2021

The Strategic Goals Of -Using Service Oriented Architecture Principles

Ashraf Saied Abdel Nabi

Faculty of Commerce & Business Information System, Helwan University,
ashraf.saied@commerce.helwan.edu.eg

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer and Systems Architecture Commons](#), and the [Management Information Systems Commons](#)

Recommended Citation

Abdel Nabi, Ashraf Saied (2021) "The Strategic Goals Of -Using Service Oriented Architecture Principles," *Future Computing and Informatics Journal*: Vol. 6: Iss. 2, Article 6.

DOI: <http://doi.org/10.54623/fue.fcij.6.2.6>

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol6/iss2/6>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

Future Computing and Informatics Journal

Volume 6
Issue 2 (2021) *Issue 2*

Article 6

2021

The Strategic Goals Of -Using Service Oriented Architecture Principles

Ashraf Saied Abdel Nabi

Faculty of Commerce & Business Information System, Helwan University,
ashraf.saied@commerce.helwan.edu.eg

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer and Systems Architecture Commons](#), and the [Management Information Systems Commons](#)

Recommended Citation

Abdel Nabi, Ashraf Saied (2021) "The Strategic Goals Of -Using Service Oriented Architecture Principles," *Future Computing and Informatics Journal*: Vol. 6 : Iss. 2 , Article 6.

DOI: <http://doi.org/10.54623/fue.fcij.6.2.6>

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol6/iss2/6>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

The Strategic Goals of -Using Service Oriented Architecture Principles

Ashraf Saied Abd El-Naby^{1, a}, Yehia M. Helmy^{1, b}, Emadeldin Helmy Khalil^{2, c}

¹ Business Information Systems, Faculty of Commerce and Business Administration,
Helwan University, Egypt

² Cyber Risk Bureau Head in National Telecom Regulatory Authority, Egypt

^a ashraf.saied@commerce.helwan.edu.eg, ^b ymhelmy@yahoo.com, ^c ehelmy@tra.gov.eg

ABSTRACT

Organizations use heterogonous systems and legacy systems, they are implemented at different platforms, different databases, and different language programs. These systems need to exchange information and reuse the same functionality to achieve integration between these systems. Many Software companies failed to achieve information follow and reuse the same functionality. This paper introduces guidelines introduces Service-Oriented Architecture Principles guidelines and rules to help Applications developers to achieve information integrations and reuse the same functionality, SOA principles providing rules and guidelines that specify exactly how solution logic should be decomposed and molded into technology solutions

Keywords: Service-Oriented Architecture principles, Business and Technology Domain Alignment, Service Contract Standardization, Organizational Agility, Commercial Design Contemplations

1. INTRODUCTION

Service-Oriented Architecture (SOA) is at the heart of a revolutionary computing platform that is being adopted world-wide and has earned the support of every major software provider. In Service-Oriented Architecture: Concepts, Technology, and Design [1]

In any event, the majority of companies still have legacy inherited programs that were not built for integration. Organizations cannot afford to take down or replace them overnight since they are mission vital, nor can they afford to create their complete data systems from start in today's business environment.

These apps are often built on a variety of platforms, technologies, and programming languages. [2] .

A service-oriented architecture (SOA) is a set of ideas and methodologies for creating and building interoperable software services. change operational behavior and policies, such as the implementation of new business rules and laws, and in the kind of business-related events [3].

SOA logic is decomposed into smaller, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed [1].

A service-oriented architecture (SOA) is an architectural approach in computer software development in which application

components deliver services to other application components via a communication protocol, generally through a network. Adoption of the notion of services - removing the greatest degree of abstraction independent of the application or infrastructure platform access technology, context, or other services The service transports IT to another location. In this part, we present the SOA Design Principles that are fundamental to a proper understanding of services and SOA.

2. STANDARDIZED SERVICE CONTRACT

Contract standardization One of the fundamental components of SOA is standardized service contracts. They guarantee that the positive outcomes of services in the same service inventory stay compatible with international contract design rules. During the SOA, services are able to declare their capabilities as well as the general objective of the contract. [4]

A service contract is made up of one or more public contracts that provide Meta information about a service. A technical service contract of this sort develops an API for the functionality supplied by the service via its capabilities. WSDL definition, XML schema definition, and WS policy definition are the most typical service description papers in Web services. A service contract may additionally include human-readable documentation, such as the "SLA" Service Level Agreement, which specifies extra quality-of-service characteristics.

2.1 Type of Contract

Technical contracts are a well-established aspect of the IT industry. A technical contract is necessary whenever two programs or two pieces of programming logic need to communicate.

2.1.1 Technical Contracts

A service contract (or contract for a service) sets the conditions of engagement, including technical limits and requirements

as well as any semantic information that the service owner decides to make public.

The SOA service contract is a design document, collecting all the functional information and professional business that determines its scope and use situations, the structure of the message you use, the details of the information that must be transmitted in each node of the message, its sequence diagrams, which demonstrates the typical communication pattern of that service (synchronization, asynchronous, sequence of events/messages, etc.), functional errors that may occur, and the number of references necessary for design rules developed by the management office.

- WSDL definition
- XML schema definition
- WS-Policy description

2.1.2 Non-Technical Service Contract Documents

A service contract Non-Technical Service Contract called service level agreement (SLA) is a formal contract between a service consumer provider and a service provider and formally defines the level of service that focus on (Availability , Reliability , Performance , Mentoring Network , Response time , Traffic Data per hour – minute- second , Down Time and Quality Information) [5]

2.2 Types of Service Contract Standardization

The standardized service contract is a software design principle used within the service-oriented design paradigm to ensure that all service contracts in a service inventory (enterprise or domain) comply to the same set of design criteria. This makes it easier to create consistent service contracts throughout the service inventory. A typical technical service contract includes a WSDL document, XML schema(s), and policy document (s).

2.2.1 Functional expression standardization

The service's operations must be defined using standardized naming conventions; this also applies to the constituent input and output message names and their corresponding type names. This aids in extending the service contract's correct interpretation, which increases service reuse and interoperability. When service contracts state their capabilities clearly, the possibility of service duplication is limited even more.

2.2.2 Standardization of Service Data Representation

The technical interface description that serves as the basis of any service contract will sometimes include a formal definition of the input and/or output data required by each service capability.

Two services that exchange messages based on the same type of data exemplify a sales order — may model that data consistent with different schemas that require a transformation of a data model.

2.2.3 Standardization of Service Policies

The Service Regulations address the scope and application of help. As a result, in order for the aid to be reusable, the conditions for its use must be properly communicated through institutionalized policy articulations that rely on industry standard terminology [6].

As the two parties begin to communicate, this business policy is deconstructed into a set of objectives, strategies, and tactics that specify how the business need will be executed and enforced throughout the company. The organization of the breakdown of the basic business need into a collection of more comprehensive policies is divided into two main dimensions:

- A horizontal dimension (policy domains and types)
- A vertical dimension (levels of abstraction of policies)

2.3 Data Representation Standardization & Data Transformation Rules

Data Standardization interacts with the transformation of datasets after data is extracted from multiple sources and before it is loaded into target systems as part of the data Preparation field. Data Transformation Rules are a set of computer instructions that dictate consistent manipulations to transform the structure and semantics of data from source systems to target systems. The most popular forms of Data Transformation Rules are Taxonomy Rules, Reshape Rules, and Semantic Rules.

2.3.1 Taxonomy Rules

These rules associate the columns and values of the source data with the columns and values of the destination data. A source, for example, can define its transactions as having two columns: a settlement amount and a type, with the type being one of three alternatives.

2.3.2 Reshape Rules

These rules define how to gather data items from the source side and distribute them on the destination side. For example, a retailer may offer all transaction data in a single file, but the aggregator must separate it into three tables: one for transactions, one for retailer data, and one for customers.

2.3.3 Semantic Rules

These rules express the definition of the data elements and how the business provides them to define its domain, data provider has its own semantics that make sense in the context of its activities, but one that the data aggregator must integrate with the definitions of all other data providers.

3. Loose Coupling

Loose coupling is a method of integrating the components of a system or network in such a way that those components, also known as elements, rely on each other to the greatest extent possible. The term "coupling" might be a very simple aspect of an IT vocabulary contract that is disconnected from its surroundings.

They contain the definition of SOA as well as the idea of loose coupling. This term really refers to how the interconnected software components diminish their in-built knowledge of one another: they discover the information they require when they want it. [3]. This is one of the most characteristic features of web services, which states clearly that there should be with less dependence as possible between web services and, therefore, the client that invokes the web service.

In computing, tight coupling (or close coupling) may be a kind of coupling that defines a system on which hardware and software are not only connected together, but also mutually dependent. [7] .Loose coupling—Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other [1].

Systems don't communicate with each other, and even one system normally has very little knowledge of the other system, which is generally limited to details about

mutual interfaces, named, loose coupling, and tight coupling. [7] .Within the service orientation design model, service loose coupling can be a design concept that refers to services in order to ensure that the contract is not closely connected to the service users and to the underlying business logic and execution.

This principle emphasizes the reduction ("loosening") of coupling between the parts of a service-oriented solution, especially in comparison to how applications have traditionally been designed.

3.1 Service Contract Coupling Types

Coupling refers to a connection or relationship between two things., The measure of the coupling is proportional to the degree of dependency, this concept encourages the formation of a different form of relationship inside and outside the service boundaries, with a constant focus on reducing ('loosening') the dependency between the service contract, its implementation and its service customers.

3.1.1 Logic-to-Contract Coupling

A recommended approach to putting together a service is to style its physical contract before its underlying solution logic, It also allows us to tune the underlying logic in support of the contract, which may optimize runtime performance and reliability.

Figure 1: Tight coupling & Loose coupling

3.1.2 Contract-To-Logic Coupling

This is because the service contract is dictated by the underlying logic rather than established separately according to design principles.

Many contracts for Web services, in particular, are derived from existing solution logic. This reverses the coupling dynamic, as once these types of contracts inherit existence, they are instantly tied into the underlying logic and implementation. Contract-to-logic coupling refers to the contract's reliance on its underlying logic, as seen in the following diagram.

3.1.3 Contract-To-Technology Coupling

Service developed as a proprietary component may require that the contract exist as a proprietary extension of the service, coupled by an implementation technology contract which, in effect, induces the requirement for each service consumer to accept an equivalent non-standard protocol.

3.1.4 Contract-To-Implementation Coupling

For example, service users would not have access to the current service through automated proprietary interfaces even before it actually existed as a service. Contracts must not be connected to implementation functionality, although it is relatively common (and acceptable) for a few types of service logic to be connected and linked to other implementation technology and to elements outside the core service logic.

3.1.5 Contract-To-Functional Coupling

Services built to enable communication with a business partner or a service that executes a part of the business process logic or is itself the parent controller service in a service composition that executes the business process logic.

4. SERVICE ABSTRACTION

This principle concerns the abstraction of multiple service specifics so as to minimize the interaction between consumers of services and therefore the implementation of services. Additionally, these abstractions make it possible for service designers to switch the library or algorithm in question without impacting other parts of the service, which successively encourages the development of the service logic.

Figure 2: publish Necessary Service information

The principle of abstraction helps to avoid the publication of unnecessary service information, meta or otherwise. Transform service to a black box by hiding information structures, logic processing, and programmatic approaches. (Java, ABAP etc).

4.1 Types of Meta Abstraction

The application of this design principle requires looking into four different types of abstractions that could be applied to a service.

Figure 3: Common Types of Service Meta-Information

When it comes to using this approach, we are interested in the following forms of meta information:

- Information on Technology Metadata describing the technical implementation of the underlying service logic.

- Useful Information Metadata describing the service's capabilities.
- Programming logic knowledge Metadata that explains how the service performs its functions.
- Information on service quality Metadata that explains the behavior, restrictions, and interaction needs of a service.

4.1.1 Technology Information Abstraction

Try to not tell clients how service logic and implementation are structured. Conceal subtleties of technology used to fabricate a software program all at once that we hold opportunity to shape specialized changes without influencing existing clients.

4.1.2 Functional Abstraction

Abstract function works by allowing the client to call a function coded by the "programmer" implementer without actually seeing how it is implemented. Functional abstraction offers tools and methods to try to perform some kind of job (what) while avoiding specifics of how this is always achieved

4.1.3 Programmatic Logic Abstraction

Programmatic logic abstraction refers to the internal details of a few programs that are consciously hidden from the surface world. Those involve specifics such as low-level architecture, algorithms, exception handling and logging routines. [8], [9].

4.1.4 Quality Of Service Abstraction

Quality of service data is an extended form of behavioral, rule-based, and reliability-related meta-information about a service. Information on how the service integrates into the overall business cycle and which other tools it uses to fulfill its features. Information of how a company works within the general business cycle and which other services it uses to meet its functionality.

5. SERVICE AUTONOMY

Services have extensive control over their underlying runtime execution environment. Services should be able to control the logic that they enclose. To be considered autonomous, a service must be as functionally and runtime environment independent as possible from other services with whom it interacts. [10].

Figure 4: Service autonomy

Speaks to the opportunity and control to take on its own choices without the requirement for outer approval or association with which a program can perform its logic.

5.1 Types of Service Autonomy

Service autonomy involves two types of autonomy that allow an increase the overall autonomy of the service, design time autonomy and run time autonomy.

5.1.1 Levels Of Autonomy

Service-oriented systems, being supported the principles of service-oriented architecture (SOA), are a current trend in software architecture. Monitoring information on current service performance in reference to requirements are often wont to automatically provision services, adapting reaction time, availability and throughput. **Levels Of Autonomy are 4 types**

1. Service Contract Autonomy

Service contracts are designed in alignment with each other to avoid overlap of expressed functionality.

Table 1 Service autonomy

	Run-time autonomy (Execution)	Design-time autonomy (Governance)
definition	Level of control a service has over its processing logic at the time the service is invoked and executing.	Level of freedom we, as service owners, have to make changes to a service over its lifetime (Regardless of whether a service has control over its runtime execution environment)
Objectives	-Consistently acceptable runtime execution performance -Greater degree of performance reliability - isolated in response to specific security, reliability, or performance requirements - Greater level of behavioral predictability (especially when concurrently accessed)	-The ability to scale a service in response to higher usage demands - the option to further modify or enhance a service's hosting environment - the freedom to augment, upgrade, or replace the technology of a service in response to new requirements or a desire to leverage new innovation All of forms of design-time autonomy can be attained by applying Service Loose Coupling principle

2. Shared Autonomy

It is impractical for many companies to build all services from the ground up, some systems are often custom designed, while others need to encapsulate older legacy technology to imply that other sections of the business are required to access (and maybe even compete for) whatever processing logic might fall within the service boundary.

3. Service Logic Autonomy (Partially Isolated Services)

Service logic autonomy is perhaps the most common standard reached when designing new custom services. it's considered a type of partial autonomy since databases, directories, and other resources are still shared among services and other parts of the enterprise

4. Partially Isolated Services at Design-Time

Since distributed systems inherently abstract the info access layer into their own physical tier (usually specified by one or more database servers), we may assume absolute control over custom components of the service in design-time.

5. Pure Autonomy (Isolated Services)

The right implementation environment for a service is one where its runtime life is fully managed and where we have top-to-bottom governance over its architecture and design. In particular, the service is delivered with a higher quality of service guarantee

– Functional Isolation

Service components and physical data models are dedicated, but the service is hosted on a server with others

Isolated Services at Design-Time (Absolute pure autonomy)

We have complete control of their architecture and hosting environments, and even their data model is typically increased in relative isolation. When marking a service with a measure of autonomy, we would like to ensure that this measure is representative of the service as a whole. The overall autonomy of certain services is reasonably easy to calculate, since they are part of a custom-developed solution structure where all the service capabilities enjoy an equal set of underlying resources.

5.2 Service Autonomy & Service Model

Service Autonomy is applied to the analysis of services additionally to their physical design (Service Re usability and repair Discoverability being the opposite two).

The parent service-oriented analysis process contains an operation step dedicated to identifying systems related to the automation of a specific business process.

5.2.1 Service Region Influence - Autonomy

The Autonomy principle is nearly exclusively focused on the service implementation, with a stress on the Core Service Logic.

The contract may be affected in some cases, too.

6. SERVICE REUSABILITY

The principle of service reusability can be a design concept applied to the generation of services within the framework of the market orientation architecture which will be reused in the industry. Reusability of Resources-Concept is divided into resources with the goal of optimizing reuse.

Definition of the re-use service as having the capacity to communicate in several service assemblies (compositions). Service reuse is closely connected to software composability [11].

Figure 5: Service Reusability

Reusability of the service is a design philosophy to be remembered during the design of the service as well as an essential goal of the SOA. Whereas something that is useful for a single purpose will provide value, something that is repeatedly useful will provide a repeat value and is therefore a more attractive investment.

6.1 Service Reusability Goals

The theory of reuse of the service by trying to compare the design of two applications required to give the same functionality at present. While something that would be useful in one go will provide value, something that is useful again and again will bring repeated value and thus make an investment more attractive.

1. Single-Purpose Programs

Creating a single software system helps us to focus only on a very specific set of needs. In order to achieve its single target and the consistent use situation it would have to promote, everything in the system can be optimized and personalized.

2. Multi-Purpose Programs

Not only one goal. For starters, we will establish how it is to be used in multi-use scenarios when developing a multi-use system. The development is more complicated and leads to an increased effort to accommodate all the scenarios associated with the variety of expected capabilities.

6.1.1 Origins of Reuse

Any software system ever designed for general release with a view to reuse has been installed. The considerations we described in the last section, regardless of whether it is an operating system, a shrink-wrapped accounting product or a full middleware platform, in the initial design phases of these programmes.

6.1.2 Estimating Service Reusability and Applying Commercial Design

Commercial design requirements are similar to current service-oriented research and design processes in that they are extracted from established business product development lifecycles and are essential in planning, developing and eventually implementing reusable resources as services.

6.1.2.1 Commercial Design Contemplations (Considerations)

The purpose of the commercial design approach is not to guarantee absolute re-use, but rather to apply expert judgment when trying to determine, Apply judgement to determine:

Measures of Planned Reuse

To improve the consistency and availability of functions and data, computers can be used to centralise logics that can be accessed within and around the enterprise. Categories such as the one below can be used to describe or identify programmes in the research and design processes [12].

1. Reusability Of Tactical

The environment of the application is divided into smaller, separate components that reveal their functionality to other services applications. Focus is put on designing business operations to meet immediate needs The company is needed.

2. Reusability Targeted

When a project team has the opportunity to provide a software framework that offers features beyond what would be currently needed, this strategy incorporates only extensions that have the highest degree of guaranteed reuse potential. The aim of targeted reusability is to carry out programs

which have the highest potential for reuse in more than one scenario.

7. SERVICE STATELESSNESS

Services decrease resource usage and reduce state details. Ideally, the service should be stateless. Service Statelessness Programs will ideally be stateless. This ensures that services should not hold information from one state to another. You can now get a web service that offers you the price of a given object.

Figure 6: Service Role -Statelessness

Objectives Service Statelessness is designed to optimize service scalability, and services and their infrastructure are also designed to enable delegation and defer State management responsibilities.

7.1 Types of State Conditions

Each state can be specified and characterized by data typically having a lifetime equal to the duration of time the system remains active for a given reason or task. The following State conditions and data types can exist:

1. Active and Passive Primary State

As defined in the State as an example State, Active and Passive refers to the general condition of something. A car that is moving is in a state of motion (active), while a car that is not moving is in a stationary state (passive).The software programme may also have a transition between various states, typically due to its participation in a runtime activity.

Active: service being invoked or executed and therefore entering an active state.

Passive: Period during which service is not in use. Exists in a passive or [non-active] state.

2. Conditions of Primary State (Stateful , Stateless)

While in Stateless mode, the server stores data in a database to validate the user/client everytime it wants to connect. In Stateful, the server considers a client to be a dumb computer, whereas in Stateless, the server considers the client to be an intelligent computer that does not require any server-side state.

Stateful State: Service that is actively processing or retaining state data

Stateless State (idle condition): Active service but may not be engaged in processing of state data as example : http protocol when server respond to requested web page

3. State Information Type (Session / Business Context)

State data is information mainly correlated with current events, and state management is responsible for handling this information.

Session: Represents details related to the preservation of a relation between the programme and its client programme as an example of a web site session

Context: Information about a particular service activity, the larger complex a service composition, the more context information will generally need to be managed, Types: context data and context rules (work flow rules)

- **Context data** information within service and seen as part of the current service activity as an example: quantity of services actually involved in the activity, which services are currently active and which were active in the service activity, length of the service activity, how many instances of the activity are currently being carried out.
- **Context Rule:** Protocols and restrictions applicable to the execution of a particular [Service

Activity Rules for Workflow Regulations for the Processing of Activities] as an example (Acceptable Length of Service Operation, Allowable Quantity of Service Activities, Allowable Quantity of Participating Services).

- **Business:** information relevant to the business activity currently being worked out. Example records return from database query stored in memory for future needs

7.1.1 Deferral vs. Delegation

Table 2 Deferral vs. Delegation

Deferral	Delegation
The temporary relocation of state information is referred to as state deferral	To accomplish state management deferral we temporarily delegate this responsibility to another part of the architecture (such as a database). Therefore, we achieve state management deferral through temporary and periodic state management delegation.

The temporary transfer of state information is known as a state postponement, as the purpose is to get information later. We temporarily delegate this responsibility to another architectural section (such as the database) to perform state management postponement. We therefore obtain a postponement of policy administration through temporary and intermittent delegations of state management.

7.1.2 Types of Deferred State Management

There are 5 Management Deferral Sample Types, we will illustrate each in the next section:

1. Non-Deferred State Management (low-to-no statelessness)

A service functionality may be configured to delay state data without the need to switch between stateless and stateless situations (low-to-no statelessness).

2. Partially Deferred Memory (Reduced Statefulness)

Service ability can be configured to delay state data without the need to switch between stateless and stateless situations. If a programme is expected to collect large quantities of state data at runtime, it can be programmed to unload parts of this data at times when data is not required.

3. Deferral of State Partial Architectural Management (Moderate Statelessness)

Although the service will be converted into stateless modes during these periods of inactivity, it is not intended to take advantage of every chance to become stateless. (In longer-term activities, the service will be changed into stateless modes during these inactivity intervals., At times, it takes advantage of the stateless, Not intended to utilize every chance to become stateless)

4. Full Architectural State Management Deferral (High Statelessness)

Service technologies are designed to maximize any logical opportunity to become stateless, and the ability to unload state information (primarily context and business data) when state-of-the-art wherever possible is also leveraged. (Services are designed to exploit any legitimate capacity for statelessness, state (primarily context and business information) is not loaded even if it is stat.

5. Internally Deferred State Management (high statelessness)

The highest degree of independence of full autonomy [the service system is independent and firmly under our control] has been achieved. This is usually achieved by a dedicated database that the provider

may use to store and retrieve data in temporary operations.

8. SERVICE DISCOVERABILITY

The principle of discoverability should be applied to the design of a private service in such a way as to make it as discoverable as possible if the extension of discoverability or product is actually taking place within the current setting of implementation. SOA Discoverability is intended to help us avoid the unplanned creation of a service that is either redundant or unnecessary.

SOA Discoverability is meant to assist one prevent the unplanned development of a service that's either redundant or a logic that's redundant.

Figure 7: Service Discovery Information

The principle of service discovery refers to the need for the availability, publishing, accessibility and information of our SOA Services Catalog with a set of meta data which allow us to conduct rich searches to determine services that can be reused.

Discoverability of services can be found (usually in the registry of a service). This has been seen already in the UDDI definition of a registry that can hold web-service information.

8.1 Discovery Requires

The main method involved in the success of Service Discovery is the service registry, which includes appropriate information on accessible and expected services, as well as references to relevant service contract documents that may include SLAs. In modern architectures, nodes come and go and you need to decouple individual service instances from the knowledge of the deployment topology of your service. Service discovery is the method of finding and retrieving web service provider specifications that were previously published.

8.2 Interpretability Process

It is the system designers and developers that are expected to work with or construct new parts of the enterprise that are most in need of a means to find what already exists. In order for anything inside the enterprise to be discoverable, it must be fitted with meta-information that will enable it to be included in the scope of discovery searches.

1. Service Registry

Service-oriented architecture registry (SOA registry) may be a database that sets out the info access rights needed for service-oriented architecture projects, the SOA registry allows service providers to get around and communicate easily with users, provides a link between service providers and repair customers, and the service registry is used to identify procedures, systems, accidents and others.

When a business focuses the service register at the heart of its infrastructure, it should create a structured enterprise system for finding, retrieving and interpreting on-demand meta-data services. Main reason for the introduction of discovery and interpretability Architecture features help to establish structured discovery and understanding processes based on the use of a service register.

Figure 8: Discovery and interpretation in action.

Figure 9. Service Decomposition

8.3 Universal Description, Discovery, and Integration (UDDI)

UDDI (Universal Definition, Discovery, and Integration) is an XML-based business registry that lists itself worldwide on the Internet. Its primary aim is to streamline electronic transactions by allowing businesses to locate each other on the Web and to interoperability their e-commerce systems. UDDI also compares with white, black, and green telephone book lines.

8.4 Service Inventory & Service Registry

There are important terminology that helps us differentiate the most common discovery and interpretability approaches and meta-information styles.

1. Design Time Discovery:

Manual process of discovery by human:

Requires

- Central discovery mechanism (service registry)
- Service contracts

2. Runtime Time Discovery

- Discovery technologies like UDDI
- Provide programmatic interfaces into service registry repositories.(dynamic discovery queries)

9. SERVICE COMPOSABILITY

Composition of services means the creation of new services by the integration of existing services. Services must be composable in order to integrate smaller services with larger value-added services. Composability integrates design principles to ensure that programmers are able to communicate in a number of variations in order to solve a number of biggest problems.

9.1 Service Decomposition

Services are breaking big problems into little problems. One can never integrate all the functionality of an application into a single service, but instead break down the service into modules each with a separate business functionality.

Separation of concerns approach enables us to break a larger problem Through several smaller problems (concerns).

9.2 Service Composability

Service composition is a collection of services where, many smaller services are combined together to a larger service.

Figure 10: Service Composability

Units are gathered in a specific sequence in order to solve a big problem, so that they can coordinate their solution logic.

9.3 Service Re-Composition

The theory of separating issues facilitates us to break up a bigger problem into several smaller issues (problems). This gives us the chance to construct appropriate logic solutions, each of which solves a specific problem. As a result, the same logic for solving new problems can be recreated again and again.

Service Re-composition is break big problem into smaller problem to solve this issue as The same units originally created to solve Big Problem A are recomposed

9.4 Composition Concepts and Terminology

Compositions integrate different terminology and definitions which must be specifically incorporated in the exact requirements of composition design. This next section describes and discusses the following terms:

1. Compositions and Composition Instances

A service composition is usually related to the automation of a business process.

Figure 11: Service Composability

Therefore, it is often helpful to differentiate between a static business process definition (comprised of workflow logic) and a business process instance that represents what parts of the workflow logic actually occurred at runtime

Service Composition Automation of a business process (Static business process definition)

Composition instance: what happens when an occurrence of the workflow logic is carried out by a series of service instances at runtime.

Services can perform different roles depending on how they are configured within the total composition configuration when participating in compositions. It occurs as the implemented program function includes and actually carries out functions of certain

2. Composite Controller:

The Control capacity System is situated at the top of a hierarchy of structure.

Composition Member Composition
member capability service being composed
by Another.

3. Composition Sub Controller & Designated Controller

As described in the Service Composition and Service Models section, the controller role can just as easily be assumed by an entity service as it can by a task service.

Composition sub Controller:
Composition members that compose other services

Designated Controllers : Service limits to controller role only

4. Service Activities

So we can model a composition, we need to develop a means of mapping the flow of data and processing via a composite context. Purposely a service activity is limited to representing only interaction between the services. A service operation does not reflect what takes place within the fundamental structure of the service. Service activity include :

- Mapping of an inter-service message path
- Representing interaction between services only
- Does not represent what occurs within the underlying service logic

5. Composition Initiator

The scope of the service composition is often not a mapping of the corresponding service activity. A runtime parts that exists outside of this boundary is usually represented by a service consumer program responsible for imposing the composition controller in order to start the service composition.

6. Point to Point Data Exchange

Although we are mainly interested in models that integrates various services into actual compositions in relation to the

Service Composability principle, it is important to recognize that the point-to-point model represents the vast majority of past service implementations, particularly for services built as web services.

10. SERVICE INTEROPERABILITY

Services should employ standards that allow a wide range of subscribers to use them. To guarantee that online services comply to this, standards such as XML and HTTP communication are utilized. The foundation of the SOA method is to describe and expose multiple units of solution logic as services. SOA design principles are industry-accepted, defined methodologies for deciding how solution logic should be deconstructed and molded into software services.

CONCLUSION

The study contributes to explaining benefits, its stages of development, and the principles it follows when integrating ERP applications using the web service.

The study contributes to explaining SOA, its stages of development, and the principles it follows when integrating ERP applications using the web service.

Service Oriented Architecture Principles
Implementation of the Capacity to Help Build Sustainable, Capacity to Build New Feature Combinations Rapidly, Capacity to Incorporate Existing Software, Data Privacy and Integrity, Cost Reduction for Software Developing, Capacity to Maximize Efficiency, Functionality and Cost, Easier Implementation of System Updates, Improved information flow

The main objectives of the SOA principles

This study contributes to explaining that, The SOA is an essential guideline to design applications which use different languages like PHP, JAVA, VB NET ... etc, using different databases like sql server, oracle and different OS can be implemented by using SOA principles

Implementation of a good SOA is much more than automation solutions. In the scope of the SOA, the principle defines the context within which customers and services could communicate and serves as a general guide for the design and implementation of services [13] .

The performance of software development based on any specific design model was not guaranteed. In order to ensure that the step towards service orientation is a positive change that delivers on its expected benefits, it is helpful to follow a set of guidelines SOA principles [14] .

1- Principles that Increase Intrinsic Interoperability

Interoperability refers to the sharing of data. Therefore, SOA integration can be seen as a process that enables interoperability. A goal of service-orientation is to establish native interoperability within services in order to reduce the need for integration [15].

Interoperability of mission operational software systems is a key element in the area of technology development, both within the organization and in the context of cross-organization collaboration scenarios [16].

2- Principles that Increase Federation

In support of a growing federation, standardization becomes part of the extra up-front focus that each service gets at the time of design. Service-oriented architecture (SOA) applied by web services technology enables standardized protocols for exchanging knowledge across organizational boundaries. As a result of globalization of industry, the service federation seems to be a solution that enables various companies to work together in order to rapidly accomplish their shared objectives [17] , .

3- Principles that Increase Vendor Diversification

Diversification of vendors means a company's ability to pick and use "best-of-breed" vendor products and technological innovations in a single company. By implementing a service-oriented approach in line with but objectively, the major SOA provider platforms and by defining service contracts as standard endpoints across a federated enterprise, individual service implementation features can be abstracted in order to construct a consistent inter service communications framework [18].

4- Principles that Increase Business and Technology Domain Alignment

Traceability from organizational drivers to service-orientation begins with organizational agreement on task, vision, and priorities. IT can not be organized and established separately from the company and the company can not be structured and developed separately from IT. It is really important for a company to understand the market factors, their effect on the corporate vision, purpose, and objective, and the measures that can be used to evaluate them [19].

The extent to which IT business requirements are met is also related to the consistency with which the solution logic communicates and automates business logic. While doing so, common pre-existing representations of business logic may occur as physical services in an implemented form (business entities, business processes). [18].

5- Increase ROI

The extent to which IT business requirements are met is also related to the consistency with which the solution logic communicates and automates business logic. While doing so, common pre-existing representations of business logic may occur as physical services in an implemented form (business entities, business processes). Service-oriented computing supports the development of an agnostic logic solution—a logic that is agnostic for

any one reason and thus useful for various purposes [20] [15].

The introduction of SOAs costs 20 % less and saves 50% more for each reuse. Traditional component-based growth. In addition to regularly observed (but difficult to quantify) improvements in quality and efficiency, the degree of reuse in SOA growth averaged 2.5 times more than non-SOA development, contributing to measurable cost savings in development in the surveyed organizations [21] .

6- Principles that Increase Organizational Agility

SOA may be the best technical approach for increasing organizational agility, increasing flexibility of software and interoperability of processes, and enabling the reuse of legacy resources. Some SOA perceived advantages can only be accomplished at the inter-organizational level through organizational agility (time to market) [22]

Agility, at the organizational level, reflects the efficiency with which an organization can respond to change, especially in the private sector. As a result, the time and effort needed to automate new or modified business processes is correspondingly reduced, as implementation projects can now be implemented with significantly less customized activity. [15].

V - Reduce the Overall Burden of IT

The difficulty and hardness of traditional systems lead in a very two-way misalignment between IT and business. The IT sector is weighed down with the burden of maintaining structures and processes of the 20th century that became too inefficient and wasteful to affect the standards established by the market climate of the 21st century. the maintenance burden approaches 80 % of the total IT budget in many organizations [23].

A distributed system could prevent failure point. A secondary service provider can also be connected to any essential services.

As described above, encoding an application with several features can be time-consuming. On the other side, a well-designed SOA design can dramatically lessen the burden on end developers. The development of applications can therefore be greatly encouraged [24].

Service-orientation is continuously applied to an IT business including cost reductions and duplication, small volume and operating costs and reduced overhead aligned with its governance and growth. Such an enterprise can support an organization by significant improvements in productivity and cost-effectiveness Basically, the success of the objectives mentioned above can lead to a stronger, more flexible IT department[15].

REFERENCES

- [1] T. Erl, M. Ogrinz, N. Bieberstein and A. Karmarkar, SOA_eBook-Inform IT, Pearson Education, Inc., 2009.
- [2] N. Niknejad , W. Ismail , I. Ghani, B. Nazari and M. Bahari, "Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation," Elsevier, January 2020.
- [3] R. I. R. Ibrahim, S. Kadry, K. Akil's and A. Kalakech's, "A New Solution for Service Oriented Architecture Versioning," Asian Journal of Computer Science And Information Technology, 2016.
- [4] C. Vennapoosa, 2007. [Online].
- [5] P. Merson and G. A. Lewis, "Service Level Agreements in Service-Oriented Architecture Environments," ResearchGate, 2008.

- [6] R. Laird, J. Falkl, S. Cocks and D. Clark, "SOA Policy," 08 2018. [Online]. Available: bm.com/developerworks/library/ws-soa-policy/ws-soa-policy-pdf.pdf.
- [7] V. Beal, "loose_coupling," 2003. [Online]. Available: https://www.webopedia.com/TERM/L/loose_coupling.html.
- [8] T. Perumal, C. Leong and A. R. Ramli, "SOA-Based Framework for Home and Building Automation Systems (HBAS)," International Journal of Smart Home , 2014.
- [9] D. Miller, "Abstractions in Logic Programs," University of Pennsylvania Department of Computer and Information Science, 1989.
- [10] T. Erl, A. Tost, S. Roy and P. Thomas, SOA with Java Realizing Service-Orientation with Java Technologies, Arcitura Education Inc., 2014.
- [11] G. Feuerlicht and J. Lozina, "Understanding Service Reusability," ResearchGate, 2014.
- [12] T. Kohlborn, A consolidated approach for service (Master's Thesis), Universitat Munster, 2008.
- [13] J. Bean, SOA And Web Services Interface Design Principles , Techniques And Standard, Morgan Kaufmann publications, 210.
- [14] J. Brodtkin, "Service-orientation design principles," April 2010. [Online]. Available: https://en.wikipedia.org/wiki/Service-orientation_design_principles#cite_note-Pitfalls-.
- [15] T. Erl, SOA Principles Of Service Design, Prentice Hall, 2008.
- [16] M. Sarkarati, M. Merri and M. Spada, "Intrinsic Interoperability Of Services A Dream Or A Key Objective For Mission Operation Systems," Researchgate, 2012.
- [17] A. Bah, P. André, C. Attiogbé and J. Konaté, "Federated Access Control in Service Oriented," Semantic Scholar, 2019.
- [18] T. Erl, P. Merson and R. Stoffers, Service-Oriented Architecture Analysis and Design for Services and Microservices, Mark Taub, 2017.
- [19] The Open Group, SOA Source Book, Van Haren, 2009.
- [20] J. Ricken, PhD: SOA Method Engineering Framework for process oriented and model-driven SOA implementation with underlying models of abstraction, University of Namur, 2013.
- [21] J. Poulin and A. Himler, The ROI of SOA Based on Traditional Component Reuse, LogicLibrary, Inc. , 2006.
- [22] G. Lewi, E. Morris, D. Smith, S. Simanta and L. Wrage, "Common Misconceptions About Service-Oriented Architecture," Springer, 2007.
- [23] C. Stack, "The future is service-oriented architecture," 20006. [Online]. Available: <https://www.networkworld.com/article/2308447/the-future-is-service-oriented-architecture.html>.

- [24] P. Chu, M. Park, S. Chevtsov and J. Wu, "SERVICE ORIENTED ARCHITECTURE FOR HIGH LEVEL APPLICATIONS," Research Gate, 2010.
- [25] M. M. AbdAllah, "A Quick Introduction to SOA," Software Engineering Competence Center, 213.