Quality Attributes and Service-Oriented Architectures

Liam O'Brien
Len Bass
Paulo Merson

September 2005

Software Architecture Technology Initiative

Technical Note
CMU/SEI-2005-TN-014

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Tables

Abstract

This report examines the relationship between service-oriented architectures (SOAs) and quality attributes. Because software architecture is the bridge between mission/business goals and a software-intensive system, and quality attribute requirements drive software architecture design, it is important to understand how SOAs support these requirements. This report gives a short introduction to SOAs and outlines some of the main business goals that may lead an organization to choose an SOA to design and implement a system. This report outlines a set of quality attributes that may be derived from an organization's business goals and examines how those attributes relate to an SOA. In addition, this report describes how the SOA impacts those attributes and how choosing an SOA can help an organization achieve its business goals.

1 Introduction

Building systems to satisfy current and future mission/business goals is critical to the success of a business or organization. Software architecture is the bridge between mission/business goals and a software-intensive system. Quality attribute requirements drive software architecture design [SEI 05]. Choosing and designing an architecture for such systems—one that satisfies the functional as well as the nonfunctional or quality attribute requirements (reliability, security, maintainability, etc.)—are vital to the success of those systems. Recently, the use of a service-oriented architecture (SOA) as the underlying architecture has been gaining popularity and widespread use as the architectural approach for various types of systems. Though some work has been done on how particular quality attributes such as security and interoperability are handled within an SOA, a more thorough examination of the relationship between SOA and quality attributes is needed. Such an examination is the subject of this report.

There are various definitions of what constitutes an SOA, some of which are outlined in Section 2 along with principles of service orientation. In this report, we use the term *service-oriented architecture* to mean an architectural approach for building systems or applications that use a set of services and not just a system that is built as a set of services. A service is an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes.

The choice to use an SOA approach in the development of an architecture depends on several factors including the architecture's ultimate ability to meet functional and quality attribute requirements. Usually, an architecture needs to satisfy many quality attribute requirements in order to achieve the organization's business goals. In almost all cases, tradeoffs have to be made between these requirements. In some cases, satisfying these requirements may be easier using an SOA; on others, it may be more difficult. If an SOA is being considered, several questions need to be asked; for example

- What effect will the choice of an SOA have on the business goals?
- Which quality attribute requirements will be positively impacted and which will be negatively impacted by the use of the SOA?
- What tradeoffs need to be made among the quality attributes?

Within the Department of Defense (DoD), digital information is rapidly becoming integrated into all aspects of military activities [Cherinka 05]. The DoD's goal is to find new and better ways of managing information and providing capabilities in response to quickly changing needs. The DoD is currently using the Network Centric Warfare (NCW) approach [Alberts 99]. This approach entails the networking of information producers (e.g., sensors); decision

makers and consumers to achieve shared awareness; increased speed and quality of decision making; and a higher tempo of dynamic operations. One of the approaches being employed to support NCW is SOA. This report does not address the DoD's usage of SOA but rather fundamental technical information about SOA. This report explores the relationships among business/mission goals, quality attribute requirements, and the use of an SOA approach.

This report is structured as follows. Section 2 has an introduction to SOAs and service-orientation principles. Section 3 outlines some business goals that an organization may have that would lead to choosing an SOA approach. Section 4 examines some quality attributes that may be important in helping an SOA achieve an organization's business goals. Section 5 outlines which of the quality attributes are supported and which are violated by the use of an SOA and then summarizes the impact of the SOA on the achievement of the organization's business goals. Section 6 summarizes the work outlined in this report.

2 Service-Oriented Architectures (SOAs)

There is no single, official definition of what an SOA is. Consequently, many of the organizations promoting the use of SOAs and building technologies to make it easier for organizations to adopt an SOA approach have defined the term. As a result, SOA is defined in many different ways, including

- "A service-oriented architecture (SOA) is an application framework that takes everyday business applications and breaks them down into individual business functions and processes, called services. An SOA lets you build, deploy and integrate these services independent of applications and the computing platforms on which they run."*—IBM Corporation*
- "Service-Oriented Architecture is an approach to organizing information technology in which data, logic, and infrastructure resources are accessed by routing messages between network interfaces."*—Microsoft*
- An SOA is "a set of components which can be invoked, and whose interface descriptions can be published and discovered."*— Worldwide Web Consortium [W3C 04]*

Our use of SOA in this report is as an architecture for a system or application that is built using a set of services. An SOA is not just a system built as a set of services. Most definitions of an SOA would agree that an SOA defines application functionality as a set of shared, reusable services. Our definition does not rely on any particular technology or implementation of an SOA.

Just as there is no official definition of SOA, there is no official set of service-orientation design principles. There are, however, a common set of service-level design principles most associated with service orientation [Erl 05, McGovern 03]:

- **Services are reusable.** Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
- **Services share a formal contract.** In order for them to interact, they need not share anything but a formal contract that defines the terms of information exchange and any supplemental service description information.
- **Services are loosely coupled.** They must be designed to interact on a loosely coupled basis, and they must maintain this state of loose coupling.
- **Services abstract underlying logic.** The only part of a service that is visible to the outside world is what is exposed via the service's description and formal contract. The underlying logic (beyond what is expressed in the description and formal contract) is invisible and irrelevant to service requestors.

- **Services are composable.** They may compose other services. This possibility allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.
- **Services are autonomous.** The logic governed by a service resides within an explicit boundary. The service has complete autonomy within this boundary and is not dependent on other services for the execution of this governance.
- **Services are stateless.** They should not be required to manage state information, since that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- **Services are discoverable.** They should allow their descriptions to be discovered and understood by humans and service users who may be able to make use of the services' logic. Service discovery can be facilitated by the use of a directory provider, or, if the address of the service is known during implementation, the address can be hard-coded into the user's software during implementation.
- **Services have a network-addressable interface.** Service requestors must be able to invoke a service across the network. When a service user and service provider are on the same machine, it may be possible to access the service through a local interface and not through the network. However, the service must also support remote requests.
- **Services are location transparent.** Service requestors do not have to access a service using its absolute network address. Requestors dynamically discover the location of a service looking up a registry. This feature allows services to move from one location to another without affecting the requestors.

Of the principles described above, autonomy, loose coupling, abstraction, and the need for a formal contract can be considered the core principles that form the baseline foundation for SOA.

Each service implements a discrete business function, and any application that needs to perform that function uses the shared service to do so. Furthermore, an application is created by assembling and coordinating the activities between the appropriate services it needs to accomplish its business process. The services within the application are loosely coupled and reusable across multiple applications. All of an application's functionality does not have to be made up of services, but it could be. Additional code can be contained in the application that carries out specific functionality for that application.

The realization of the vision for an application that uses an SOA approach is dependent on which services are exposed. Services may already exist within the organization and thus can be reused when building an application. Services provided from outside of the organization may also be considered when building an application. Part of the functionality for an application may be built as services that are exposed, so they can be used by other applications within the enterprise.

With an SOA, if the business rules associated with a specific function change, developers must modify only the one service that implements the function. In theory, all applications that use the service will then automatically adopt the new business rules. And when a new business opportunity appears, developers can rapidly implement a new business process by assembling a new application from the available services and adding additional services as required.

It is critically important to identify what piece of functionality will become services and to define the interfaces of those services. The granularity of the service (i.e., the scope of functionality a service exposes) is also important because having many fine-grained services may result in a lot of message passing between the service users and the service providers. Coarse-grained services are recommended, which, like any other large pieces of software, may need to be architected themselves.

Once the services are implemented and the interfaces are published and used by applications, it is difficult to refactor or change them because doing so could potentially require changes to all applications that use those services. Such changes should be avoided. Once developed, services can be combined in different ways to help achieve the specific business goals of an organization.

3 From Business Goals to Service-Oriented Architectures

This section describes typical business goals for a system and how they might lead an organization to use an SOA for the system's implementation.

3.1 Typical Business Goals

Typical business goals that an organization may have for its system are to

- be agile—able to adapt quickly to new opportunities and potential competitive threats
- be first to market with innovative services for its customers
- streamline business processes to reduce operating costs
- enable easy and flexible integration with legacy systems

Because architects try to satisfy the system's requirements (both functional and nonfunctional) and meet the organization's business goals, they may choose to use an SOA approach in architecting and designing the system.

3.2 How Business Goals Are Satisfied with an SOA

Agility can be achieved in many ways. However, a thorough analysis of the business needs with agility in mind is required, along with a careful design of the solution that is architected for agility. Wilkes and Veryard outline a set of principles for architecting and designing an SOA that impacts agility [Wilkes 04]. These principles include some of those outlined earlier—such as loose coupling and precise specification of services—and additional ones such as standardized services, standards compliance, and defining services as coarse-grained.

The ability to reuse both internal and external services within an SOA approach, combine services in new and different ways, and add additional services where needed reduces the time required to develop new applications and bring them to the market.

Integrating and streamlining business processes is a critical component of business success—especially when an organization is dealing with the supply chain and trading partners. Streamlining the business process reduces support costs and effort. One of the benefits of using a service-oriented approach is that it lets organizations couple disparate computing systems, which means that existing and newly developed processes can be combined to meet new process needs.

Most legacy applications have rich functionality that, if service enabled, can be reused in an SOA as components. If built properly with non-invasive tools and techniques, service-

enabled legacy components can remain intact and operational within a legacy architecture and be made available as services at the same time. The cost of redeveloping legacy systems is often prohibitive, so making them service enabled is an economic option that many organizations are pursuing.

3.3 Using Internal or External Services

As an organization moves ahead with an SOA approach, it has to determine whether to develop the required services internally or use those developed outside of the organization. Internally developed services are those developed from scratch or created from proprietary legacy components. An organization may not have the resources or expertise to develop services internally, so it gets the functionality needed for its applications from outside vendors.

Externally developed services are those provided and managed by a third party. The third party providing the service can be, for example, a business partner that offers services for use in business-to-business transactions. In other cases, the third party offers a public service that can be accessed for free (e.g., browsing Amazon catalogs) or for a subscription fee. The organization can choose to identify what third-party services to use at either design time or runtime through the use of discovery services or agents. If a service is identified at design time, its location can be hard-wired into the application, and the organization can establish a service level agreement (SLA) with the third party to make sure that an adequate level of service is provided for it. If a service is discovered at runtime, identifying which third parties are providing it and arranging an SLA may be more difficult.

4 SOAs and Quality Attributes

As mentioned earlier, quality attribute requirements drive the design of an architecture. But what are their implications when designing and implementing an SOA? This section describes these implications for a set of quality attributes in the context of an SOA.

4.1 Interoperability

Interoperability refers to the ability of a collection of communicating entities to share specific information and operate on it according to an agreed-upon operational semantics [Brownsword 04]. Increased interoperability is the most prominent benefit of SOA, especially when we consider Web services technology [McGovern 03]. Distributed systems have been developed using various languages and platforms that vary from portable devices to mainframes. They have used technologies such as the Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI), Distributed Component Object Model (DCOM), Remote Procedure Call (RPC), and sockets for communication. However, until the advent of Web services, there was no standard communication protocol or data format that could be used effectively by systems using different technologies to interoperate on a worldwide scale.

Today, mainstream development platforms—such as Microsoft's .NET and Sun's Java 2 Enterprise Edition (J2EE), as well as open source alternatives (e.g., Perl and PHP)—provide frameworks to implement Web services. Components implemented in disparate platforms using different languages can interact transparently through a call-and-return mechanism. That is possible because Web services define the interface format and communication protocols but do not restrict the implementation language or platform. Other SOA technologies—such as Jini (http://www.sun.com/jini) and CORBA—have not yet produced enough of a following to become a universal standard for interoperability [McGovern 03].

However, the promise of cross-vendor and cross-platform interoperability in Web services begins to fall short when services start to use features beyond the basic Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) standards. Over the last few years, a myriad of Web services standards (e.g., Business Process Execution Language [BPEL], WS-Security, and ebXML) has emerged from a number of standards bodies. Web services development platforms do not implement the same standards and the same versions, so interoperability may not be as seamless in practice as it is in theory.

Recognizing that reality, the Web Services-Interoperability Organization (WS-I) (http://www.ws-i.org) was chartered in 2002 to promote the interoperability of Web services across platforms, applications, and programming languages. WS-I publishes *profiles* that

prescribe adherence to a group of specific versions of well-defined standards. Another goal of WS-I is to provide tools to certify conformance with those profiles. The WS-I initiative has grabbed considerable attention in the industry through its 130 (approximately) member organizations, including Web services platform vendors (e.g., IBM, Microsoft, BEA, Oracle, and Sun) and end-user organizations (e.g., AOL, General Motors, MCI, and SAP). Many Web services products were updated in recent years because of this initiative; it is not uncommon now to find "compliant with WS-I Basic Profile 1.0" as a specification in data sheets. WS-I has created a few profiles and other deliverables but still has a lot of work to do to cover all layers and standards in the Web services stack. Because the major benefit of Web services is interoperability, the success of this initiative will determine the success of Web services. The past has shown that the existence of published standards is not sufficient to ensure interoperability across platforms from different vendors. CORBA, for example, allows objects written in different languages to interoperate. In practice, communicating objects have to use underlying object request brokers (ORBs) from the same vendor because of incompatibilities across vendor implementations.

4.2 Reliability

Reliability is the ability of a system to keep operating over time [Clements 02]. Several aspects of reliability are important within an SOA, particularly the reliability of the messages that are exchanged between the application and the services, and the reliability of the services themselves. Applications developed by different organizations may have different reliability requirements for the same set of services. And an application that operates in different environments may have different reliability requirements in each one.

4.2.1 Message Reliability

Services are often made available over a network with possibly unreliable communication channels. Connections break and messages fail to get delivered or are delivered more than once or in the wrong sequence. Although techniques for ensuring the reliable delivery of messages are reasonably well understood and available in some messaging middleware products today, messaging reliability is still a problem. If reliability is addressed by service developers who are incorporating reliability techniques directly into the services and application, there is no guarantee that they will make consistent choices about what approach to adopt. The outcome might not guarantee end-to-end reliable messaging. Even in cases in which the application developers defer dealing with the reliable messaging to messaging middleware, different middleware products from different vendors do not necessarily offer a consistent approach to dealing with the problem. The use of middleware from different vendors might preclude reliable message exchange between applications and services that are using different message-oriented middleware [Weerawarana 05].

Two specifications—WS-Reliability (OASIS Consortium) and WS-ReliableMessaging (developed by IBM, BEA, Microsoft, and TIBCO Software)—address these issues and the

task of defining protocols that enable services to ensure the reliable, interoperable exchange of messages with specified delivery assurances. These specifications define four basic assurances that can be combined:

1. in-order delivery – The messages are delivered in the same order in which they were sent.
2. at-least-once delivery – Each message that is sent is delivered at least one time.
3. at-most-once delivery – No duplicate messages are delivered.
4. exactly once – Each message is sent only once.

WS-Reliability was approved as a standard in January 2005. WS-ReliableMessaging has been submitted to OASIS for ratification as a standard. It is uncertain which one will survive and be used in the future as the basis for reliable messaging within an application. Many of the main service software vendors are backing WS-ReliableMessaging.

4.2.2 Service Reliability

Service reliability means the service operates correctly and either does not fail or reports any failure to the service user. Service reliability also means making sure that the service is obtained from a reliable provider so that a level of trust in the service's accuracy and reliability can be established. Issues that have to be dealt with include managing the transactional context; for example, dealing with failures or some form of compensation if the service fails. In some cases, brokers or intermediaries may link the service users and providers. As a result, these issues may be handled by third parties.

4.3 Availability

Availability is the degree to which a system or component is operational and accessible when required for use. Availability of services both from the user's and provider's perspectives is a concern for the success of an SOA. From the services user's perspective, if the system relies on a set of services being available in order to meet its functional requirements and one of those services becomes unavailable (even transiently), it could have dire consequences on the success of the system. From the service provider's perspective, in order for the services to be used (for which the provider may receive compensation), they must be available when needed. Otherwise, the provider's finances and reputation could be impacted (especially if compensation has to be paid when the services are not available).

Service providers usually agree to provide to the service users a set of services and to include each service in an SLA. The SLA defines the contract for the provision of the service with details such as who provides the service, the guaranteed availability of the service, the escalation process (which is followed if the service is not handled to the service user's satisfaction), and the penalties to the provider if the service level is not met. Usually, both the provider and user offer some form of capability for monitoring the services' availability and other quality-of-services requirements such as performance. Furthermore, service users who

build systems that rely on particular services being available must build contingencies (such as exception handling) into those systems, in case the services become unavailable. For example, the application could find an alternative provider for a service.

4.4 Usability

Usability is a measure of the quality of a user's experience in interacting with information or services. To provide a more usable system, service providers should consider several things that derive from the distributed and service nature of SOA: data granularity, services to support usability, and disconnected operation.

4.4.1 Data Granularity

In SOAs, service users and service providers communicate over a network—a process that can introduce delays, possibly on the order of seconds, in user interactions. To avoid these delays, not only must the service respond to user requests with the data requested but also with other relevant data that may not be immediately displayed. Examples of the additional data that might be necessary include

- lists of valid inputs and alternatives to incorrect inputs. For example, in an airline reservation system, the user inputs an ambiguous destination, such as Paris. The system will normally provide a list of alternatives for Paris: for example, Paris, France; Paris, Texas; or Paris, Tennessee. If the provision of these alternatives requires net traffic, the response time to the user may be delayed. Hence, the service should provide a list of valid destinations to the service user when the user brings up the reservation system.
- map detail. If the user is examining a particular portion of a map, likely operations are panning and zooming. If each request by the user requires invoking the map service, again, the response time may be increased. The protocol for the service should include the provision of additional data (to support panning and zooming) based on the current data provided.
- supplemental information for possible choices. A common interface technique is to display additional information when the user places the mouse over a potential selection. The service must provide enough information for the service user to be able to display the additional information for each possible selection.

4.4.2 Normal Usability Operations

The service must provide interfaces that support the normal usability operations such as canceling a request, undoing the last request, providing the service on aggregated data, and gaining information for feedback such as percentage completed and time to completion. Bass and John provide a list of possible operations [Bass 03].

The service user might be mobile and lose connectivity for periods of time. If the service is implemented in a stateless fashion, reestablishing the context is the responsibility of the

service user. Service users should probably cache sufficient data to allow for reduced operation for some time until reestablishment occurs. If, however, the service provides some operations that require persistent data (such as adding items to an e-shopping cart), the service should provide a means of reestablishing context or synchronizing the current state of the client with the state maintained by the service.

4.5 Security

Although security denotes different things with respect to software systems, in general, it is associated with four principles:

1. confidentiality – Access to information/service is granted only to authorized subjects.
2. authenticity – We can trust that the indicated author/sender is the one responsible for the information.
3. integrity – Information is not corrupted.
4. availability – The information/service is available in a timely manner.

Security is a major concern for SOA and Web services. Architects should pay heed to some characteristics that are inherent to SOAs and directly impact security:

- Messages often contain data in text format (e.g., XML), and, even worse, metadata is embedded. That means that someone intercepting a message may clearly see a 16-digit number as well as metadata revealing that the number is the value of a credit card field. Encryption must be in place to preserve privacy.
- A system built using an SOA approach may encompass services provided by third-party organizations. Trust must be built into the security of such external services. The identity of the external service provider has to be authenticated, but sometimes authentication is not enough. Building trust may involve other concerns. For instance, if the system sends classified data to the external service, the data should be protected not only when it is transmitted but also when it is stored.
- Services may have access restrictions based on the identity of the service user. In that case, an authorization mechanism should be in place that allows configuring and enforcing permissions to be set to specific users, groups of users, or roles.
- An SOA solution may rely on looking up services in a public directory. It is important to ensure that information in the directory is up to date and was added by valid publishers.

Web services solutions have been addressing some of the security concerns at the network infrastructure level. For example, Web servers that host Web services can be configured to use Secure Sockets Layers (SSLs) and digital certificates to encrypt data transmission and authenticate the communicating parties. In intranet solutions, Kerberos is an option—users receive a ticket for access to each Web service they have permission to use. However, these solutions merely help to protect point-to-point interaction: A comprehensive mechanism that covers end-to-end security is required.

In 2002, IBM, Microsoft, and VeriSign proposed Web Services Security as a comprehensive security model for Web services. Besides the core WS-Security policy for message protection, the original proposal contained a roadmap of complementary security specifications [Atkinson 02]. These specifications (WS-Authorization, WS-Privacy, WS-Trust, WS-Federation, WS-Policy, and WS-SecureConversation) are gradually developing into standards. The WS-Security specification was submitted to OASIS, and the first version of the standard was approved in 2004 [OASIS 04]. WS-Security defines a standard set of SOAP extensions that can be used to provide message content integrity and confidentiality. It accommodates a variety of security models and encryption technologies and is extensible to support multiple security token formats.

Two other proposed standards are also relevant to the Web services and security concerns: (1) Security Assertions Markup Language (SAML) and (2) eXtensible Access Control Markup Language (XACML). SAML provides a standard, XML-based format to exchange security information between different security agents over the Internet. It allows services to exchange authentication, authorization, and attribute information without organizations and their partners having to modify their current security solutions [McGovern 03]. XACML complements SAML by providing a language to specify role-based, access control rules in a declarative format.

The architect should be aware of the security features offered by the target Web services platform. Security mechanisms often have a negative impact on performance and modifiability, so the architect may want to investigate these tradeoffs on specific platforms. Also, adherence to security standards is important to preserving interoperability. In fact, WS-I has been working on a basic security profile that will ensure interoperability of security features among compliant vendors [WS-I 04].

In addition to the SOA security mechanisms available (e.g., encryption, authentication, and authorization), the architect should consider the configuration required for the infrastructure of the chosen technology with respect to security. For example, it is possible for a service user to interact with a remote provider via the Internet using CORBA. In this case, the rules of the firewall on both ends would probably have to be relaxed to allow Internet Inter-ORB Protocol (IIOP) communication. On the other hand, in a Web services solution, the firewall rules don’t have to change because the SOAP interaction is over a protocol that is normally open (e.g., HTTP or SMTP).

One of the goals of security is to maintain information integrity. One major challenge in SOAs and Web Services is to maintain data integrity during failures and concurrent access. Transaction management is more difficult in such a distributed, loosely coupled context for two reasons. First, services are usually implemented in a stand-alone fashion, and transactions begin and end within the service. Therefore, transactions that involve the composition of services require either nested transactions or a redesign of transaction demarcation. Second, agents performing data changes (i.e., the services) are distributed, and, hence, a distributed transaction model is needed. Because services may be implemented in

different languages and platforms, the implementation of distributed transactions—using two-phase commit, for example—requires compatible transaction agents in all end points that interact using a standard format. Two different standards have been proposed that address transactions across Web services: (1) the Business Transactions Protocol (BTP) by OASIS and (2) the Web Services Transactions (WS-Tx) published by IBM, BEA, Microsoft, and others [Little 03]. The architect of an SOA solution that requires transactions should understand the differences and limitations of existing standards and look for what is supported by the infrastructure to be used.

4.6 Performance

Like security, performance can have different meanings in different contexts. In general, it is related to response time (how long it takes to process a request), throughput (how many requests overall can be processed per unit of time), or timeliness (ability to meet deadlines, i.e., to process a request in a deterministic and acceptable amount of time). Performance is an important quality attribute that is usually affected negatively in SOAs. Careful design and evaluation of the architecture for the specific solution is necessary to avoid performance pitfalls. The key factors in SOA that contribute to performance issues are

- SOA involves distributed computing. Service and service user components are normally located in different containers, most often on different machines. The need to communicate over the network increases the response time. Typical networks used for SOA, such as the Internet, do not guarantee deterministic latency. Therefore, SOA is not considered a feasible solution for real-time systems where timeliness is a strict requirement. SOA presents challenges for near real-time systems, where latency is not a safety-critical requirement but rather a business one (i.e., meet business goals). For a heavily used service, many queued requests may already be outstanding, and they are usually serviced in a FIFO manner. Such a situation can have a significant impact on latency, though it can still be predicted stochastically. However, if more queue space has to be created dynamically, latency will be impacted further.
- The interaction protocol sometimes requires a call to a directory of services to locate the desired service. This extra call increases the total time needed to perform the transaction. One way to reduce the response time and improve throughput is to prevent the call to the directory by having the location of the provider end point hard-coded (or cached after the first lookup) in the service user. However, hard-coding reduces availability, and caching must be reestablished after failure when another replica is found.
- The ability to make services on different platforms interoperate seamlessly has a performance cost. Intermediaries are needed to perform data marshalling and handle all communication between a service user and a service provider. Depending on the SOA technology or framework being used, stubs, skeletons, SOAP engines, proxies, and other kinds of elements are in place. All such intermediaries negatively impact performance.

- The use of a standard messaging format increases the time needed to process a request. As an example, the next section describes how the use of XML impacts the performance of Web Services.

On the positive side, SOA provides location transparency. Service users do not necessarily know the location of the service until they look it up in the registry. Thus, a deployed service can be moved from location to location without affecting the consumers. This feature permits the deployment of services to multiple locations (replicas), which can be allied to a load-balancing strategy to improve the total throughput and availability of the system.

Many SOA technologies permit the service user to call the provider asynchronously. In that case, the user does not get blocked waiting for the response. For operations that fit that model of interaction, asynchronous calls should be used to reduce the response time.

4.6.1 XML in Web Services as a Performance Factor

XML is the data format used throughout the Web Services infrastructure. It is an open standard endorsed and widely adopted by the industry. XML is flexible and extensible, making it suitable to represent any data that can be stored in text format. It has internationalization mechanisms to support multilingual documents. XML documents are human readable: that is, they use a text rather than binary format. In addition to data, XML documents may embed metadata describing the structure of the data. Despite all the benefits, the use of XML as the data representation format in Web Services creates additional overhead in the transmission and processing of data.

XML messages can be 10 to 20 times larger than the equivalent binary representation, so transmitting them over a network takes longer. Because XML uses a text format, it has to be processed before any operation is performed. XML processing consists of at least three distinct activities, all of which are CPU and memory intensive:

1. **parsing:** the translation of XML data into the proper data structures of the component that consumes the XML. Parsing involves a lot of string processing.
2. **validation:** a step prior or concomitant to parsing that ensures that the XML document follows a predefined structure. Validation can be more time-consuming than parsing, especially when a reference to a remote Document Type Definition (DTD) or schema has to be resolved.
3. **transformation:** the translation from one XML structure to another or from XML to some other format. Transformation is usually required when integrating services and components that come from different providers. Transformation can be 10 times slower than XML parsing and should be the first target for performance improvement when optimization of Web Services' performance is the goal.

Many techniques and best practices can be applied to minimize the performance costs of transmitting and processing XML documents; for example

- Use data compression (e.g., Zip format) on the XML document. There is a tradeoff with a loss in interoperability because both end points must be able to compress/decompress the documents using the same algorithm.
- Use the appropriate parsing model. The Document Object Model (DOM) should be used when elements in the XML document have to be accessed randomly or when the document has to be processed multiple times or modified. Use the Simple Application Programming Interface (API) for XML (SAX) when the elements have to be processed serially and just once. Also, don't parse the entire document if you can obtain the desired information by reading only part of it.
- Turn off validation for documents that were generated by an application and are known to be valid. Once a document is validated, it can be converted to its DTD-less or schema-less equivalent. Also, remote DTDs and schemas can be cached locally or embedded into the XML documents to avoid the remote access.

4.7 Scalability

Scalability is the ability of an SOA to function well (without degradation of other quality attributes) when the system is changed in size or in volume in order to meet users' needs [W3C 04]. Very little has been done to address the scalability issues related to SOA [Cohen 05]. One of the major issues in scalability is the capacity of the site where the services are located to accommodate an increasing number of service users without a degradation of the services' performance as already described.

Because service users know only about the service's interface and not its implementation, changing the implementation to be more scalable requires little overhead [McGovern 03]. Options for solving scalability problems include

- horizontal scalability: distributing the workload across more computers. Doing so may mean adding an additional tier or more service sites.
- vertical scalability. upgrading to more powerful hardware for the service site

If addressing scalability poses potential performance issues, the source of the delays must be identified—whether it is the transport protocol, the XML parser, or the SOAP runtime. The performance of the system must be studied, and performance models must be built that capture the magnitude of the scalability and load tests required; for example, if the system will need to deal with 10, 1,000, or 10,000 service users. These performance models will need to be recalibrated based on whichever solution option or combination of options is chosen to make sure that the new deployment configuration will meet the intended scalability requirements without affecting the system's performance.

4.8 Extensibility

Extensibility is the ease with which the services' capabilities can be extended without affecting other services or parts of the system. Extensibility for an architecture today (in particular, an SOA) is important because the business environment in which a software system lives is continually changing and evolving. These changes in the environment will mean changes in the software system, service users, and services providers and the messages exchanged among them. Extending an SOA means making changes that include extending

- the architecture to add additional services. SOAs allow for the easy addition of new services through loose coupling and the use of various Web standards. Services can be created and published by the providers and discovered by service users. Service users must update their application code to incorporate these new services.
- existing services without changing the interfaces. Because services are loosely coupled, adding new capabilities to them that do not require a change in the service interface can be done without affecting other services. However, an application may require changes if these new capabilities were already incorporated into the application (i.e., the functionality for these capabilities was either included in the application or handled by additional services). Identifying the services' capabilities when they are first designed and implemented is very important because later, changes may cause problems within the service users' applications.
- existing services with changes to interfaces. Adding new capabilities to a service—ones that require changes to the service interface—may have a major impact on the success of an SOA. Usually, an application learns about a service's interface by reading information provided by the directory provider, and the interface may change over time. The service users' application must be able to handle any changes to the interface.

A major obstacle to extensibility is the interface message. If interface messages are not extensible, users and providers will be locked into one particular version of the interface to a service. Moreover, messages must be written in a format, structure, and vocabulary understood by all parties. Limiting the vocabulary and structure of messages is a necessity for any efficient communication. The more restricted a message is, the easier it is to understand, although it comes at the expense of reduced extensibility. Restriction and extensibility are deeply entwined. Both are needed, and increasing one comes at the expense of reducing the other. Tradeoffs between them are necessary to achieve the right balance.

4.9 Adaptability

Adaptability means the ease with which a system may be changed to fit changed requirements. Adaptability for a business means it can adapt quickly to new opportunities and potential competitive threats, which implies that the application development and maintenance groups within the business can quickly change the existing systems. The use of an SOA approach brings various benefits to the ability to adapt by allowing the following:

- Services can be built and deployed using the principles of location and transport independence and declarative policy. As a result, service users can dynamically discover and negotiate the method to be used for binding and the behavior to be exhibited for interacting with a service. If the service needs to adapt, this discovery and binding should be automated and not require a change in the application. It is the role of the application development group to build and maintain the abstraction layer between services and the underlying technology. It is the role of the architect to build applications that are resilient to infrastructure and organizational policy changes [Bruce 04].
- Business processes that are modeled using services can be adapted, and those services can be combined in new and different ways. Additional services can be added, or adapted services can be swapped in where needed. What will require changes is the underlying application using these services. In an SOA, in which most of the system's functionality is contained in the services, this task should not be a major one, as compared to a system in which the majority of the underlying application makes up the business process, a large amount of which will have to change.
- Services are being developed that must operate on different platforms, in different computing environments (including development and testing environments), using different combinations of sensors, multiple diverse communication protocols, human-computer interaction (HCI), and applications, and pursuing different missions. These services must be "configurable" to the environment in which they will reside—a significant adaptation challenge that requires "spiral" development with incremental deliveries to particular platforms, interoperability between different platforms, and backwards compatibility to multiple previous releases.

To achieve adaptability, the services will need to be managed and monitored properly as a single cohesive solution, and the interaction between the service and the underlying infrastructure will have to be managed. Proper measurements of capacity, performance, and availability are required to support this management function.

4.10 Testability

Testability is the degree to which a system or service facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE 90]. Testing a system that uses an SOA can be complex for many reasons including

- Interactions may be required between distributed pieces of the system (i.e., pieces that run on different machines across a network).
- The organization may not be able to access the services' source code, so it can't identify the test cases required to thoroughly test them. This problem occurs when the services are external to the organization that owns the applications.
- Services may be discovered at runtime, so it may be impossible to predict which service or set of services is actually used by a system until the system is executing. In addition, different services from different providers may be used at various times when the system

runs. The services used may be running on different platforms or operating systems and use different middleware technologies. Building repeatable tests and automating the testing process for such a system will be a challenge.

The authors of this technical note are not aware of any standards being developed to support the testing of an SOA. However, some of the issues related to the testability of a system that uses an SOA approach include

- Will the service user be able to obtain information about the service that will allow the user to sufficiently test the service?
- What information needs to be captured in log files, and where should those files be located?
- Will the service provider be able to provide logging information about the service to the service users?
- How can the application builder that is using a set of services guarantee that the system is relatively bug free?

If a problem occurs when the system is running, it may be difficult to find the source of the problem. The problem may be

- within the application
- within a service that is being used by the application
- within the infrastructure that is used by either the application or the service
- due to the load on the platform where the service executes
- within the discovery agent that locates the service

There are many potential sources for the problem, and trying to replicate it in a test environment may be extremely challenging, if not impossible, because the service is provided by an outside source—a fact the service user can't change. Service providers may need to build additional services and infrastructure that support the testing and debugging processes of both the service and the service users. Some tools are available to support the unit testing of services and integration testing when services are used in an application.

Within the DoD, integration testing of a weapon or sensor platform is the responsibility of the program of record associated with the platform. This testing is conducted through extensive and expensive test facilities (including hardware-in-the-loop and extensive simulation capabilities). As more services move "off board" or "between platforms," testing may involve integrating these currently stovepiped facilities and running coordinated and repeatable tests. Such integration is a significant side effect of SOA.

4.11 Auditability

Auditability is the quality factor representing the degree to which an application or component keeps sufficiently adequate records to support one or more specified financial or legal audits. With the ever-increasing need for systems to comply with business and regulatory legislation (financial and health sectors especially), the ability to audit a system for compliance is an important consideration. However, the flexibility offered by SOAs may make such audits difficult. If an application using an SOA approach dynamically uses external services, it may be difficult to track which services are actually used. If an outside service uses additional services (i.e., is composed of other services) to carry out its functionality, the audit process becomes even more complex.

A well-defined model for the end-to-end audit, logging, and reporting of distributed service requests is needed [Gall 03]. An authorization decision must be traceable retroactively to the true identity of the entity accessing the service. One way of achieving the end-to-end auditability is to include the business-level-identifying metadata in each SOAP header so that each SOAP agent can capture the metadata in its audit logs. Tracing identity end to end would consist of tracing through the SOAP node's audit log to discover each identity transformation. This means that various service providers and users will need to use standards that allow their services to be audited.

Because some archiving capability may be required and the usefulness of the archive decays over time, data must be moved from storage that can be accessed easily to storage that is more difficult and time-consuming to access. And eventually, the data must be destroyed.

4.12 Operability and Deployability

Typical data centers are complex, heterogeneous collections of hardware, middleware, and software from multiple vendors. These centers are increasingly difficult to create and maintain. The projected growth trends for data centers show that the complexity of operating these centers may outgrow the capability of manually managing them. The industry has recognized these trends and problems and responded with several initiatives. IBM's Autonomic Computing Initiative has the vision that systems will be self-regulating for those functions necessary for the data center to continue to operate [IBM 05]. Microsoft's Dynamic Systems Initiative has the vision that ongoing operations will become increasingly more automated [Microsoft 05]. Since these data centers house the service providers, SOA must be able to operate in an increasingly self-healing and automated operations environment.

The following main activities could be better automated:

- security policy development
- asset management
- authentication systems including password management
- backup

- security monitoring (traffic, application systems, IDS [Intrusion Detection System], firewall)
- patch coordination
- vulnerability assessment (proactive scanning)
- special system security administration (Web server, mailer, ftp, firewall, IDS)
- deployment of service updates

Organizations that run Web services frequently have an SLA through which they guarantee their service users particular levels of service. During operations, these SLAs need to be monitored, and when a violation has occurred or is likely to occur, remedial action must be taken. The IBM and Microsoft visions mean, among other things, that policies for the remedial action would be machine interpretable and the service (or the infrastructure) would take actions such as restricting access, allocating more resources, and creating replicates. The goal of these actions is to improve one of the service's qualities such as security, performance, or reliability.

4.13 Modifiability

Modifiability is the ability to make changes to a system quickly and cost-effectively [Clements 02]. SOA promotes loose coupling between service consumers and providers. Services are self-contained, modular, and accessed via cohesive interfaces. These characteristics contribute to the creation of loosely coupled SOAs where there are few, well-known dependencies between services. That fact tends to reduce the cost of modifying the implementation of services, hence increasing the system's modifiability. However, if service interfaces need to be changed, the change may create problems because once service interfaces are published and used by applications, it can be difficult to identify who is using a service and what impact changing its interface will have.

5 Summary of SOA's Impact on Quality Attributes and Business Goals

This section summarizes the positive and negative effects that using an SOA has on a system's quality attributes. In addition, this section describes how quality attributes that are inherently affected by an SOA approach impact business goals. As with any architecture, tradeoffs between quality attribute requirements must be made, and the resultant decisions may impact the organization's ability to meet its business goals.

In Section 4, SOAs and quality attributes were discussed in general. In each system and hence for each SOA, quality attributes must be characterized specifically (e.g., using scenarios), and then these characterizations must be weighed against the information provided in Section 4.

5.1 SOA Approach's Impact on Quality Attributes

The SOA approach's impact on the quality attributes is shown in Table 1. The "Status" column refers to the level of maturity SOA has in each area. The color green indicates that there are known solutions for the SOA based on relatively mature standards and technology. The color yellow indicates that some solutions exist but need further research to prove their usefulness in handling the requirements for the quality attribute. The color red indicates that the standards and technology are immature and further significant effort is required to fully support the quality attribute within an SOA.

Table 1: SOA Approach's Impact on Quality Attributes

Quality Attribute	Summary	Status
Interoperability	Through the use of the underlying standards, an SOA provides good interoperability technology-wise overall, allowing services and applications built in different languages and deployed on different platforms to interact. However, semantic interoperability is not fully addressed. The standards to support semantic interoperability are immature and still being developed.	**Green**
Reliability	Potentially, problems can occur in many areas, but the use of the underlying standards (WS-Reliability and WS-ReliableMessaging) should mean that messages are transmitted reliably. Service reliability is still an issue as with any element in an architecture.	**Yellow**

Table 1: SOA Approach's Impact on Quality Attributes (cont'd.)

Quality Attribute	Summary	Status
Availability	It is up to the service users to negotiate an SLA that can be used to set an agreed-upon level of availability and to include penalties for noncompliance with the agreement. Also, if a service provider can build into its applications contingencies such as exception handling when an invoked service is not available (dynamically locating another source for the needed service), availability would not decrease and could actually be improved as compared with other architectural approaches.	**Yellow**
Usability	Usability may decrease if the services within the application support human interactions with the system and there are performance problems with the services. It is up to the services users and providers to build support for usability into their systems.	**Yellow**
Security	The need for encryption, authentication, and trust within an SOA approach requires detailed attention within the architecture. Many standards are being developed to support security, but most are still immature. If these issues are not dealt with appropriately within the SOA, security could be negatively impacted.	**Red**
Performance	An SOA approach can have a negative impact on the performance of an application due to network delays, the overhead of looking up services in a directory, and the overhead caused by intermediaries that handle communication. The service user must design and evaluate the architecture carefully, and the service provider must design and evaluate its services carefully to make sure that the necessary performance requirements are met.	**Red**
Scalability	There are ways to deal with an increase in the number of service users and the increased need to support more requests for services. However, these solutions require detailed analysis by the services providers to make sure that other quality attributes are not negatively impacted.	**Yellow**
Extensibility	Extending an SOA by adding new services or incorporating additional capabilities into existing services is supported within an SOA. However, the interface/formal contract must be designed carefully to make sure that it can be extended, if necessary, without causing a major impact on the service users.	**Green**

Table 1: SOA Approach's Impact on Quality Attributes (cont'd.)

Quality Attribute	Summary	Status
Adaptability	The use of an SOA approach should have a positive impact on adaptability, as long as the adaptations are managed properly. However, the management of this quality attribute is left up to the service users and providers, and no standards exist to support it. This attribute must be managed in coordination with other quality attributes including stability, performance, and availability, and the necessary tradeoffs must be identified and made.	Yellow
Testability	Testability can be negatively impacted when using an SOA due to the complexity of the testing services that are distributed across a network. Those services might be provided by external organizations where access to the source code is not available, and if they implement runtime discovery of services, it may be impossible to identify which services are used until a system executes. It is up to the service users and providers to test the services, and very little support is currently provided for the end-to-end testing of an SOA.	Red
Auditability	Auditability can be negatively impacted if the right capabilities for end-to-end auditing are not built into the system by the service users and if capabilities are not incorporated into the services by the service providers.	Red
Operability and Deployability	Operating and deploying services and systems that use services need to be managed carefully to avoid problems. The interactions and tradeoffs among this and other quality attributes need to be monitored and managed.	Yellow
Modifiability	Modifiability of services or an application that uses services is directly supported using an SOA approach. However, a service interface must be designed carefully because the changes that impact service users might be difficult to identify if the service is externally available.	Green

5.2 SOA's Impact on Business Goals

Specific business goals propagate required quality attributes; for example, the business goals of agility and being first to market may mean that adaptability, interoperability, scalability, and extensibility become the major drivers for the architecture of the application and the development of the services. Agility will require that services can be adapted and reused in new applications, combined in new ways, and scaled to meet increasing demands for the functionality within the application.

Many vendors are selling technology that supports the development of applications using an SOA approach. Choosing a technology—and a corresponding vendor to partner with—is an important decision for an organization. That decision becomes even more important if the

organization is streamlining its process across multiple partners that may be using different technologies. Interoperability and security considerations may become the driving forces for choosing which technology and standards are needed.

Performance, reliability, and testability may be the main architectural drivers if a key requirement for the organization is to develop services based around its legacy systems. Whatever the business goals, it is important to understand both the quality attribute requirements they necessitate and realistic characterizations of those requirements. The information provided in Section 4 offers helpful guidance in using an SOA approach in the face of specific business goals.

6 Conclusion

Software architecture is the bridge between mission/business goals and a software-intensive system. Quality attribute requirements drive software architecture design. Choosing and designing an architecture for such systems that satisfies the functional as well as the nonfunctional or quality attribute requirements are vital to the success of those systems. An SOA is an architecture for a system that is built using a set of services. It is import to examine how an SOA supports quality attributes. This report examines several quality attributes, explores how an SOA supports them, and identifies issues and problems related to them. The SEI is investigating how this information impacts the DoD and its use of SOAs—a topic that will be the subject of future reports.

When an organization designs and implements its application using an SOA approach, it must make various tradeoffs among the quality attribute requirements that will affect whether the organization will fully meet its business goals. SOA is still an emerging technology, many of the issues between SOA and quality attributes have not been thoroughly researched, and many of the standards posing as remedies are immature.

If external services, or even services outside the control of the development department, are used, an SLA must be established between the various parties to guarantee quality of service for a set of essential services. Building a system that relies on third parties without the necessary agreements in place will be detrimental to the system's success because, as a result, it may be difficult to meet the system's quality attribute requirements. Failing to achieve those requirements would adversely affect an organization's ability to meet its business goals.

References

URLS are valid as of the publication date of this document.

[Alberts 99] Alberts, D.; Garstka, J.; & Stein, F. *Network Centric Warfare: Developing and Leveraging Information Superiority, Second Edition.* Washington, DC: National Defense University Press, 1999 (ISBN 1-579-06019-6).

[Atkinson 02] Atkinson, Bob, et al. *Specification: Web Services Security (WS-Security), Version 1.0*. http://www-128.ibm.com/developerworks/library/ws-secure/ (April 2002).

[Bass 03] Bass, Len & John, Bonnie. "Linking Usability to Software Architecture Patterns Through General Scenarios." *Journal of Systems and Software 66*, 3 (June 15, 2003): 187-197.

[Brownsword 04] Brownsword, L. et al. *Current Perspectives on Interoperability* (CMU/SEI-2004-TR-009, ADA421613). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tr009.html

[Bruce 04] Bruce, Ian. *Thoughts from the Integration Consortium: Dynamic Interoperability and Service-Oriented Architectures, Part 2*. http://www.dmreview.com/editorial/dmreview/print_action.cfm?articleId=1008062 (August 2004).

[Cherinka 05] Cherinka, R.; Miller, R.; & Smith, C. "Beyond Web Services: Towards On-Demand Complex Adaptive Environments," 815-816. *2005 IEEE International Conference on Web Services: (ICWS 2005): Proceedings*. Orlando, Florida, July 11-15, 2005. Los Alamitos, CA: IEEE Computer Society, 2005.

[Clements 02] Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures*. Boston, MA: Addison-Wesley, 2002.

[Cohen 05] Cohen, Frank. *Web Services Scalability and Fragmentation*. http://www.pushtotest.com/thecohenblog/wsscaleandfrag.html (2005).

[Erl 05] Erl, Thomas. *A Look Ahead to the Service-Oriented World: Defining SOA When There's No Single, Official Definition.* http://weblogic.sys-con.com/read/48928.htm (April 2005).

[Gall 03] Gall, Nick & Perkins, Earl. *The Intersection of Web Services and Security Management: A Service-Oriented Security Architecture.* http://whitepapers.ostg.com/detail/RES/1066934618_724.html (July 2003).

[IBM 05] IBM. *Autonomic Computing*. http://www.research.ibm.com/autonomic/ (2005).

[IEEE 90] IEEE Computer Society. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, 610*. New York, N.Y.: Institute of Electrical and Electronics Engineers, 1990.

[Koch 03] Koch, Christopher. "The Battle for Web Services." *CIO 17*, 1 (October 2003): 54-63.

[Litoiu 02] Litoiu, M. "Migrating to Web Services – Latency and Scalability," 13-20. *Proceedings Fourth International Workshop on Web Site Evolution (WSE 2002)*. Montreal, Quebec, Canada, October 2, 2002. Los Alamitos, CA: IEEE Computer Society, 2002.

[Little 03] Little, Mark & Freund, Thomas J. *A Comparison of Web Services Transaction Protocols*. http://www-128.ibm.com/developerworks /webservices/library/ws-comproto/ (October 2003).

[McGovern 03] McGovern, James; Tyagi, Sameer; Stevens, Michael; & Matthew, Sunil. *Java Web Services Architecture*. San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[Microsoft 05] Microsoft. *Core Principles of the Dynamic Systems Initiative*. http://www.microsoft.com/windowsserversystem/dsi/dsicore.mspx (2005).

[OASIS 04] OASIS. *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*. http://docs.oasis-open.org/wss/2004/01 /oasis-200401-wss-soap-message-security-1.0.pdf (March 2004).

[Ogbuji 02] Ogbuji, Uche. *The Past, Present and Future of Web Services, Part 2*. http://www.webservices.org/ws/content/view/full/2455 (2002).

[SEI 05] Software Engineering Institute. *Software Architecture Technology Initiative*. http://www.sei.cmu.edu/architecture/sat_init.html (2005).

[W3C 04] Worldwide Web Consortium (W3C). *Web Services Glossary*. http://www.w3.org/TR/ws-gloss/ (February 2004).

[Weerawarana 05] Weerawarana, Sanjiva; Leymann, Frank; Curbera, Francisco; Ferguson, Donald; & Storey, Tony. “Chapter 3: Web Services: A Realization of SOA.” *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.

[Wilkes 04] Wilkes, Lawrence & Veryard, Richard. *Service-Oriented Architecture: Considerations for Agile Systems*. http://msdn.microsoft.com/library /default.asp?url=/library/en-us/dnmaj/html/aj2service.asp (April 2004).

[WS-I 04] Web Services-Interoperability Organization (WS-I). *Basic Security Profile Version 1.0 – Working Group Draft*. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html (May 2004).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2005	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Quality Attributes and Service-Oriented Architectures	FA8721-05-C-0003

6. AUTHOR(S)

Liam O'Brien, Len Bass, Paulo Merson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2005-TN-014

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report examines the relationship between service-oriented architectures (SOAs) and quality attributes. Because software architecture is the bridge between mission/business goals and a software-intensive system, and quality attribute requirements drive software architecture design, it is important to understand how SOAs support these requirements. This report gives a short introduction to SOAs and outlines some of the main business goals that may lead an organization to choose an SOA to design and implement a system. This report outlines a set of quality attributes that may be derived from an organization's business goals and examines how those attributes relate to an SOA. In addition, this report describes how the SOA impacts those attributes and how choosing an SOA can help an organization achieve its business goals.

14. SUBJECT TERMS	15. NUMBER OF PAGES
service-oriented architecture, quality attributes, software architecture	38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL