# Updated Principles of Service Orientation

### Motivation

Current Principles of Service Orientation (SO) were formulated about 15 years ago when Service-Oriented Architecture (SOA) fought its way up into the mainstream of enterprise IT. At that time, the vocabulary of SO was compromised for the sake of mass adoption and, since SOA was rooted in technology, the representation of the principles was tuned to become easily understood by IT specialists.

As a result, thousands of developers worldwide believed that technologies like Web Services were SOA, though no architecture can be technology. Due to popularisation, the SO Principles had been articulated [1] in technology terms or for technology reasons, which defeated the purpose and value of SOA. Such a nuance as ‘architecture is not the same as its implementation’ did not bother IT developers that much.

Since that time, industry has gained a lot of experience in SO and reached a new understanding of what a service is, what orientation on service is and, finally, what a SO Ecosystem is and what it is for [2]. These last 15 years have not been easy:

* in 2006, OASIS issued the first standard – Reference Model for Service Oriented Architecture 1.0 – that preserved the ‘letter and spirit’ of SO, regardless of development habits in technologies
* by 2009, implementation technologies compromised the SO concept so much – promising benefits of excellent business-viable architecture while doing simple standardized integrations with practically no business values at all – that SOA experts agreed that ‘Technology SOA’ dramatically failed and was pronounced “dead” [3].

Nevertheless, SO, especially the idea of a SO Ecosystem, progressed and reached the business domain in 2012. The official OASIS specification of the Reference Architecture Foundation for SOA (OASIS SOA RAF) had defined a SO Ecosystem and stated, “The SO ecosystem described in this document [OASIS SOA RAF] bridges the area between business and IT. It is neither wholly IT nor wholly business, but is of both worlds. Neither business, nor IT completely own, govern and manage this SO ecosystem” [2].

Our understanding of SO has reformed, but the wording of the SO Principles remains the same and has become obsolete, confusing and, obviously, too technology centric. In the following sections of this paper, we will review and update SO Principles where necessary.

### Understanding SO Ecosystem

Service Orientation is understood as a general direction of thoughts, intentions, and willingness of a person (or an abstract entity) or a group of people (entities) to act by providing valuable capabilities and Real World Effect (RWE) to those who need them. The OASIS SOA RAF defines an RWE as follows: “real world effect is the actual result of using a service” [2]. Also, “Service providers offer capabilities that have real world effects that result in a change in state [of a SO Ecosystem]” [2]. A RWE may be direct – returned to the requestor or service consumer, shared – accessible to anticipated users other than the requestor, and shareable – accessible to unanticipated users.

A SO Ecosystem is “a space in which people, processes and machines act together to deliver those capabilities as services inhabit in order to further both their own objectives and the objectives of the larger community” [4]. In a SO Ecosystem, “there may not be any single person or organization that is really ‘in control’ or ‘in charge’ of the whole” [2] ecosystem. Services in the SO Ecosystem are the means by which “the needs of a consumer are brought together with the capabilities of a provider”[2]. Services are the realization of business functionality accessible through defined service interfaces.

Service-Oriented Architecture is an architectural realization of the concept of Service Orientation in the SO Ecosystem. SOA promotes Principles of Service Orientation via architectural modelling of business functionality, solutions and design into an implementation. SOA is a business-oriented consumer-centric architecture that natively presents in corporate business and whose technical parts may be constructed and realized by IT departments.

SO Ecosystem provides an environment for service-oriented behaviour for services and people within and outside of an enterprise. Everything people do in an enterprise may be explained and described as servicing each other and servicing external customers.

### Principles of Service Orientation

We will review SO Principles in no particular order. All SO Principles are equally important in the context of this paper. The existing formulas of the SO Principles are provided after the definitions published by Thomas Erl [1].

#### Principles Related to Service Usability

##### Service Discoverability

In a SO Ecosystem, the only way for a service to survive is to attract consumers; the more, the better. This approach anticipates that the service use has or will have a ‘commercial’ basis, in other words an unutilized service has no chance to exist for long without consumers. For a consumer to find a service it should be discoverable through known public means. For business services, such discovery takes place long before service interfaces are contacted. Here is the current formula of the principle:

###### Service Discoverability Principle

“Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.”

This formula of the Service Discoverability Principle is adequate for the modern understanding of service ‘discoverability’. Also, the OASIS SOA RAF defines the content of the ”communicative meta data” specified as a Service Description [2]. The latter contains information for a potential consumer that the consumer can use to make a decision as to whether this particular service can satisfy certain consumer’s needs. Among others things, a Service Description includes all publicly available, but not necessarily accessible, interfaces of the service.

##### Service Contract

A service requires all consumers to agree to the Service Contract before the service may be engaged. The OASIS SOA RAF defines Service Contract [2] as an explicit or implicit agreement between the service or service provider and each consumer. If the Service Contract is implicit, the Service Description can play a role of the contract. Otherwise, the consumer and the service/provder have to negotiate and agree on certain subsets of the service’s features and applicable Execution Context (EC), which we will discuss in a moment. Also, a consumer may bring its own policies into the Service Contract.

The current formula of this principle states:

###### Standardized Service Contract

“Services within the same service inventory are in compliance with the same contract design standards”.

An “service inventory” is an element of a particular implementation of service governance and, thus, cannot be a part of a principle. Service inventory is reactive with regard to service existence, because services may have their individual ownerships and governance. If consumers operate or can operate in different realms of standards, it does not needed to adhering to the same contract design standards.

A Service Description describes what design standards are followed in the service and its interfaces. A Service Contract is derived (in the majority of its elements) from the Service Description, which is specific to each service. What really would be helpful is a standardization of the types of information presented in **Service Contracts. Every consumer should be able to choose what has to be in its Service Contract using standardized types.**

The new formula and name of the principle are:

Standardized Content Types of Service Contracts   
Service Contracts are in compliance with the standardized contract content types and represent mutual agreements between the service or service provider and service consumers on what content is to be included.

##### Service Execution Context

In a SO Ecosystem, each service operates in EC. This means that both the consumer - service communication and the service execution takes place in the EC.

An EC includes business and technical sub-contexts and is defined in the OASIS SOA RAF as a set of agreements between a consumer and a service/provider that define the conditions, under which service may be engaged and executed [2]. Usually, an EC comprises laws, policies, regulations, rules, customs, technology platforms, special security considerations and so forth. In essence, the behaviour of the service and its RWE depend on the EC.

The EC plays a significant role for business services, especially within those enterprises that are in regulated domains, for example banking, insurance, telecommunication, health care, and so on. The formula of this SO Principle states:

###### Service Execution Context Principle

Service interactions and execution take place in the Execution Context. The Execution Context can affect service behaviour and lead to changes in the reachability and Real World Effect of the services.

Service Contracts link the service RWE with the EC per interface. For example, the same service used or applied in different jurisdictions or regulatory regimes may deliver different results.

#### ****Principles of Service Construction****

##### **Separation of Concerns**

**A Separation of Concerns is a well-known Computer Science principle. In the SO Ecosystem, this general principle is interwoven with the concept of ownership. The OASIS SOA RAF recognizes and outlines the fact that each service may have its owner with specific rights and preferences. It is assumed that each service is totally independent from other services and utilizes the outcome of outer services on a contractual basis. For an enterprise, this means that it is not an IT Department who owns business services, but individual business group, team or division. We have witnessed many cases where business owners prohibited IT from modifying their services.**

**The concept of ownership leads to the ‘Knight Rules of Services’ [5], which allow a service to concentrate on its own functionality and relationships with its immediate consumers only. Particularly, an orchestrating service is similar to a regular business process, but, in contrast with business process owners, the service owners operate with supplemental service via contracts rather than aims to acquire them or obtain control over them.**

**The formula of this principle states:**

Service Separation of Concerns Principle  
Services own and carry out only their own functionality, being independent from providers of supplemental functionality and decoupled from information sources and original information structures as much as possible.

**This principle results in a high degree of appreciation of the interconnections between services. In order to guarantee its own business continuity, a service can contract redundant supplier services; the service consumer can do the same – to contract a few redundant supplier services for the same task.**

**Another consequence of this principle is an ability to separate a business service from its data sources. To obtain data from sources or storages, special data access services may be used. This enables business services to realize the principle of Composability in full and participate in as many compositions/cooperations as they want without ‘an anchor’ in data sources.**

##### Service Abstraction

A service works with consumers based on Service Contracts. The letter, as well as a Service Description (acting as an implicit contract), abstracts actual realization of the service – the service body – and provides only the information, which can help a potential consumer to make a decision as to whether this service can meet certain consumer’s needs. A Service Description is the only form of public information about the service.

Current formula of the principle states:

###### Service Abstraction

“Service contracts only contain essential information, and information about services is limited to what is published in service contracts”.

Apparently, Service Contracts are published only if they are implicit; otherwise, they are a private matter. Also, the current formula does not state that a Service Contract abstracts the service, which it does. In programming, an interface between interacting entities is known as a contract. We can say the same about services, but in this case the term ‘contract’ should be an interaction or communication contract. Without such explanation, the use of the term ‘contract’ with regard to services becomes ambiguous.

Thus, the current formula of this principle is not correct any more. We introduce two principles instead of the current one:

###### Service Abstraction Principle

Service interfaces abstract the service realization or body. Not all service functionality and capabilities may be visible through the service interfaces. Service interfaces cannot be reliable resources for understanding what the service does and its RWE.

###### Service Contract Abstraction Principle

A Service Contract abstracts the service. A Service Contract contains only information about the service that is agreed between the service or service provider and the service consumer(s). This information has to be sufficient for interacting with the service, utilizing agreed service functionality and reaching agreed Real World Effect in particular Execution Context.

##### Service Composability

Current formula of the principle of Composability states:

###### Service Composability

“Services are effective composition participants, regardless of the size and complexity of the composition.“

As we see, this formula does not support popular opinion that a service “May be composed of other services” [6]. From the formula, it is unclear what “composition” means. We believe it is a composition of services, at least. Such a composition may have its own unique value; a composition outcome may be more than just an aggregation of an individual service outcome. Thus, a composition of services appears to the consumer of the composition outcome as a service of its own though it may be not formally a particular service.

If a consumer creates a composition, holds and manages states of composed services, this composition is not a service overall. If a composition is created by an orchestrating service[[1]](#footnote-1), the latter hires supplement services via contracts. Thus, an updated formula of the principle is:

###### Service Composability Principle

Services are effective composition participants, as well as effective composition holders, regardless of the size and complexity of the composition.

This principle explains that a service may not be composed of other services. Instead, the principle allows the orchestrating service to be highly dynamic in contracting and using different supplemental services.

##### Autonomy

Current formula of the autonomy principle states:

###### Service Autonomy

“Services exercise a high level of control over their underlying runtime execution environment.”

The comments to this principle point to “an independence with which a service implementation can carry its logic and manage the resources it may need at runtime” [1]. If “a high level of control”may be interpreted as “an independence with which a service implementation can… manage the resources”, we may assume that such resources should always be individual, private. Otherwise, if a resource may be public and shared, there are no guarantees that two independent services can manage it concurrently and with integrity.

We believe that autonomy is about ability of a service to operate by itself, on itself. The only thing a service needs is data; if all needed data is provided by the consumer, the service must be able to accomplish its work by itself.

We distinguish between management of a resource and interacting with it. An autonomic service interacts with others (consumers/suppliers) on a contractual basis and never manages anything that is outside of its own boundaries [8]. All these interactions take place in EC, which can impact the service RWE.

The updated principle is as follows:

Service Relative Autonomy Principle  
Services exercise a relative independence from their Execution Contexts, control their internal resources and interact with external entities of the SO Ecosystem, if needed, on a contractual basis.

##### Reusability

Current formula of the principle of reusability states:

###### Service Reusability

“Services contain and express agnostic logic and can be positioned as reusable enterprise resources.”

We believe that a service may be used multiple times or reused. For example, if a bus is used on multiple routes, this is a multiple time use – the bus behaves in the same manner and operates in the same EC. However, if that bus pulls a platform with construction materials, this is a reuse – the bus is used for a slightly different functionality and in the changed EC.

The purpose of the service is to be available for multiple use and reuse. In a business landscape of a company, there are not many functions and features that are used in different combinations and contexts, that is to say really reused. The reusability of the service is important for the future adoption of business changes rather than for the implementation of existing functionality. Redesigning and re-developing existing business functionality exclusively for the purpose of reusability is usually inefficient.

Since the multiple use of services totally depends on the business (consumer) needs, it cannot be a principle. However, reusability is a special quality of services. This is the updated formula:

Service Reusability Principle  
Services contain and express logic that can be reused in different Execution Contexts; services can be positioned as reusable intra- and inter-enterprise resources.

The reusability of the services is about a reuse of the service as a whole. It is not tied to the enterprise borderline. If a service is properly designed, it may be offered to external consumers, or replaced, or outsourced.

##### Service State

Every service – business or technical – has a state. A service does not necessarily execute on a computer and may have as many manual operations in its body/implementation as needed. Business services, which form the core of a SO Ecosystem of an enterprise, manage their states as needed for the business functionality they provide – they may be statefull or stateless. However, the current title and formula of the principle promotes one particular software design pattern that supports a highly transactional scalable process:

###### Service Statelessness

“Services minimize resource consumption by deferring the management of state information when necessary.

This formula sounds trivial – indeed, if a service does not manage its state, it saves resources needed for such management. The question is only why a service should “minimize resource consumption”; why this is a principle of services and not of something else. We believe that the criteria in this case should be business efficiency or rationalism. What if the cost of minimizing resources is higher than the cost of consumed resources?

The aforementioned definition of the principle carries a trap for those (thousands and thousands of) people who think that the title of the principle is self-explanatory and its formula (which includes “when necessary”) may be forgotten. Those people believe that any service must be stateless. Well, on multiple occasions we saw that for the sake of this principle an enormous state data was sent over to the stateless service, resulting in degrading and even putting down the bandwidth. In our opinion, the title of this principle was chosen for the purpose of promoting services at that time. Statelessness is only one type of service applicability and it is driven only by technology. It is unclear why a SO principle has to be named in a way that promotes a particular implementation pattern.

Thus, this principle is in need of generalization. Here are the updated title and formula:

Service State Management Principle  
Services manage their own state as needed and may defer the management of their states in order to minimize consumption of service environment resources.

For example, if a company utilizes several Clouds that have to work together, the company can get into an inter-Cloud brokering role. The company now can manage the states of its Cloud services instead of paying a fortune for an integration between those independent Cloud providers.

##### Loose Coupling

Loose coupling is one of the most important design principles of services. The current formula of this principle states:

###### Service Loose Coupling

"Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.”

Based on the principles we have discussed already, the current wording of this principle appears erroneous. A Service Contract is an agreement between the service and its consumer, and it couples them as much as they want; there is no need to impose ‘low consumer coupling requirements’. Moreover, a consumer has to understand what Service Description and Service Contract says, i.e. they have to share the ontology and semantics. We do not have a measure of how much coupling such sharing creates. In general, business would never work with totally unknown (decoupled) service/provider – there will be no means of business trust between them.

Also, a Service Contract cannot be “decoupled from their surrounding environment” because in this case it will ignore the EC and risks to get in troubles. For example, a Financial Service working in the US, but compliant only with UK regulations, is likely to be illegal in the US and may be arrested and fined.

The confusion with this principle is that a simple service interface has been renamed into ‘service contract’, and the ontology of this term has evolved over time. In reality, we certainly need to preserve a loose coupling between the service body/implementation and its interfaces, regardless of whether they are programmatic or manual. At the same time, we cannot reach a full decoupling between the service interface and the service – it is the service that exposes the interface while an interface has no business meaning without the service that exposes it. An actual service interface behaves differently in different EC driven by the surrounding environment.

All these lead to the following formula for this principle:

###### Service Loose Coupling Principle

Services impose low coupling requirements between their interfaces and the service body, as well as between their interfaces and the service consumers in the Execution Contexts.

##### Service Boundaries

A combination of SO Principles that we have reviewed already shapes the conclusion that a service body/implementation should be self-contained and decoupled from consumers’ and supplement services’ implementation as much as possible. While it sounds trivial, this principle is too frequently violated in practice, especially if the same development team creates a service and its consumers, or reuses the same software code for different services (for example, utilizing a legacy application). Though this is more typical of IT departments, it can be found in the sphere of inter-company collaboration design as well (based on a shared codebase).

Any service has its interfaces and internal communication means that are dedicated to interactions with interfaces of external entities in the SO Ecosystem. A consumer has to provide the service with all information defined in the Service Contract. A service can obtain any additional information via its internal communication means by invoking public interfaces of supplement services. No other channels of interaction between the service and the external world may be permitted if we preserve such principles as Separation of Concerns, Autonomy, Composability and Reusability. That is, service interfaces and internal communication means constitute service boundaries. A service ‘owns’ everything within its boundaries.

The concept of service boundaries impacts service implementation a great deal. A software code in the service body may not interact, communicate or inherit from a code under different ownership. Otherwise, we will have service ‘coupling by implementation’ [8], which violates the aforementioned SO Principles. This principle states:

###### Service Precise Boundaries Principle

Service implementation means may not cross service boundaries, neither via invocation nor via dependency on the resources that are out of the service’s ownership or explicit control.

We understand ‘explicit control’ as the right and ability of the service/provider to modify and move the resources independently from anybody else.

### Conclusion

This paper has reviewed the existing principles of Service Orientation and found that some of them became obsolete. An understanding of the Service Orientation concept, SO Ecosystem and services existing in it significantly changed over the last 15 years. This relates to such new aspects as service EC and a re-defined notion of the Service Contract. Also, a principle of service boundaries can finally segregate services development practice from the one of applications, which a lot of developers are confused with.

We have redefined the formulas for several principles and renamed a few of them. This work has been based on the first OASIS SOA RAF standard, which has crossed the ‘line’ between business and technology. We tried to avoid and eliminate technology specific wording in the principles’ formulas. The paper has also included our reasons for particular revisions and a few explanations and examples.

### References

1. Thomas Erl SOA Principles *of Service Design*. Prentice Hall; 1st edition. Jul. 2007. ISBN-10: 0132344823 ISBN-13: 978-0132344821
2. Reference Architecture Foundation for Service Oriented Architecture, Version 1.0. Committee Specification 01. 04 December 2012
3. Anne Thomas Manes, “SOA is Dead; Long Live Services”. BLOGs, Burton Group. Jan. 2009
4. Reference Architecture Foundation for Service Oriented Architecture, Version 1.0. Public Review Draft 01. April 2008
5. Michael Poulin, Knight Rules of Ownership in Service-Oriented Ecosystem Business Ecology Initiative & Service-Oriented Solution, ebizQ, Jun 2012. <http://www.ebizq.net/blogs/service_oriented/2012/06/knight_rules_of_ownership_in_service-oriented_ecosystem.php>
6. *Service Oriented Architecture : What Is SOA?*. The Open Group. <http://www.opengroup.org/soa/source-book/soa/soa.htm>
7. Michael Poulin, Architects Know What Managers Don’t. Butechcon-Troubador Publishing, Apr. 2013. ISBN-10: 0957519907 ISBN-13: 978-0957519909 <http://www.mpoulin.com/architects-know-what-manager-dont/>
8. Michael Poulin, A Domain Service-Oriented Modelling or How SOA Meets DDD, Parts 1-5, BLOG, ebizQ. <http://www.ebizq.net/blogs/service_oriented/2009/01/a_domain_service-oriented_modelling_or_how_soa_meets_with_ddd.php>

# Bio

**Michael Poulin** is Head of Enterprise Architecture at Clingstone Ltd., a consulting firm based in the UK and working with business Change Management and Enterprise/Solution Architecture. He has built up a wealth of experience architecture in both the UK and United States. His work focuses on bridging the gap between business architecture and modern technology.    
Michael provides professional coaching and knowledge exchange on his Web Site www.mpoulin.com.  Michael is actively engaged in the Enterprise Architecture realm having actively contributed in OASIS SOA standards. Michael has also authored two books – ”Ladder to SOE” and “Architects Know What Managers Don’t” – and various other publications. He can be contacted via [michael.poulin@clingstone.co.uk](mailto:michael.poulin@clingstone.co.uk).

1. A choreography of services is a form of collaboration, which assumes changes in the service bodies for the sake of the collective work, i.e. defeat the purpose of services in SO Ecosystem [7]. [↑](#footnote-ref-1)