CHAPTER 7

Technicalities of the Service Orientation Concept

…

7.4 SOA Service and Service Attributes

'In SOA, services are the mechanism by which needs and capabilities are brought together', according to SOA RM. This sounds very simple, but there is no definition of what 'needs' and 'capabilities' mean; why an application in IT, which is considered to be an SOA service, becomes a mechanism; and, finally, where all the technical definitions of interfaces, clients, providers, QoS controls, etc., are. In fact, they still exist but now they have been repurposed toward business agility.

The SOA RM states that an SOA service operates in an execution context (EC), which is defined as *'the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities.'* The standard expands an interpretation of 'those with needs' and 'those with capabilities': 'those' are not only IT applications and operations, they are the business elements as well and, moreover, the business elements come first: *'The execution context is not limited to one side of the interaction; rather it concerns the totality of the interaction - including the service provider, the service consumer and the common infrastructure needed to mediate the interaction.' An* EC is illustrated in Figure 7.4.a.

If IT takes a look at its own products from the business perspective, it can find that very few applications correspond directly to the core business tasks; many applications support a lot of 'just-this-moment' operational requirements that were actual years ago. If you add runtime and procedural application integration to this picture, it becomes apparent why IT has difficulties with implementing and supporting frequent business changes.

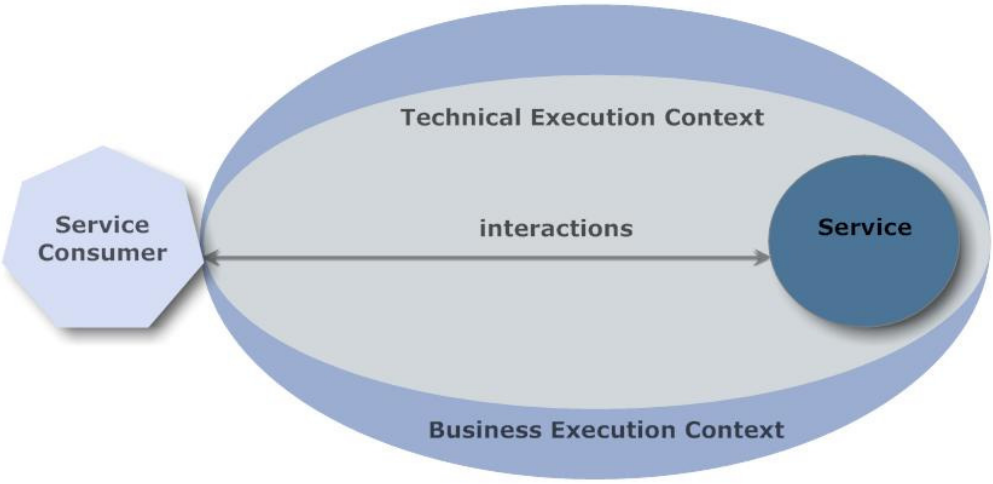

Figure 7.4.a. Service Execution Context

It's important to point out that not every SOA service is that complex and there are a lot of simple and very useful services that IT can offer. However, the power of SOA is in the services that implement business services and, accordingly, get involved in the complexity and risks of real business life.

So, what is the SOA service? It is an entity with business capabilities (offered business functionality) that may be announced, i.e. made visible to the consumers, and employed; it is an entity that operates based on SO principles in the execution contexts providing for the Real World Effect (RWE). The entity has to comprise at least one element - business or technical - or both. When a business element presents, we call it a business service; otherwise, it is one of several types of technical infrastructural services. The business element may be expressed via a description of the business service in the business enterprise model, or via a description of the business process, business function, business feature, business data, or via a description of a combination of all the above. It is assumed that some of the business elements listed may contain business logic, business rules, and business policies.

The SOA business services may be implemented as manual business operations as well as automated services that will be discussed later. Here we elaborate more on the service Execution Context and service capabilities. According to the SOA RM, the *'Execution context is central to many aspects of a service interaction.'* The SOA Reference Architecture to-be' standard (which is OASIS Public Review Draft 1 at the time of writing this book; hereinafter SOA RA PRO 1) [6] explains that 'service interaction' includes all aspects of interaction between the service consumer and service provider. That is, 'service interaction' consists of at least two parts - service communication and service execution. Both are performed under related communication and execution policies that, actually, constitute the EC.

The EC may include business and technical contexts as well as behavioural and information models. Business context contains business policies, rule, and regulations that affect functional and non-functional aspects of the service. Technical context contains technical policies and rules including protocols, security access controls, non-functional characteristics, communication origins, service interfaces, and message formats as well as service execution policies that influence service results, i.e. RWE. The major structural difference between an SOA service and a traditional application is that business and technical policies situate outside of the service and can change independently, ruling the service in the changed environment.

Due to the role of EC, the same service can deliver different RWE in different execution contexts. This is the exclusive quality of SOA business service, which contrasts with the popular IT assumption that a service has to produce exactly the same results in any circumstances. Such a belief in environment-independent behaviour might be justified only if a service is seen as simply an interface; otherwise, it would be silly. In SOA, consumers see only service interfaces while service implementation is invisible; however, despite an immutable interface, the dependence of the service results on the execution context is very visible. This dependence is the natural business behaviour associated with business adaptability to the local external environment.

Here are two examples of the role of EC in the world of financial IT. First, a company's Logging Service may store data 'in the clear' when serving load-balancing calculations but it has to encrypt data for financial risk calculations due to financial regulations that constitute the Execution Context. Thus, depending on the EC, the performances of the Logging Service will be different (data encryption is a time-consuming operation), the service throughput will be different, and the service's resource - log store - will have different load, which may affect other services using the same store.

Another example, illustrated in Figure 7.4.b, is more complex. A global fund management organisation operates in the USA and the UK. In both places, it calculates Mutual Fund prices daily. The UK IT developed a Fund Price Service, which was intended for use in both countries. The User Acceptance Test suddenly demonstrated that intra-day calculated prices for the same Fund in the UK and in the US differed while the end-of-day results matched each other. It was found that the price difference came from the difference in the EC - local regulations for Fund prices: in the UK, it was based on the bid/offer price that changed for the day, while in the US it was based on the 'closing' price calculated by the end of the day.

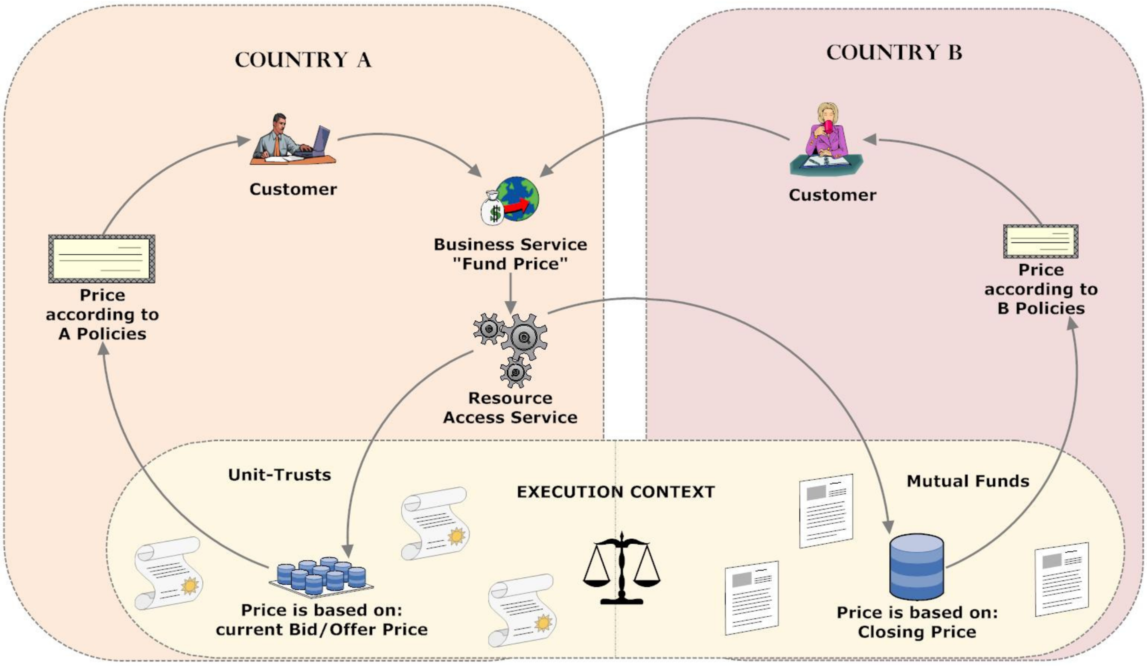
COUNTRY A
COUNTRY B
Customer
Customer
Business Service "Fund Price"
Price according to A Policies
Price according to B Policies
Resource Access Service
Unit-Trusts
EXECUTION CONTEXT
Mutual Funds
Price is based on: current Bid/Offer Price
Price is based on: Closing Price

Figure 7.4.b, Influence of the Execution Context on the SOA service's Real World Effect

To conclude, the SOA Reference Model describes a RWE in terms of changes of the state of the world shared by the service consumer and service provider, in particular, the public aspects of the sate. Consequently, the SOA RA PRD 1 defines service capability as a functionality provided by the *'resource that may be used by a service provider to achieve a real world effect on behalf of a service consumer".* According to these definitions, we can conclude that service capabilities and resources are not necessarily technical or automated entities; they may be human resources performing manual operations, e.g., loan approval or manual corrections of absent data in the financial transactions. This is an important result that extends SOA beyond the Technology world into the Business world.

Since it is now clearer what service capabilities are, we are saying that a SOA service implements much more than the interface 'API'. The SOA service is responsible for the functional and non-functional aspects of business task implementation. Not all of those aspects may be visible through the service interfaces but the consumer has to be aware of them. The 'invisible' capabilities are as important to the service consumer as the visible ones, because the consumer becomes depend on the 'honest' behaviour of the service outside of the consumer's control. For example, your company engages a third-party service to publish your results on the Internet. Would you use that service if you discovered that it did not filter porn ads and they were published together with your data?

To help service consumers in dealing with the services, SOA extends the concept of Service Discovery and Invocation known from the CORBA and Web Service technologies. SOA offers two major attributes of the service that are identified in the SOA RM and refined in the SOA RA PRD 1: Service Description and Service Contract. Both of these attributes distinguish SOA service from other services and it is very important to understand them properly.

...

7.4.4 Service Attributes Working Together

Figure 7.4.4 shows how the SOA service attributes mentioned can work together in the execution environment. The actors in the diagram are: service consumer, service registry/repository, environment intermediary, monitoring systems, policy controls and enforcement systems, service, service's RWE, and service provider.

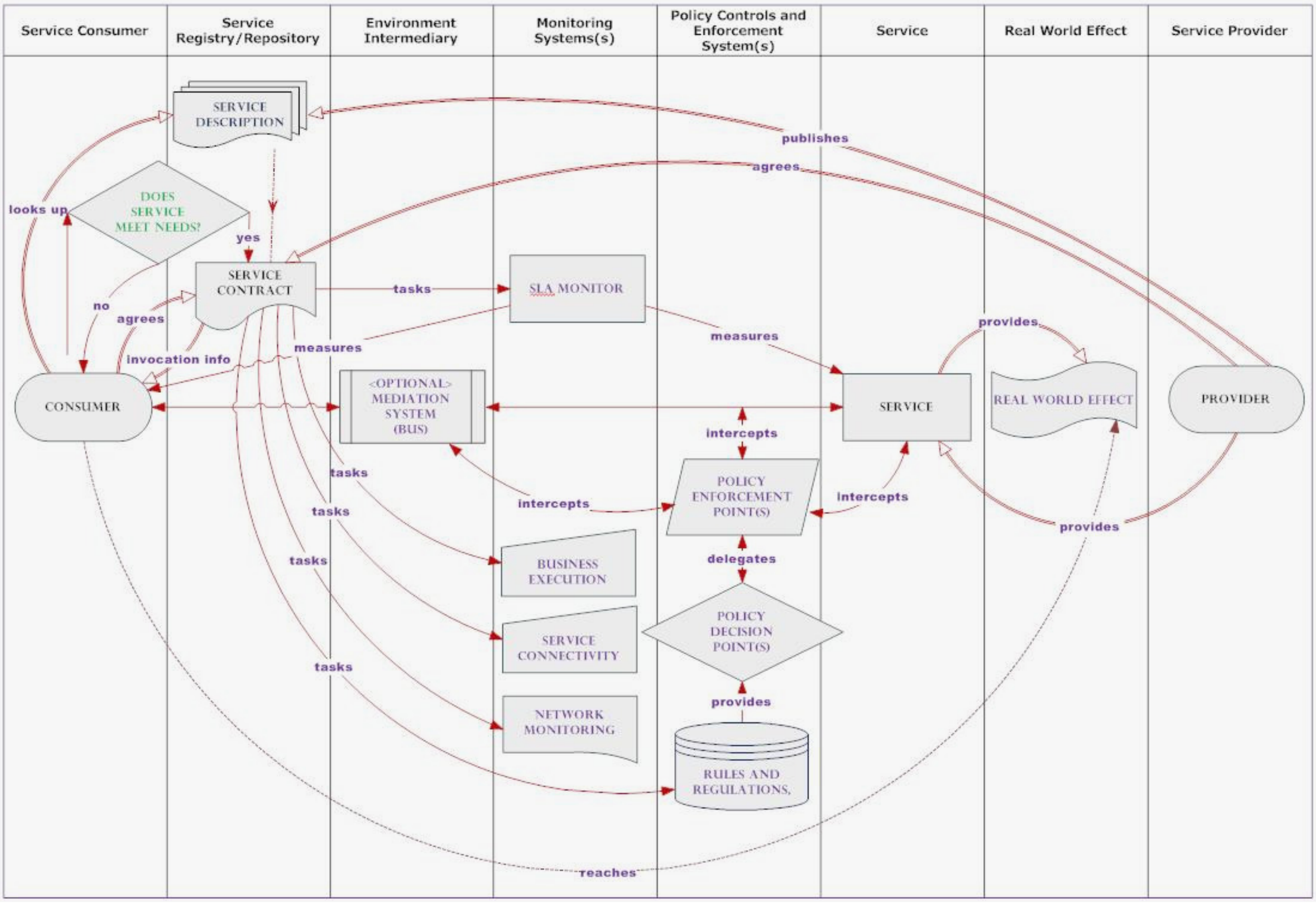

Figure 7.4.4. Service attributes usage pattern

It is important to note that a service consumer does not always interact with a service provider directly but sometimes via the service. The service provider publishes the Service Description and provides the service execution. Publishing may include a registration of some sort and an announcement about service availability directed to potential consumers. It is a responsibility of the service consumers to find or lookup the service and decide whether the service functionality and offered RWE meet their needs. Whatever the service complexity is, the Service Description has to provide enough information to attract consumers. When the service is chosen, the consumer sets the Service Contract with the service provider. Ideally, the Service Contract has to be in a form that is not only readable by a human consumer but also suitable for automatic processing. The Service Contract becomes the source of information about programmatic service interaction - invocation pattern, logical and physical connectivity interfaces, and exchange message formats. Additionally, the programmatic processing of the Service Contract can supply the Monitoring systems with the service's SLA characteristics. The service consumers should be able to assess compliance of the service with the SLA based on the measurements of monitored SLA characteristics.

The Policy Control and Enforcement system (PCES) can use the Service Contract to identify which policies ought to be applied to the service interactions with the relevant consumer. It is assumed that all run-time policies that a Service Contract might contain are enclosed in the Run-time Policy Registry (a part of PCES) as a preliminary. Since popular practice considers a service interaction exclusively as communication with the service, it is important to highlight that PCES applies to both service communication channels and service body execution via Policy Enforcement Points and related Policy Decision Points.

Figure 7.4.4 also shows that a service consumer can interact with the service directly or via an optional environmental service Intermediary like Enterprise Service Bus (ESB). In the latter case, the consumer treats the Intermediary as a part of the service. Since the consumer enters the Service Contract with the service provider, not with the Intermediary, the service provider has to take care of the service delivery to the consumer, i.e. via the Intermediary or some other way. Otherwise, the consumer has to set the Service Contract with the Intermediary and it should be the Intermediary who promises access to the business functionality and RWE (how it would do this does not matter). Finally, the consumer accesses the service execution results - RWE - via a service interface or via a notification about the RWE. All exceptions that might be exposed to the consumer ought to be passed only through the service interfaces.

7.4.5 Service Granularity

Service granularity is the driver for the service interface granularity. It is one of the key design considerations for the services. Depending on the nature of the service - business or infrastructural - the interface granularity may be more or less granular; there is no standardised set of rules that governs the granularity.

Service granularity affects many aspects of the service development, management, and maintenance. For example, the more fine-grain the service interface is, the higher the probability that market changes will cause changes in the interface. This may result in several service versions being maintained simultaneously and will require the need for strong version control. Also, the more a service has been reused as is, the more difficult it would be for the consumers to absorb the service changes, which requires even more versions to be supported. This is why a coarse service interface is generally favoured.

Here are a few popular considerations about the service granularity:

- A coarse-grained interface reduces chattiness of the service interaction but can result in increasing message sizes. These two aspects have to be in a balance with network and service performance requirements.

- Common opinion states that the reuse of a service in the 'as is' form is generally encouraged by the finer-grained service interfaces because they are more 'adaptable' to alternate scenarios of the use. In my opinion, this is erroneous.

- Granularity of service interface operations is constrained by the limitations of the message size and memory. Actually, this is a typical example of bearded IT approach - technical capabilities dictate business solutions while in SOE it is supposed to be reversed.

The observations about granularity and reuse 'as is' are an inheritance from the Web Services vision that arose from the so-called RPC-style of the Web Services. This style mimics classical RPC and becomes associated directly in the developer's mind with a remote API. The simpler and more fine-grained API is, the easier to reuse it as is. It is a common programming rule for 00 programming languages. However, this rule does not really work in distributed computing and developers who use this programming style for the services usually try to avoid answering a simple question - how much effort/time it would take them to implement a change in the API after it was widely reused in the products. SOA is about flexibility and readiness to change with minimal time and effort, and while reusability might be a way to reach such flexibility it appears that a reuse 'as is' does not help much. Fortunately services may be reused in different manner - as containers. This type of reuse is 'by extension'; it permits defining business-specific (typed) but coarse-grained interface operations that are capable of communicating extended/changed messages without changing anything else in the service interface. This is possible to do using, e.g., document-style of Web Services and operating on the message (document) namespaces [7].

An alternative approach to service granularity management may be provided by using multiple interfaces for the same SOA service. The interfaces may not only address different communication means like HTTP, HOP, or WAP but also different levels of granularity for the same communication channel. Plus, different consumers may have Service Contracts that specify interfaces of different granularity. It is important to note that service orientation principles do not include service interface supposition, i.e. each interface is an independent entity. Thus, a more coarse-grain interface does not necessary include a finer-grained interface. Be warned!

7.5 Service-based vs. Service-enabled Design

A view on the purpose of SOA was established when Web Services were used for integration between heterogeneous systems in IT. Indeed, what could be more convenient for integration than standard-based and technical platform agnostic Web Services? I refer here to SOAP Web Services rather than RESTful Services because the latter require specific environment -Internet - and protocol - HTTP - to be used for integration means (which is not a bad idea in itself, but might be not suitable for some cases).

People take monolithic legacy applications or huge packaged applications for ERP, put Web Services on top of them and claim it is a SOA design. Well, if so, which part of it is service-oriented? Many vendors and consulting firms promote this approach because it is quick, effective, and demonstrable. But is it efficient? This depends on what you want to achieve.

If a company suffers from a proprietary 'spaghetti' point-to-point inter-application links, then with Web Services integration you will get the standard-based 'spaghetti'. If a company is lucky and its business dependency logic is not complex, then an intermediary or bus/broker/mediator can help orchestrate Web Services. So, Web Services work nicely but they are not necessarily what is required. It depends whether you are looking for an elegant integration solution or for a business value resulting from collaborative work by interrelated systems.

To illustrate this point, let us look at a bank, which charges a fee for using its ATM (cash) machine. The bank wants to improve its ATM service to deliver more fees. To achieve this goal, the bank builds a special room with beautiful revolving door and places the ATM inside. Does the bank reach its goal? Yes, to some degree, because fee flow increases a little -more people are attracted by convenience of queuing inside rather than in the street. However, the ATM still processes one customer at a time and the process pace is the same as before. It is the room that produces more fees, not the ATM. It is a service-enabling solution because it just reuses the asset without making it a service. The service-based solution would be, for example, either increase the processing speed of the ATM (improve the service) or increase the number of ATM machines in one place (scale the service).

Service-enabling design provides the same effect as the Web-enabling one that is so familiar from the past. Many Web-enabled applications failed dramatically (together with their creators) because they could not meet Web usability requirements - the applications simply were not designed to sustain concurrent request flows at the Web scale. Web Service enabled applications face the same problem. Moreover, an integration per se does not produce any services. On the contrary, it deepens and preserve the silo of the applications while service orientation breaks the silo and extracts useful functionality; SOA allows the remaining parts of the application to rest (until they retire).

Service-based design creates services first of all. Services can wrap legacy applications but some of them might not be reused in a different execution context. An idea of a 'Composite Application' promoted, for example, by Sun Microsystems and IBM is a bright idea. However, it suggests that all legacy systems and old investments can work well underneath such an application, which is simply not true. Many of those systems are not multi-threaded, they cannot support a realistic number of concurrent on-demand requests, and their computational and storage resources may not be adequate to serve against modern quality of service (QoS) requirements. In other words, an interface or new communication channel does not automatically create a service. Real service has to be constructed or re-constructed on the service-oriented principles.