal, these technologies should enable implementers to meet the five key security terms [Erl05]:

1. **Identification:** The ability to identify both service and data requesters.
2. **Authentication:** The ability to verify the identity of both service and data requesters.
3. **Authorization:** The ability to ensure that callers have permissions to execute the operations they are trying to access.
4. **Integrity:** Assuring that the data being transmitted remains unaltered while being sent between callers and services.
5. **Confidentiality:** Guaranteeing that the data being transmitted cannot be viewed while in transit except by other authorized services. In addition to ensuring that this data cannot be viewed or altered by unauthorized people/processes when stored on the providers' servers.

In this context, it is worth mentioning that while SSL is widely used to secure web applications, it is not the perfect solution for web services. Alternately, WS-Security framework can play a significant role in covering different web services security requirements. This ranges from encrypting the contents of confidential messages to federated sessions that allow one service to be authenticated once in multiple security domains. Protecting stored data with the appropriate encryption may also be required to prevent any unauthorized access to the database/files. Finally, traditional solutions such as firewalls, anti-virus software, denial-of-service and intrusion-detection systems, load-balancing, secure coding and input/schema validations, and data backups should be deployed to provide a higher security level to both clients and providers.

Another concern, which is related to security, is "trust" between service providers and clients. In fact, trust is an essential issue that should be dealt with before embarking on SOA. Some of the questions that adopters may have when considering trust include:

- How is the reliability and correctness of the offered services guaranteed?
- What if clients depend on providers who decide to leave the market?
- How will support and updates be applied and maintained?

Since trust is an intangible value, which cannot be in-

cluded in the service level agreement (SLA), it is one of the hardest challenges to overcome. Nevertheless, building trust between service clients and providers is possible, yet it cannot be achieved overnight –it is similar to trust between people.

## 7. PERFORMANCE

Performance is a non-functional key requirement in all software systems. However great the business features offered by one system, it might be deprecated for slow performance.

As mentioned, most of the contemporary implementations of SOA depend on XML to formulate requests and responses being exchanged between nodes. XML uses plain-text data for describing elements of the underlying objects. This certainly leads to bigger data files when compared with binary messages of other client-server technologies such as RPC.

A list of tactics that could help in overcoming performance issues is available, including [RHH09]:

- **Binary Format:** XML format is composed of many angle bracket tags. This makes both request and response files larger in size and more complex in structure; processing such files is neither an easy nor a fast process. One way to make the generated files smaller and simpler is to use Binary XML [Gee05].
- **Efficient Parsers:** Most of the XML parsers such as SAX and DOM depend on opening and reading files more than once, paging and caching data before parsing them. These techniques are inefficient when parsing large files. To save time and resources, fast, non-extractive techniques should be utilized. VTD-XML (Virtual Token Descriptor XML) is one of the popular examples for non-extractive approaches available in the market.
- **Schema-Specific Parsers:** Many XML-based implementations use general-purpose parsers to understand documents being exchanged between nodes. This makes the parsing process run slowly due to the need to extract and understand the structure of the files before the parsing phase itself. Parsers can run faster by caching serialization assemblies of data objects at providers and clients for later use.
- **Silicon-based Engines:** Many hardware solutions are available in the market to process XML data at a higher speed. These solutions might be embedded into different components including switches, routers, PCI-cards and servers.
- **Break Large Messages:** It is known that the probability of network clogs is very high when transmitting large messages between nodes. Accordingly, implementers should divide the logic embedded in one complex operation into a set of simpler operations. This could yield messages that are smaller, thus allowing them to move faster on the network.
- **Compression Algorithms:** ZIP/GZIP algorithms might be applied on the data being transmitted between nodes. This could allow implementers to eliminate additional spaces, making data files smaller.
- **High Speed Networks:** Fast network technologies such as Gigabit Ethernets, and fiber channels and links can be deployed to speed up the transmission rate between nodes. These advanced technologies are now available in the market, making it possible for implementers to build powerful infrastructures to meet the complex needs of SOA applications.

Adopters can mix and match these techniques according to their needs and available budgets.

## CONCLUSION

SOA is being widely accepted by organizations all over the world. However, its adoption is neither easy nor straightforward. This paper has introduced seven issues that SOA initiatives should consider during adoption phases:

1. Initiatives should start their migration process with the right understanding of SOA. A demonstration pilot along with an effective roadmap would be helpful for learning the basics with lower risks and costs.
2. After understanding SOA, a deeper knowledge to its aspects will be required. Offering training programs to the stakeholders would support the success of the migration process.
3. Before digging into the migration process, initiatives should define a governance framework that includes people, constraints, policies and strategies. A strong governance mechanism would enable the stakeholders to control and monitor the implementation phases, and thus, deliver value to the organization.
4. Although cost reduction is one of the benefits of SOA, this reduction is difficult to accomplish during the initial migration stages. Planning generous budgets will be required to procure the necessary hardware and software tools, train staff, and convert legacy systems. These budgets should be planned and shared by all stakeholders.
5. Choosing implementation technologies is the first step towards the realization of identified services. Adopters should choose technologies that best fit their needs. Standards-based technologies are gaining momentum in service-based projects; how-

ever, standardization is no silver bullet. For any given standard, a vendor may provide a subset or superset of features. Thus, analyzing tools is highly recommended before using them.

6. Unique security challenges are imposed as work is no longer done by a single entity. Assuring that data hosted on providers' servers or being transmitted cannot be accessed or changed by unauthorized parties/services is one of these challenges. Reputation and reliability of service providers are the key enablers for trust between SOA parties.
7. Finally, performance is by no means less important than the business features of the constructed services. Implementers should always deal with throughput and latency as well as request-response sizes as quality indicators to the offered services.

Paying close attention to these issues is important to ensuring a successful implementation. This success will be clear to adopters once they begin reaping the promised benefits, making organizations feel more secure in having made the right choice.

**REFERENCES**

[B+05] **N. Bieberstein et al**: - *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, 2005.

[Erl05] **T. Erl** - *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[Gee05] **D. Geer** - *Will Binary XML Speed Network Traffic?* IEEE Computer Society, 2005.

[Has09] **Q. F. Hassan** - *Aspects of SOA: An Entry Point for Starters*, Annals. Computer Science Series, 2009.

[Kob05] **J. Kobielus** - *The ROI of SOA: The more you reuse, the more you save*. Network World, 2005, www.networkworld.com/techinsider/2005/101005-roi-of-soa.html.

[KBS04] **D. Krafzig, K. Banke, D. Slama** - *Enterprise SOA Service-Oriented Architecture Best Practices*. Prentice Hall, 2004.

[L+05] **G. Lewis et al** - *SMART: Service-Oriented Migration and Reuse Technique*. Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, 2005.

[Pap03] **M. P. Papazoglou** - *Service-Oriented Computing: Concepts, Characteristics and Directions*. Proceeding of the Fourth International Conference on Web Information Systems Engineering, 2003.

[Rai09] **G. Raines** - *Leveraging Federal IT Investment with Service-Oriented Architecture*. The Journal of Defense Software Engineering, 2009.

[RH08] **A. M. Riad, Q. F. Hassan** - *Service-Oriented Architecture - A New Alternative to Traditional Integration Methods in B2B Applications*. Journal of Convergence Information Technologies, 2008.

[RHH09] **A. M. Riad, A. E. Hassan, Q. F. Hassan** - *Investigating Performance of XML Web Services in Real-Time Business*

*Systems*. Journal of Computer Science and System Biology, 2009.

[T+06] **Tsai et al** - *Perspectives on Service-Oriented Computing and Service-Oriented System Engineering*. Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering, 2006.

[T+**] **Tsai et al** - *Service-Oriented Computing and System Engineering*, Unpublished.

[***07a] *** - *ABCs of SOA*, CIO Magazine, http://www.cio.com/article/40941, 2007.

[***07b] OASIS - IT Governance and SOA Governance. 2007.