FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Rajendra Kharbuja

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Entwerfen einer Business-Plattform mit microservices.

Author:	Rajendra Kharbuja
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Manoj Mahabaleshwar
Submission Date:	

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Rajendra Kharbuja

Acknowledgments

Abstract

Contents

1 Context

1.1 Monolith Architecture Style

A Mononlith Architecture Style is the one in which an application is deployed as a single artifact. The architecture inside the application can be modular and clean. In order to clarify, the figure 1.1 shows architecture of an Online-Store application. The application has clear separation of components such as Catalog, Order and Service as well as respective models such as Product, Order etc. Despite of that, all the units of the application are deployed in tomcat as a single war file.[Ric14a][Ric14b]

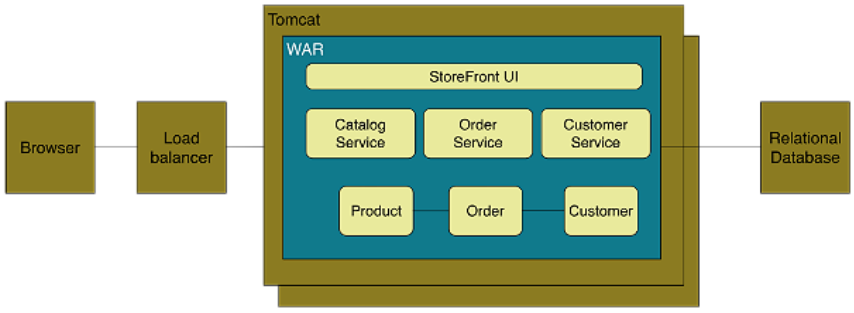

Figure 1.1: Monolith Example from [Ric14a]

1.1.1 Types of Monolith Architecture Style

According to [Ann14], a monolith can be of several types depending upon the viewpoint, as shown below:

1. Module Monolith: If all the code to realize an application share the same codebase and need to be compiled together to create a single artifact for the whole application then the architecture is Module Monolith Architecture. An example is show in figure 1.2. The application on the left has all the code in the same codebase in the form of packages and classes without clear definition of modules and get compiled to a single artifact. However, the application on the right is

developed by a number of modular codebase, each has separate codebase and can be compiled to different artifact. The modules uses the produced artifacts which is different than the earlier case where the code referenced each other directly.

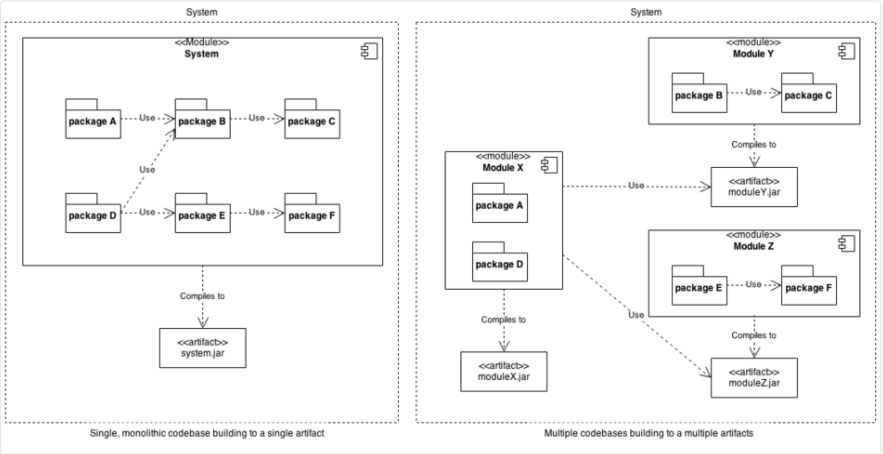

Figure 1.2: Module Monolith Example from [Ann14]

2. Allocation Monolith: An Allocation Monolith is created when all code is deployed to all the servers as a single version. This means that all the components running on the servers have the same versions at any time. The figure 1.3 gives an example of allocation monolith. The system on the left have same version of artifact for all the components on all the servers. It does not make any differenct whether or not the system has single codebase and artifact. However, the system on the right as shown in the figure is realized with multiple version of the artifacts in different servers at any time.

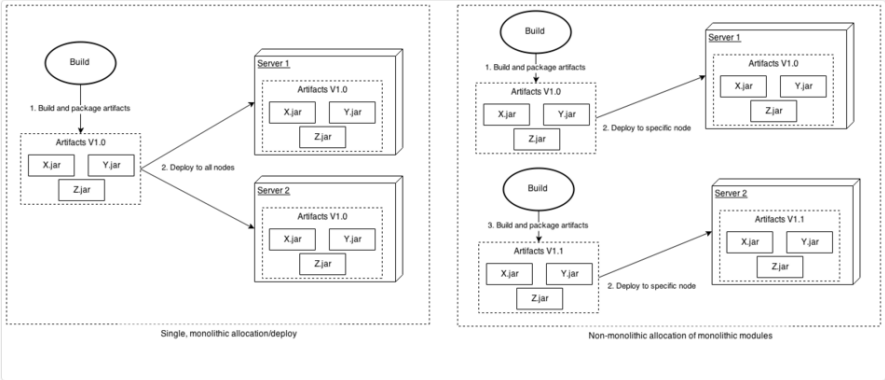

Figure 1.3: Allocation Monolith Example from [Ann14]

3. Runtime Monolith: In Runtime Monolith, the whole application is run under a single process. The left system in the figure 1.4 shows an example of runtime monolith where a single server process is responsible for whole application. Whereas the system on the left has allocated multiple server process to run distinct set of component artifacts of the application.

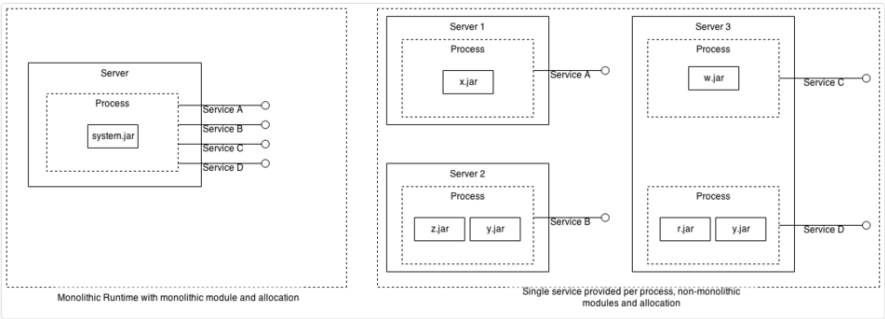

Figure 1.4: Runtime Monolith Example from [Ann14]

1.1.2 Advantages of Monolith Architecture Style

The Monolith architecture is appropriate for small application and has following benifits:[Ric14b][FL14][Gup15][Abr14]

- It is easy to develop a monolith application since various development tools including IDEs are created around the single application concept. Nevertheless,

it is also easy to test the application by creating appropriate environment on the developer's machine.

- The deployment can be simply achieved by moving the single artifact for the application to an appropriate directory in the server.
- The scaling can be clearly and easily done by replicating the application horizontally across multiple servers behind a load balancer as shown in figure 1.1
- The different teams are working on the same codebase so sharing the functionality can be easier.

1.1.3 Disadvantages of Monolith Architecture Style

As the requirement grows with time, alongside as application becomes huge and the size of team increases, the monolith architecture faces many problems. Most of the advantages of monolith architecture for small application will not be valid anymore. The challenges of monolith architecture for such agile and huge context are as given below:[NS14][New15][Abr14][Ric14a][Ric14b][Gup15]

- Limited Agility: As the whole application has single codebase, even changing a small feature to release it in production takes time. Firstly, the small change can also trigger changes to other dependent code because in huge monolith application it is very difficult to manage modularity especially when all the team members are working on the same codebase. Secondly, to deploy a small change in production, the whole application has to be deployed. Thus continuous delivery gets slower in case of monolith application. This will be more problematic when multiple changes have to be released on a daily basis. The slow pace and frequency of release will highly affect agility.

- Decrease in Productivity: It is difficult to understand the application especially for a new developer because of the size. Although it also depends upon the structure of the codebase, it will still be difficult to grasp the significance of the code when there is no hard modular boundary.Additionally, the developer can be intimidated due to need to see the whole application at once from outwards to inwards direction. Secondly, the development environment can be slow to load the whole application and at the same time the deployment will also be slow. So, in overal it will slow down the speed of understandability, execution and testing.

- Difficult Team Structure: The division of team as well as assigning tasks to the team can be tricky. Most common ways to partition teams in monolith are by technology and by geography. However, each one cannot be used in all the situations. In any case, the communication among the teams can be difficult and slow. Additionally, it is not easy to assign vertical ownership to a team from particular feature from development to relase. If something goes wrong in the deployment, there is always a confusion who should find the problem, either operations team or the last person to commit. The approprate team structure and ownership are very important for agility.

- Longterm Commitment to Technology stack: The technology to use is chosen before the development phase by analysing the requirements and the maturity of current technology at that time. All the teams in the architecture need to follow the same techonology stack. However, if the requirement changes then there can be situation when the features can be best soloved by different sets of technology. Additionally, not all the features in the application are same so cannot be treated accordingly in terms of technology as well. Nevertheless, the technology advances rapidly. So, the solution thought at the time of planning can be outdated and there can be a better solution available. In monolith application, it is very difficult to migrate to new technology stack and it can be rather painfull process.

- Limited Scalability: The scalability of monolith application can be done in either of two ways. The first way is to replicate the application along many servers and dividing the incoming request using a load balancer in front of the servers. Another approach is using the identical copies of the application in multiple servers as in previous case but partitioning the database access instead of user request. Both of these scaling approaches improves the capacity and availability of the application. However, the individual requirement regarding scaling for each component can be different but cannot be fulfilled with this approach. Also, the complexity of the monolith application remains the because we are replicating the whole application. Additionally, if there is a problem in a component the same problem can affect all the servers running the copies of the application and does not improve resilency.[Mac14][NS14]

1.2 Decomposition of Application

The section 1.1.3 specified various disadvantages related to monolith architecture style. The book [FA15] provides a way to solve most of the discussed problems such as agility, scalability, productivity etc. It provides three dimensions of scalability as shown in figure 1.5 which can be applied alone or simultaneously depending upon the situation and desired goals.

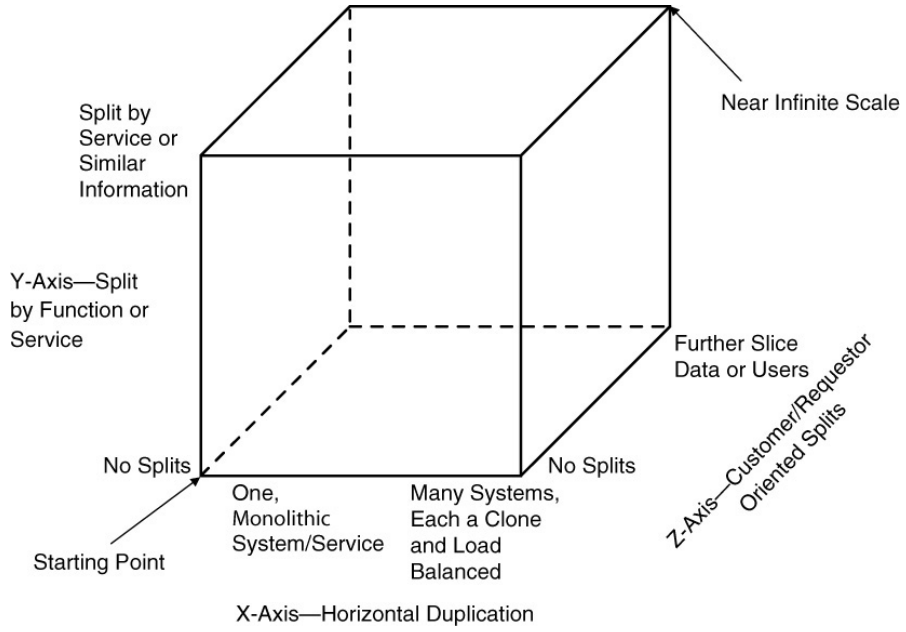

Figure 1.5: Scale Cube from [FA15]

The scaling along each dimensions are described below. [FA15][Mac14][Ric14a]

1. X-axis Scaling: It is done by cloning the application and data along multiple servers. A pool of requests are applied into a load balancer and the requests are deligated to any of the servers. Each of the server has the full capability of the application and full access to all the data required so in this respect it does not make any difference which server fulfills the request. Rather, it is about how many requests are fulfilled at any time. It is easy to scale along X-axis as the number of requests increases. The solution is as simple as to add additional clones. However, with this type of scaling, it is not scale with the increase in data. Moreover, it also does not scale when there are large variation in the frequency of any type of requests or there dominant requests types because all the requests

are handled in an unbiased way and allocated to servers in the same way.

2. Z-axis Scaling: The scaling is done by spliting the request based on certain critera or information regarding the requestor or customer affected by the request. It is different than X-axis scaling in the way that the servers are responsible for different kinds of requests. Normally, the servers have same copy of the application but some can have additional functionalies depending upon the requests expected. The Z-axis scaling helps in fault isolation and transaction scalability. Using this scaling, certain group of customers can be given added functionality or a new functionality can be tested to a small group and thus minimizing the risk.

3. Y-axis Scaling: The scaling along this dimension means the spliting of the application responsibility. The separation can be done either by date, by the actions performed on the data or by combination of both. The respective ways can be referred to as resoure-oriented or service-oriented splits. While the x-axis or z-axis split were rather duplication of work along servers, the y-axis is more about specialization of work along servers. The major advantage of this scaling is that each request is scaled and handled differently according to its necessity. As the logic along with the data to be worked on are separated, developers can focus and work on small section at a time. This will increase productivity as well as agility. Additionally, a fault on a component is isolated and can be handled gracefully without affecting rest of the application. However, scaling along Y-axis can be costly compared to scaling along other dimensions.

2 Granularity

2.1 Introduction

The granularity of a service is often ambiguous and has different interpretation. In simple term, it refers to the size of the service. However, the size itself can be vague. It can neither be defined as a single quantitative value nor it can be defined in terms of single dependent criterion. It is difficult to define granularity in terms of number because the concepts defining granularity are vague and subjective in nature. If we choose an activity supported by the service to determine its granularity then we cannot have one fixed value instead a hierarchical list of answers; where an activity can either refer to a simple state change, any action performed by an actor or a complete business process. [Linnp], [Hae+np]
Although, the interest upon the granularity of a component or service for the business users only depends upon their business value, there is no doubt that the granularity affects the architecture of a system. The honest granularity of a service should reflect upon both business perspective and should also consider the impact upon the overall architecture.
If we consider other units of software application, we come from object oriented to component based and then to service oriented development. Such a transistion has been considered with the increase in size of the individual unit. The increase in size is contributed by the interpretation or the choice of the abstraction used. For example, in case of object oriented paradigm, the abstraction is chosen to represent close impression of real world objects, each unit representing fine grained abstraction with some attributes and functionalities.
Such abstraction is a good approach towards development simplicity and understanding, however it is not sufficient when high order business goals have to be implemented. It indicates the necessity of coarser-grained units than units of object oriented paradigm. Moreover, component based development introduced the concept of business components which target the business problems and are coarser grained. The services provide access to application where each application is composed of various component services. [Linnp]

2.2 Related Work

A number of researches have been done in the area of service and component granularity. The focus of the researches includes various areas such as definition, effect, its dependent criteria and various measurement metrics.
According to [Pad04], the granularity of a component is inversely related to various non functional qualities such as customization and maintainaibility.
According to Hazen and Sims [HS00], component granularity is rather recursive as a component can be composed of various fine-grained components. They further classify recursion as of discrete or continuous nature but then prefer on discrete recursion. Thus, component granularity can be divide into three discrete layers: system, business and distributed where the composition happens from right to left order and granularity decreases in the opposite order. It is also useful to relate granularity with reusability. Coarse-grained components have high reuse efficiency because they are well focused to a specific business functionality. Whereas fine-grained components have high reusability since they focus on small functionality and thus can be used by other coarse-grained components to accomplish higher level business capabilities. [Mil+01], [WXZ05]
Additionally, Sims [Sim05] gave some insights to measure granularity. The granularity can be measured either a) by the number of components called from an operation on a service interface, b) by the number of components calling the service interface or c) by the number of database tables updated.
Service Oriented Architecture Framework (SOAF [EAK06] also gives some way to measure the granularity in terms of quantitative value. The value can be derived from a) the number of components that are invoked by the service interface and b) the number of changes of states in resources like the database tables updates influenced by the service. Again, the granularity can affect reusability and the network communication to accomplish any task. Coarse grained components have more volume of data exchanged where as fine grained components have high frequency of round trips to complete the full business functionality. It is recommended to balance them which can depend upon various factors such as abstraction level, change expectation rate, service complexity, desired coupling and cohesion. Furthermore, the granularity can be at different levels within an application. High-level business process can be realized as a coarse-grained services and each coarse-grained component can be composed of various fine-grained services.
The idea of multi-layered granularity is further supported by Security Trading case study [EAK06] and International Financial and Brokerage Services IFBS case study [Nav08]. The design of pilot application for Security Trading consists of various services across various levels such as process, application, shared data and infrastructures and containing services with granularity decreasing from left to the right in the order they

are listed. The application for IFBS is refactored from large number of fine-grained service to multi-layered granular services such as process services, business services, composite services or orchestration services, utility services etc.
A research performed by Steghuis [Ste06] talks about various aspects of granularity such as functionality, flexibility, reusability, complexity, context-independence, Performance, Genericity and sourcing. Some differenciating aspects in comparision to previous researches are context-independence, genericity and sourcing. Context-independence means independence and self sufficiency in terms of data as well as logic. It promotes loose coupling and insists lack of knowledge of state of surrounding and inessential maintainance of own state. Genericity is the quality of making generic services which can be used in different situations such as messaging service. Finally, sourcing is the process of identifying cohesive group of tasks which can be outsourced by considering the coupling associated.

2.3 Basic Principles to Service Granularity

In addition to quantitative perspective on granularity, it can be helpful to approach the granularity qualitatively. The points below are some basic principls derived from various scientific and research papers, which attempt to define the qualitative properties of granularity. Hopefully, these principles will be helpful to come with the service of right granularity.

1. The correct granularity of a component or a service is dependent upon the time. The various supporting technologies that evolve during time can also be an important factor to define the level of vertical decomposition. For eg: with the improvement in virtualization, containerization as well as platform as a service technologies, it is fast and easy for deployment automation which supports multiple fine-grained services creation.[HS00]

2. A good candidate for a service should be independent upon the implementation but should depend upon the understandability of domain experts.

   [Hae+np; HS00]

3. A service should be an autonomous reusable component and should support various cohesion such as functional (group similar functions), temporal(change in the service should not affect other services), run-time(allocate similar runtime environment for similar jobs; eg. provide same address space for jobs of similar computing intensity) and actor (a component should provide service to similar users). [Hae+np], [HS00]

4. A service should not support huge number of operations. If it happens, it will affect high number of customers on any change and there will be no unified view on the functionality. Furthermore, if the interface of the service is small, it will be easy to maintain and understand.[Hae+np], [RS07]

5. A service should provide transaction integrity and compensation. The activities supported by a service should be within the scope of one transaction. Additionally, the compensation should be provided when the transaction fails. If each operation provided by the service map to one transaction, then it will improve availability and fault-recovery. [Hae+np], [Foo05] [BKM07]

6. The notion of right granularity is more important than that of fine or coarse. It depends upon the usage condition and moreover is about balancing various qualities such as reusability, network usage, completeness of context etc. [Hae+np], [WV04]

7. The level of abstraction of the services should reflect the real world business activities. Doing so will help to map business requirements and technical capabilities. [RS07]

8. If there are ordering dependencies between the operations, it will be easy to implement and test if the dependent operations are all combined into a single service. [BKM07]

9. There can be two better approaches for breaking down an abstraction. One way is to separate redundant data or data with different semanting meaning. The other approach is to divide services with limited control and scope. For example: A Customer Enrollment service which deals with registration of new customers and assignment of unique customer ID can be divided into two independent fine-grained services: Customer Registration and ID Generation, each service will have limited scope and separate context of Customer.[RS07]

10. If there are functionalities provided by a service which are more likely to change than other functionalities. It is better to separate the functionality into a fine-grained service so that any further change on the functionality will affect only limited number of consumers. [BKM07]

2.4 Dimensions of Granularity

As already mentioned in 2.1, it is not easy to define granularity of a service quantitatively. However, it can be made easier to visualize granularity if we can project it

along various dimensions, where each dimension is a qualifying attribute responsible for illustrating size of a service. Eventhough the dimensions discussed in this section will not give the precise quantity to identify granularity, it will definitely give the hint to locate the service in granularity space. It will be possible to compare the granularity of two distinct services. Moreover, it will be interesting and beneficial to know how these dimensions relate to each other to define the size.

2.4.1 Dimension by Interface

One way to define granularity is by the perception of the service interface as made by its consumer. The various properties of a service interface responsible to define its size are listed below.

1. Functionality: It qualifies the amount of functionality offered by the service. The functionality can be either default functionality, which means some basic group of logic or operation provided in every case. Or the functionality can be parameterized and depending upon some values, it can be optionally provided. Depending upon the functionality volume, the service can be either fine-grained or course-grained than other service. Considering functionality criterion, A service offering basic CRUD functionality is fine-grained than a service which is offers some accumulated data using orchestration. [Hae+np]

2. Data: It refers to the amount of data handled or exchanged by the service. The data granularity can be of two types. The first one is input data granularity, which is the amount of data consumed or needed by the service in order to accomplish its tasks. And the other one is ouput data granularity, which is the amount of data returned by the service to its consumer. Depending upon the size and quantity of business object/objects consumed or returned by the service interface, it can be coarse or fine grained. Additionally, if the business object consumed is composed of other objects rather than primitive types, then it is coarser-grained. For example: the endpoint "PUT customers/C1234" is coarse-grained than the endpoint "PUT customers/C1234/Addresses/default" because of the size of data object expected by the service interface. [Hae+np]

3. Business Value: According to [RKKnp], each service is associated with an intention or business goal and follows some strategy to achiee that goal. The extent or magnitude of the intention can be perceived as a metric to define granularity. A service can be either atomic or aggregate of other services which depends

upon the level of composition directly influenced by the extent of target business goal. An atomic service will have lower granularity than an aggregate in terms of business value. For example, sellProduct is coase-grained than acceptPayment, which is again coarse-grained than validateAccountNumber. [Hae+np]

2.4.2 Dimension by Interface Realization

The section 2.4.1 provides the aspect of granularity with respect to the perception of customer to interface. However, there can be different opinion regarding the same properties, when it is viewed regarding the imlementation. This section takes the same aspects of granularity and try to analyze them when the services are implemented.

1. Functionality: In section 2.4.1, it was mentioned that an orchestration service has higher granularity than its constituent services with regard to default functionality. If the realization effort is focused, it may only include compositional and/or compensation logic because the individual tasks are acomplshed by the constituent service interfaces. Thus, in terms of the effort in realizing the service it is fine-grained than from the view point of interface by consumer. [Hae+np]

2. Data: In some cases, services may utilize standard message format. For example: financial services may use SWIFT for exchanging finanicial messages. These messages are extensible and are coarse grained in itself. However, all the data accepted by the service along with the message may not be required and used in order to fulfil the business goal of the service. In that sense, the service is coarse-grained from the viewpoint of consumer but is fine-grained in realization point of view. [Hae+np]

3. Business Value: It can make a huge impact when analyzing business value of service if the realization of the service is not considered well. For example: if we consider data management interface which supports storage, retrieval and transaction of data, it can be considered as fine-grained because it will not directly impact the business goals. However, since other services are very dependent in its performance, it becomes necessary to analyse the complexity associated with the implementation, reliablity and change the infrastructure if needed. These comes with additional costs, which should well be analyzed. From the realization point of view, the data service is coarse-grained than its view by consumer. [Hae+np]

Principle: A high Functionality granularity does not necessarily mean high business value granularity
It may seem to have direct proportional relationship between functionality and business value granularity. The business value of a service reflects business goals however the functionality refers to the amount of work done by the service. A service to show customer history can be considered to have high functionality granularity because of the involved time period and database query. However, it is of very low business value to the enterprise. [Hae+np]

2.4.3 R^3 Dimension

Keen [Kee91] and later Weill and Broadbent [WB98] introduced a separate group of criteria to measure granularity of service. The granularity was evaluated in terms of two dimensions as shown in the Figure 2.1

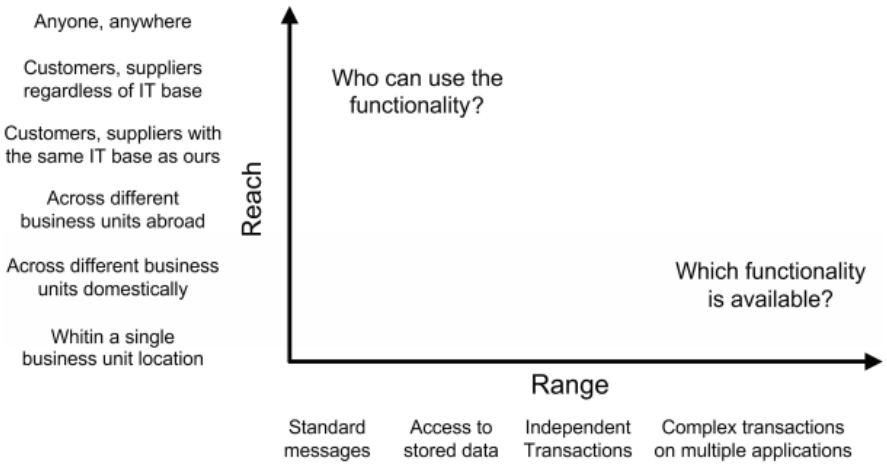

Figure 2.1: Reach and Range model from [Kee91; WB98]

Reach: It provides various levels to answer the question "Who can use the functionality? ". It provides the extent of consumers, who can access the functionality provide by the service. It can be a customer, the customers within an organization, supplier etc. as given by the Figure 2.1.
Range: It gives answer to "Which functionality is available? ". It shows the extent to which the information can be accessed from or shared with the service. The levels of

information accessed are analyzed depending upon the kind of business activities accessible from the service. It can be a simple data access, a transaction, message transfer etc as shown in the Figure 2.1 The 'Range' measures the amount of data exchanged in terms of the levels of business activities important for the organization. One example for such levels of activities with varying level of 'Range' is shown in Figure 2.2.

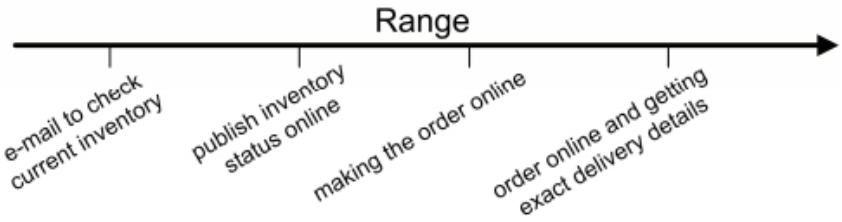

Figure 2.2: Example to show varying Range from [Kee91; WB98]

In the figure 2.2, the level of granularity increases as the functionality moves from 'accessing e-mail message' to 'publishing status online' and then to 'creating order'. It is due to the change in the amount of data access involved in each kind of functionality. Thus, the 'Range' directly depends upon the range of data access.
As the service grows, alongside 'Reach' and 'Range' also peaks up, which means the extent of consumers as well as the kind of functionality increase. This will add complexity in the service. The solution proposed by [Coc01] is to divide the architecture into services. However, only 'Reach' and 'Range' will not be enough to define the service. It will be equally important to determine scope of the individual services. The functionality of the service then should be define in two distinct dimensions 'which kind of functionality' and 'how much functionality'. This leads to another dimension of service as described below. [Kee91; WB98; RS07]
Realm: It tries to create a boundary around the scope of the functionality provided by the service and thus clarifies the ambiguity created by 'Range'. If we take the same example as given by Figure **??**, the range alone defining the kind of functionaly such as creating online order does not explicitly clarifies about what kind of order is under consideration. The order can be customer order or sales order. The specification of 'Realm' defining what kind of order plays role here. So, we can have two different services each with same 'Reach' and 'Range' however different 'Realm' for customer order and Sales order. [Kee91; WB98; RS07]

The consideration of all aspects of a service including 'Reach', 'Range' and 'Realm' give us a model to define granularity of a service and is called R^3 model. The volume in the R^3 space for a service gives its granularity. A coarse-grained service has higher R^3

volume then fine-grained service. The figures **??** and 2.4 show such volume-granularity analogy given by R^3 model. [Kee91; WB98; RS07]

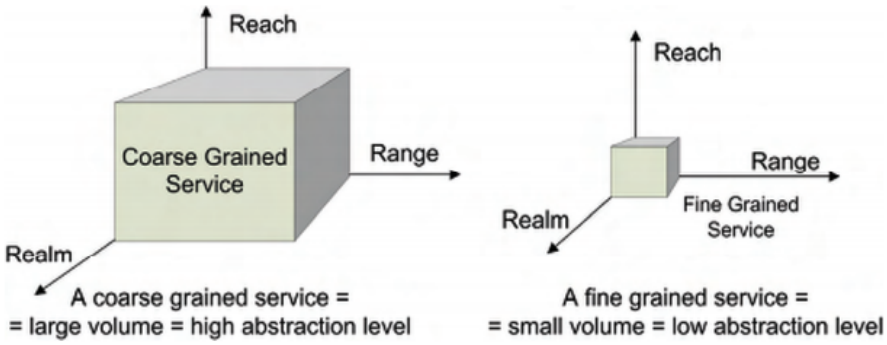

Figure 2.3: R^3 Volume-Granularity Analogy to show direct dependence of granularity and volume [RS07]

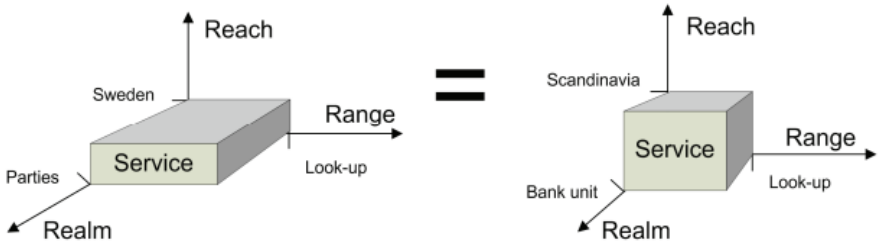

Figure 2.4: R^3 Volume-Granularity Analogy to show same granularity with different dimension along axes [RS07]

2.4.4 Retrospective

The sections 2.4.1 and 2.4.2 classified granularity of a service along three different directions: data, functionality and business value. Moreover, the interpretation from the viewpoint of interface by consumer and the viewpoint of producer for realization were also made. Again, the section 2.4.3 divides the aspect of granularity along Reach, Range and Realm space, the granularity given by the volume in the R^3 space. Despite of good explanation regarding each aspect, the classification along data, functionality and business value do not provide discrete metrics to define granularity in terms

of quantitatively. However, the given explanations and criteria are well rounded to compare two different services regarding granularity. These can be effectively used to classify if a service is fine-grained than other service. The R^3 model in section 2.4.3 additionally provides various levels of granularity along each axis. This makes it easy to at least measure the granularity and provides flexibility to compare the granularity between various services.
A case study [RS07] using data, functionality and business criteria to evaluate the impact of granularity on the complexity in the architecture is held. The study was done at KBC Bank Insurance Group of central Europe. Along the study, the impact of each kind of granularity is verified. The important results fromt the study are listed below.

1. A coarse-grained service in terms of 'Input Data' provides better transactional support, little communication overhead and also helps in scalability. However, there is a chance of data getting out-of-date quickly.

2. A coarse-grained service in terms of 'Output Data' also provides good communication efficiency and supports reusability.

3. A fine-grained service along 'Default Functionality' has high reusability and stability but low reuse efficiency.

4. A coarse-grained service due to more optional functionalities has high reuse efficiency, high reusability and also high stability but the implementation or realization is complicated.

5. A coarse-grained service along 'Business Value' has high consumer satisfaction value and emphasize the architecturally crucial points fulfilling business goals.

Similarly, a study was carried out in Handelsbanken in sweden, which has been working with Service-Oriented architecture. The purpose of the study was to analyze the impact of R^3 model on the architecture. It is observed that there are different categories of functionalities essential for an organization. Some functionality can have high Reach and Range with single functionality and other may have complex functionality with low Reach and Range. The categorization and realization of such functionalities should be accessed individually. It is not necessary to have same level of granularity across all services or there is no one right volume for all services. However, if there are many services with low volume, then we will need many services to accomplish a high level functionality. Additionally, it will increase interdependency among services and network complexity. Finally, the choice of granularity is completely dependent upon the IT-infrastructure of the company. The IT infrastructure decides which level of granularity it can support and how many of them. [RS07]

Principle: IT-infrastructure of an organization affects granularity.
The right value of granularity for an organization is highly influenced by its IT infrastructure. The organization should be capable of handling the complexities such as communication, runtime, infrastructure etc if they choose low granularity. [RS07]

Additionally, it is observed that a single service can be divided into number of granular services by dividing across either Reach, Range or Realm. The basic idea is to make the volume as low as possible as long as it can be supported by IT-infrastructure. Similarly, each dimension is not completely independent from other. For example: If the reach of a service has to be increased from domestic to global in an organization then the realm of the functionality has to be decreased in order to make the volume as low as possible, keep the development and runtime complexities in check. [RS07]

Principle: Keep the volume low if possible.
If supported by IT-infrastructure, it is recommended to keep the volume of service as low as possible, which can be achieved by managing the values of Reach, Range and Realm. It will help to decrease development and maintenance overload. [RS07]

%comment from Andrea

3 Quality of Service

3.1 Introduction

In addition to allign with the business requirements, an important goal of software engineering is to provide high quality. The quality assessment in the context of service oriented product becomes more crucial as the complexity of the system is getting higher by time. [ZL09; GL11; Nem+14] Quality models have been devised over time to evaluate quality of a software. A quality model is defined by quality attributes and quality metrics. Quality attributes are the expected properties of software artifacts defined by the chosen quality model. And, quality metrics give the techniques to measure the quality attributes. [Man+np] The software quality attributes can again be categorized into two types: internal and external attributes. [Man+np; BMB96] The internal quality attributes are the design time attributes which should be fulfilled during the design of the software. Some of the external quality attributes are loose coupling, cohesion, granularity, autonomy etc.[Ros+11; SSPnp; EM14] On the other hand, the external quality attributes are the traits of the software artifacts produced at the end of Software Development Life Cycle Some of them are reusability, maintainability etc. [EM14; Man+np; FL07; Feu13] For that reason, the external quality attributes can only be measure after the end of development. However, it has been evident that internal quality attributes have huge impact upon the value of external quality attributes and thus can be used to predict them. [HS90; Bri+np; AL03; Shi+08]The evaluation of both internal and external quality attributes are valuable in order to produce high quality software.[Man+np; PRF07; Per+07]

3.2 Quality Attributes

As already mentioned in section 3.1, the internal and external qualities determine the overall value of the service composed. There are different researches and studies which have been performed to find the features affecting the service qualities. Based on the published research papers, a comprehensive table 3.1 has been created. The table provides a minimum list of quality attributes which have been considerd in various research papers.

#	Attribute	[SSPnp]	[Xianp]	[AZR11]	[Shi+08]	[Ma+09]	[FL07]
1	Coupling	✓	✓	✓	✓	✓	✓
2	Cohesion	✓	X	✓	✓	✓	✓
3	Autonomy	✓	X	X	X	X	X
4	Granularity	✓	✓	✓	✓	✓	✓
5	Reusability	✓	X	X	✓	X	✓
6	Abstraction	✓	X	X	X	X	X
7	Complexity	X	X	X	✓	X	X

Table 3.1: Quality Attributes

The table 3.1 gives a picture of the studies done so far around service quality attributes. It can be deduced that most of the papers focus on coupling and granularity of the service and only few of them focus on other attributes such as complexity and autonomy.

Coupling refers to the dependency and interaction among services. The interaction becomes inevitable when a service requires a functionality provided by another service in order to accomplish its own goals. Similarly, Cohesion of any system is the extent of appropriate arrangement of elements to perform the functionalities. It affects the degree of understandability and sensibility of its interface for its consumers. Whereas, the complexity attribute provides the way to evaluate the difficulty in understanding and reasoning the context of the service or component.[EM14]
The reusability attribute for a service measures the degree to which it can be used by multiple consumer services and can undergo multiple composition to accoplish various high order functionalities. [FL07]
Autonomy is a broad term which refers to the ability of a service for self-containment, self-controlling, self-governance. Autonomy defines the boundary and context of the service. [MLS07] Finally, granularity is described in chapter 2 in detail. The basic signifance of granularity is the diversity and size of the functionalities offered by the service. [EM14]

3.3 Quality Metrics

There have been many studies made to define metrics for quality components.A major portion of the research have been done for object oriented and component based development. These metrics are refined to be used in service oriented systems.[Xianp; SSPnp] Firslty, various papers related to quality metrics of service oriented systems are collected. The papers which conducted their findings or proposition based on quality

amount of scientific studies and have produced convincing result are only selected. Additionally, the evaluation presented in the research papers were only done using the case studies of their own and not real life scenarios. Furthermore, they have used diversity of equations to define the quality metrics. On the basis of case studies made in the papers only, a fair comparision and choice of only one way to evaluate the quality metrics cannot be made. This is further added by the fact that the few papers cover most of the quality metrics, most of them focus on only few quality metrics such as coupling but not all of them focus on all the quality metrics. Nevertheless, they do not discuss about the relationship between all quality metrics. [EM14] This creates difficulty to map all the quality metrics into a common calculation family.
With such situation, this section will focus on facilitating the understanding of the metrics. Despite the case that there is no idea of threshold value of metrics discussed to define the optimal quality level, the discussed method can still be used to evaualte the quality along various metrics and compare the various design artifacts. This will definitely help to choose the optimum design based on the criteria. Also, there is complexity in understanding and confusion to follow a single proposed metrics procudure. But, the existing procedures can be broken down further to the simple understandable terms. By doing so, the basic terms which are the driving factors for the quality attributes and at the same time followed by most of the procedures as defined in the papers, can be identified.
The remaining part of this section will attempt to collaborate the metrics definitions proposed in various papers, analyze them to identify their conceptual base of measurement in simplest form.

3.3.1 Context and Notations

Before looking into the definitions of metrics, it will help understanding, to know related terms, assumptions and their respective notations.

- the business domain is realized with various processes defined as $P = \{p_1, p_2...p_s\}$
- a set of services realizing the application is defined as $S = \{s_1, s_2...s_s\}$; s_s is the total number of services in the application
- for any service $s \epsilon S$, the set of operations provided by s is given by $O(s) = \{o_1, o_2, ...o_o\}$ and $|O(s)|$ = O
- if an operation $o \epsilon O(s)$ has a set of input and output messages given by M(o). Then the set of messages of all operations of the service is given by $M(s) = \cup_{o \epsilon O(s)} M(o)$

- the consumers of the service $s \epsilon S$ is given by $S_{consumer}(s) = \{S_{c1}, S_2, ...Sn_c\}$; n_c gives the number of consumer services
- the producers for the service or the number of services to which the given service $s \epsilon S$ is dependent upon is given by $S_{producer}(s) = \{S_{c1}, S_2, ...Sn_p\}$; n_p gives the number of producer services

3.3.2 Coupling Metrics

[SSPnp] defines following metrics to determine coupling

$$SOCI(s) = |\{o_i \epsilon s_i : \exists_{o \epsilon s} calls(o, o_i) \wedge s \neq s_i\}|$$

$$ISCI(s) = |\{s_i : \exists_{o \epsilon s}, \exists_{o_i \epsilon s_i} calls(o, o_i) \wedge s \neq s_i\}|$$

$$SMCI(s) = |\cup M(o_i) : (o_i \epsilon s_i) \vee (\exists_{o \epsilon s}, \exists_{o_i \epsilon s_i} calls(o, o_i) \wedge s \neq s_i)|$$

where,

- $call(o, o_i)$ represents the call made from service 'o' to service 'o_i'

[Xianp] defines

$$Coupling(S_i) = p \sum_{j=1}^{n_s} (-log(P_L(j)))$$

$$= \frac{1}{n} \sum_{j=1}^{n_s} (-log(P_L(j)))$$

where,

- n_s is the total number of services connected
- 'n' is the total number of services in the entire application
- $p = \frac{1}{n}$ gives the probability of a service participating in any connection

[Kaz+11] defines

$$Coupling = \frac{\sum_{i \epsilon D_s} \sum_{k \epsilon O(s)} CCO(i, k)}{|O(s)|.K}$$

where,

- $D_s = \{D_1, D_2, ...D_k\}$ is set of dependencies the service has on other services
- $k = |D_s|$
- CCO is Conceptual Coupling betweeen service operations and obtained from CRUD matrix table constructed with business entities and business capabilities

[Shi+08] defines

$$coupling(s) = \frac{n_c + n_p}{n_s}$$

[AZR11] defines

$$coupling = \frac{\sum_{i=1}^{|O(s)|}(S_{i,sync} + S_{i,async})}{|S|}$$

where,

- $S_{i,sync}$ is the synchronous invocation in the operation O_i
- $S_{i,async}$ is the asynchronous invocation in the operation O_i

The table 3.2 shows simpler form of metrics proposed for coupling. Although, the table shows different way of evaluating coupling, they somehow agree to the basic metrics to calculate coupling. It can be deduced that the metrics to evaluate coupling uses basic metrics such as number of operations, number of provider services and number of messages.

3.3.3 Cohesion Metrics

[SSPnp] defines following metric for cohesion.

$$SFCI(s) = \frac{max(\mu(m))}{|O(s)|}$$

where,

- the number of operations using a message 'm' is $\mu(m)$ such that $m \epsilon M(s)$ and |O(s)| > 0
  The service is considered highly cohesive if all the operations use one common message. This implies that a higly cohesive service operates on a small set of key business entities as guided by the messages and are relavant to the service and all the operations operate on the same set of key business entities. [SSPnp]

#	Papers	Metrics Definition
1	[SSPnp]	SOCI : number of operation of other services invoked by the given service ISCI : number of services invoked by a given service SMCI : total number of messages from information model required by the operations
2	[Xianp]	Coupling is evaluated as information entropy of a complex service component where entropy is calculated using various data such as total number of atomic components connected, total number of links to atomic components and total number of atomic components
3	[Kaz+11]	The coupling is measured by using the dependency of a service operations with operations of other services. Additionally, the dependency is considered to have different weight value for each kind of operation such as Create, Read, Update, Delete (CRUD) and based on the type of business entity involved. The weight is referred from the CRUD matrix constructed in the process.
4	[Shi+08]	given by the average number of directly connected services
5	[AZR11]	defines coupling as the average number of synchronous and asynchronous invocations in all the operations of the service

Table 3.2: Coupling Metrics

[Shi+08] defines

$$cohesion = \frac{n_s}{|M(s)|}$$

[PRF07] defines

$$SIDC = \frac{n_{op}}{n_{pt}}$$

$$SIUC = \frac{n_{oc}}{n_c * |O(s)|}$$

$$SIUC = \frac{n_{so}}{n_c * |O(s)|}$$

where,

- n_{op} is the number of operations in the service which share the same parameter types
- n_{pt} is the total number of distinct parameter types in the service
- n_{oc} is the total number of operations in the service used by consumers
- n_{so} is the total number of sequentially accessed operations by the clients of the service

[AZR11] defines

$$cohesion = \frac{max(\mu(OFG, ODG))}{|O(s)|}$$

where,

- OFG and ODG are Operation Gunctionality Granularity and Operation Data Granularity respectively as calculated in section 3.3.4
- $\mu(OFG, ODG)$ gives the number of operations with specific value of OFG and ODG

The table 3.3 gives simplified view for the cohesion metrics. The papers presented agree on some basic metrics such as similarity of the operation usage behavior, commanility in the messages consumed by operations and similarity in the size of operations.

3.3.4 Granularity Metrics

[SSPnp] defines

$$SCG = |O(s)|$$

$$SDG = |M(s)|$$

[Shi+08] defines

$$granularity = \frac{|O(s)|^2}{|M(s)|^2}$$

[AZR11] defines

$$ODG = 1 - \left(\frac{\sum_{i=1}^{n_i o} W_{pi}}{\sum_{i=1}^{n_i s} W_{pi}} + \frac{\sum_{j=1}^{n_j o} W_{pj}}{\sum_{j=1}^{n_j s} W_{pj}} \right)$$

#	Papers	Metrics Definition
1	[SSPnp]	defines SFCI which measures the fraction of operations using similar messages out of total number of operations in the service operations of the service
2	[PRF07]	SIDC : defines cohesiveness as the fraction of operations based on commonality of the messages they operate on SIUC : defines the degree of consumption pattern of the service operations which is based on the similarity of consumers of the operations SIUC : defines the cohesion based on the sequential consumption behavior of more than one operations of a service by other services
3	[Shi+08]	cohesion is given by the inverse of average number of consumed messages by a service
4	[AZR11]	cohesion is defined as the consensus among the operations of the service regarding the functionality and data granularity which represents the type of parameters and the operations importance as calculated in the section 3.3.4

Table 3.3: Cohesion Metrics

$$OFG = \frac{W_i(o)}{\sum_{i=1} |O(S)| W_i(o)}$$

$$SOG = \sum_{i=1}^{|O(S)|} ODG(i) * OFG(i)$$

where,

- W_{pi} is the weight of input parameter
- W_{pj} is the weight of output parameter
- n_io is the number of input parameters in an operation
- n_jo is the number of output parameters in an operation
- n_is is the total number of input parameters in the service
- n_js is the total number of output parameters in the service

- $W_i(o)$ is the weight of an operation of the service
- $|O(S)|$ is the number operations in the service

#	Papers	Metrics Definition
1	[SSPnp]	the service capability granularity is given by the number of operations in a service and the data granularity by the number of messages consumed by the operations of the service
2	[AZR11]	ODG : evaluated in terms of number of input and output parameters and their type such as simple, user-defined and complex OFG : defined as the level of logic provided by the operations of the services SOG : defined by the sum of product of data granularity and functionality granularity for all operations in the service
3	[Shi+08]	evaluated as the ratio of squared number of operations of the service to the squared number of messages consumed by the service

Table 3.4: Granularity Metrics

The table 3.4 presents various view of granularity metrics based on different research papers. The chapter 2 discuss in detail regarding the topic. Based on the table 3.4, the granularity is evaluated using some basic metrics such as the number and type of the parameters, number of operations and number of messages consumed.

3.3.5 Complexity Metrics

[ZL09] defines

$$RCS = \frac{CS(s_i)}{|S|}$$

$$RIS = \frac{IS(s_i)}{|S|}$$

where,

- $CS(s_i)$ gives coupling value of a service

- $IS(s_i)$ gives importance weight of a service in an application
- |S| is the number of services to realize the application

The complexity is rather given by relative coupling than by coupling on its own. A low value of RCS indicates that the coupling is lower than the count of services where as RCS with value 1 indicates that the coupling is equal to the number of services. This represents high amount of complexity.
Similarly, a high value of RIS indicates that a lot of services are dependent upon the service and the service of critical value for them. This increases complexity as any changes or problem in the service affects a large number of services to a high extent.
[AZR11] defines

$$Complexity(s) = \frac{\sum_{i=1}^{|O(s)|}(SOG(i))^2}{|S|}$$

where,

- |S| is the number of services to realize the application
- SOG(i) gives the granularity of ith service calculated as described in section 3.3.4

refer to the document for effect of RCS and RIS on complixity

#	Papers	Metrics Definition
1	[ZL09]	RCS : complexity given by the degree of coupling for a service and evaluated as the fraction of its coupling to the total number of services RIS : measured as the fraction of total dependency weight of consumers upon the service to the total number of services
2	[AZR11]	the complexity is calculated using the granunarity of service operations

Table 3.5: Complexity Metrics

The table 3.5 provides the way to interprete complexity metrics. The complexity is highly dependent upong coupling and functionality granularity of the service.

3.3.6 Autonomy Metrics

[Ros+11] defines

$$SLC = \frac{1}{h_2 - l_2 + 1} \sum_{i=l_2}^{h_2} \sum_{sr \epsilon SR} (BE_{i,sr} XV_{sr})$$

$$DEP = \frac{\sum_{j=L1_i}^{h1_i} \sum_{k=1}^{BE} V_{sr_{jk}} - \sum_{j=L1_i}^{h1_i} \sum_{k=L2_i}^{j2_i} V_{sr_{jk}}}{nc}$$

$$\text{autonomy} = \begin{cases} SLC - DEP & \text{if SLC > DEP} \\ 0 & \text{otherwise} \end{cases}$$

where,

- nc is the number of relations with other services
- $V_{sr_{jk}}$ is the corresponding value of the action in jkth element of CRUD matrix, it gives the weight of corresponding business capability affecting a business entity
- $l1_i, h1_i, l2_i, h2_i$ are bounding indices in CRUD matrix of ith service

#	Papers	Metrics Definition
1	[Ros+11]	SLC : defined as the degree of control of a service upon its operations to act on its Business entities only DEP : given by the degree of coupling of the given service with other services autonomy is given by the difference of SLC and DEP if SLC > DEP else it is taken as 0

Table 3.6: Autonomy Metrics

The table 3.6 shows a way to interprete autonomy. It is calculated by the difference SLC-DEP when SLC is greater than DEP. In other cases it is taken as zero. So, autonomy increases as the operations of the services have full control upon its business entities but decreases if the service is dependent upon other services.

3.3.7 Reusability Metrics

[SSPnp] defines

$$SRI(s) = P + Q$$

where,

$$P = |\{s_i : \exists_{o \epsilon s}, \exists_{o_i \epsilon s_i} calls(o_i, o) \wedge s \neq s_i\}|$$

and

$$Q = |\{p \epsilon P : s \epsilon p\}|$$

[Shi+08] defines

$$Reusability = \frac{Cohesion - granularity + Consumability - coupling}{2}$$

#	Papers	Metrics Definition
1	[SSPnp]	SRI defines reusability as the number of existing consumers of the service
2	[Shi+08]	evaluated from coupling, cohesion, granularity and consumability of a service where consumability is the chance of the service being discovered and depends upon the fraction of operations in the service

Table 3.7: Reusability Metrics

The table 3.7 shows the reusability metrics evaluation. Reusability depends upon coupling, cohesion and granularity. It decreases as coupling and granularity increases.

3.4 Basic Quality Metrics

The section 3.3 presents various metrics to evaluate quality attributes based upon different papers. Additionally, the section analyzed the metrics and tried to derive them in simplest form possible. Eventually, the tables demonstrating the simplest definition of the metrics shows that the different quality attributes are measured on the basis of some basic metrics. Based on the section 3.3 and based on the papers, this section will attempt to derive basic metrics.

#	Metrics	Coupling	Cohesion	Granularity	Complexity	Autonomy	Reusability
1	number of service operations invoked by the service	+	X	X	+	-	-
2	number of operation using similar messages	X	+	X	X	X	+
3	number of operation used by same consumer	X	+	X	X	X	+
4	number of operation using similar parameters	X	+	X	X	X	+
5	number of operation with similar scope or capability	X	+	X	X	X	+
6	scope of operation	X	X	+	+	X	-
7	number of operations	X	X	+	+	X	-
8	number of parameters in operation	X	X	+	+	X	-
9	type of parameters in operation	X	X	+	+	X	-
10	number and size of messages used by operations	+	X	+	+	-	-
11	type of messages used by operations	X	+	X	X	X	+
12	number of consumer services	+	X	X	+	-	+
13	number of producer services	+	X	X	+	-	-
14	type of operation and business entity invoked by the service	+	X	X	+	-	-
15	number of consumers accessing same operation	X	+	X	X	X	+
16	number of consumer with similar operation usage sequence	X	+	X	X	X	+
17	dependency degree or imporance of service operation to other service	X	X	X	+	X	X
18	degree of control of operation to its business entities	X	X	X	X	+	X

Table 3.8: Basic Quality Metrics

Moreover, the effect of the basic metrics upon various quality attributes are also evaluated. Here, the meaning of the various symbols to demonstrate the affect are as given below:

- + the basic metric affect the quality attribute proportionlly
- - the basic metric inversely affect the quality attribute
- X there is no evidence found regarding the relationship from the papers

3.5 Principles defined by Quality Attributes

The section 3.1 has already mentioned that there are two distinct kind of quality attributes: external and internal. The external quality attributes cannot be evaluated during design however can be predicted using the internal quality attributes. This kind of relationship can be used to design service with good quality. Moreover, it is important to identify how each internal attributes affect the external attributes. The understanding of such relationship will be helpful to identify the combined effect of

the quality attributes and eventually to identify the service with the appropriate level of quality as per the requirement. The remaining part of the section lists relationship existing in various quality attributes as well as the desired value of the quality attributes.

1. When a service has large number of operations, it means it has large number of consumers. It will highly affect maintainability because a small change in the operations of the service will be propagated to large number of consumers. But again, if the service is too fine granular, there will be high network coupling. [FL07; Xianp][BKM07]

2. Low coupling improves understandability, reusability and scalability where as high coupling decreases maintainability. [Kaz+11][Erl05][Jos07]

3. A good strategy for grouping operations to realize a service is to combine the ones those are used together. This indicates that most of the operations are used by same client. It highly improves maintainability by limiting the number of affected consumer in the event of any change in the service. [Xianp]

4. A high cohesive quality attribute defines a good service. The service is easy to understand, test, change and is more stable. These properties highly support maintainability. enumerate[np01]

5. A service with operations those are cohesive and have low coupling which make the service reusable. [WYF03][FL07][Ma+09]

6. Services must be selected in a way so that they focus on a single business functionality. This highly follows the concept of low coupling. [PRF07][SSPnp]

7. Maintainability is divided into four distinct concepts: analyzability, changeability, stability and testability.[np01] A highly cohesive and low-coupled design should be easy to analyze and test. Moreover, such a system will be stable and easy to change.[PRF07]

8. The complexity of a service is determined by granularity. A coarse-grained service has higher complexity. However, as the size of the service decreases, the complexity of the over system governance also increases. [AZR11]

9. The complexity depends upon coupling. The complexity of a service is defined in terms of the number of dependencies of a service, number of operations as well as number and size of messages used by the service operations.[AZR11][SSPnp][Lee+01]

10. The selection of an appropriate service has to deal with multi-objective optimization problem. The quality attributes are not independent in all aspects. Depending upon goals of the architecture, tradeoffs have to be made for mutually exclusive attributes. For example, when choosing between coarse-grained and fine-grained service, various factors such as governance, high network roundtrip etc. also should be considered. [JSM08]

11. Business entity convergence of a service, which is the degree of control over specific business entities, is an important quality for the selection of service. For example: It is be better to create a single service to create and update an order. In that way change and control over the business entity is localized to a single service.[Ma+09]

12. Increasing the granularity decreases cohesion but also decreases the amount of message flow across the network. This is because, as the boundary of the service increases, more communication is handled within the service boundary. However, this can be true again only if highly related operations are merged to build the service. This suggests for the optimum granularity by handling cohesion and coupling in an appropriate level.[Ma+09][BKM07]

13. As the scope of the functionality provided by the service increases, the reusability decreases. [FL07]

14. If the service has high interface granularity, the operations operates on coarse-grained messages. This can affect the service performance. On the other hand, if the service has too fine-grained interface, then there will be a lot of messages required for the same task, which then can again affect the overall performance. [BKM07]

3.6 Relationship among Quality Attributes

In order to determine the appropriate level of quality for a service, it is also important to know the relationship between the quality attributes. This knowledge will helpfull to decide tradeoffs among them in the situation when it is not possible to achieve best of all. Based on the section 3.2 and section 3.5, the identified relationship among the quality attributes are shown in the table 3.9.

Here, the meaning of the various symbols showing nature of relationship are as given below:

#	Quality Attributes	Coupling	Cohesion	Granularity
1	Coupling		-	+
2	Cohesion	-		X
3	Granularity	+	X	
4	Complexity	+	X	+
5	Reusability	-	+	-
6	Autonomy	-	X	X
7	Maintainability	-	+	-

Table 3.9: Relationship among quality attributes

- + the quality attribute on the column affects the quality attribute on the corresponding row positively
- - the quality attribute on the column affects the quality attribute on the corresponding row inversely
- X there is no enough evidence found regarding the relationship from the papers

Acronyms

CRUD create, read, update, delete.

DEP Dependency.

IFBS International Financial and Brokerage Services.

ISCI Inter Service Coupling Index.

ODC Operation Data Granularity.

RCS Relative Coupling of Services.

RIS Relative Importance of Services.

SDLC Software Development Life Cycle.

SFCI Service Functional Cohesion Index.

SIDC Service Interface Data Cohesion.

SIUC Service Interface Usage Cohesion.

SLC Self Containment.

SMCI Service Message Coupling Index.

SOAF Service Oriented Architecture Framework.

SOCI Service Operational Coupling Index.

SRI Service Reuse Index.

SWIFT Society for Worldwide Interbank Financial Telecommunication.

List of Figures

List of Tables

Bibliography

[Abr14] S. Abram. *Microservices*. Oct. 2014. URL: http://www.javacodegeeks.com/2014/10/microservices.html.

[AL03] M. Alshayeb and W. Li. *An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes*. Tech. rep. IEEE Computer Society, 2003.

[Ann14] R. Annett. *What is a Monolith?* Nov. 2014.

[AZR11] S. Alahmari, E. Zaluska, and D. C. D. Roure. *A Metrics Framework for Evaluating SOA Service Granularity*. Tech. rep. School of Electronics and Computer Science University Southampton, 2011.

[BKM07] P. Bianco, R. Kotermanski, and P. F. Merson. *Evaluating a Service-Oriented Architecture*. Tech. rep. Carnegie Mellon University, 2007.

[BMB96] L. C. Briand, S. Morasca, and V. R. Basili. *Property-Based Software Engineering Measurement*. Tech. rep. IEEE Computer Society, 1996.

[Bri+np] L. C. Briand, J. Daly, V. Porter, and J. Wüst. *A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems*. Tech. rep. Fraunhofer IESE, np.

[Coc01] A. Cockburn. *WRITING EFFECTIVE USE CASES*. Tech. rep. Addison-Wesley, 2001.

[EAK06] A. Erradi, S. Anand, and N. Kulkarni. *SOAF: An Architectural Framework for Service Definition and Realization*. Tech. rep. University of New South Wales, Infosys Technologies Ltd, 2006.

[EM14] A. A. M. Elhag and R. Mohamad. *Metrics for Evaluating the Quality of Service-Oriented Design*. Tech. rep. Universiti Teknologi Malaysia, 2014.

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

[FA15] M. T. Fisher and M. L. Abbott. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise, Second Edition*. Addison-Wesley Professional, 2015.

[Feu13] G. Feuerlicht. *Evaluation of Quality of Design for Document-Centric Software Services*. Tech. rep. University of Economics and University of Technology, 2013.

[FL07] G. Feuerlicht and J. Lozina. *Understanding Service Reusability*. Tech. rep. University of Technology, 2007.

[FL14] M. Fowler and J. Lewis. *Microservices*. Mar. 2014. URL: `http://martinfowler.com/articles/microservices.html`.

[Foo05] D. Foody. *Getting web service granularity right*. 2005.

[GL11] A. Goeb and K. Lochmann. *A software quality model for SOA*. Tech. rep. Technische Universität München and SAP Research, 2011.

[Gup15] A. Gupta. *Microservices, Monoliths, and NoOps*. Mar. 2015.

[Hae+np] R. Haesen, M. Snoeck, W. Lemahieu, and S. Poelmans. *On the Definition of Service Granularity and Its Architectural Impact*. Tech. rep. Katholieke Universiteit Leuven, np.

[HS00] P. Herzum and O. Sims. *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley Sons, 2000.

[HS90] S. Henry and C. Selig. *Predicting Source=Code Complexity at the Design Stage*. Tech. rep. Virginia Polytechnic lnstirure, 1990.

[Jos07] N. M. Josuttis. *SOA in Practice The Art of Distributed System Design*. O'Reilly Media, 2007.

[JSM08] P. Jamshidi, M. Sharifi, and S. Mansour. *To Establish Enterprise Service Model from Enterprise Business Model*. Tech. rep. Amirkabir University of Technology, Iran University of Science, and Technology, 2008.

[Kaz+11] A. Kazemi, A. N. Azizkandi, A. Rostampour, H. Haghighi, P. Jamshidi, and F. Shams. *Measuring the Conceptual Coupling of Services Using Latent Semantic Indexing*. Tech. rep. Automated Software Engineering Research Group, 2011.

[Kee91] P. G. Keen. *Shaping The Future of Business Design Through Information Technology*. Harvard Business School Press, 1991.

[Lee+01] J. K. Lee, S. J. Jung, S. D. Kim, W. H. Jang, and D. H. Ham. *Component Identification Method with Coupling and Cohesion*. Tech. rep. Soongsil University and Software Quality Evaluation Center, 2001.

[Linnp] D. Linthicum. *Service Oriented Architecture (SOA)*. np.

[Ma+09] Q. Ma, N. Zhou, Y. Zhu, and H. Wang1. *Evaluating Service Identification with Design Metrics on Business Process Decomposition*. Tech. rep. IBM China Research Laboratory and IBM T.J. Watson Research Center, 2009.

[Mac14] L. MacVittie. *The Art of Scale: Microservices, The Scale Cube and Load Balancing*. Nov. 2014.

[Man+np] M. Mancioppi, M. Perepletchikov, C. Ryan, W.-J. van den Heuvel, and M. P. Papazoglou. *Towards a Quality Model for Choreography*. Tech. rep. European Research Institute in Services Science, Tilburg University, np.

[Mil+01] H. Mili, A. Mili, S. Yacoub, and E. Addy. *Reuse-based software engineering: techniques, organization, and controls*. Wiley-Interscience New York, 2001.

[MLS07] Y.-F. Ma, H. X. Li, and P. Sun. *A Lightweight Agent Fabric for Service Autonomy*. Tech. rep. IBM China Research Lab and Bei Hang University, 2007.

[Nav08] V. D. Naveen Kulkarni. *The Role of Service Granularity in A Successful SOA Realization – A Case Study*. Tech. rep. SETLabs, Infosys Technologies Ltd, 2008.

[Nem+14] H. Nematzadeh, H. Motameni, R. Mohamad, and Z. Nematzadeh. *QoS Measurement of Workflow-Based Web Service Compositions Using Colored Petri Net*. Tech. rep. Islamic Azad University Sari Branch and Universiti Teknologi Malaysia, 2014.

[New15] S. Newman. *Building Microservices*. O'Reilly Media, 2015.

[np01] np. *ISO/IEC 9126-1 Software Engineering Product Quality – Quality Model*. Tech. rep. International Standards Organization, 2001.

[NS14] D. Namiot and M. Sneps-Sneppe. *On Micro-services Architecture*. Tech. rep. Open Information Technologies Lab, Lomonosov Moscow State University, 2014.

[Pad04] F. Z. Padmal Vitharana Hemant Jain. *Strategy-Based Design of Reusable Business Components*. Tech. rep. IEEE, 2004.

[Per+07] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari. *Coupling Metrics for Predicting Maintainability in Service-Oriented Designs*. Tech. rep. RMIT University, 2007.

[PRF07] M. Perepletchikov, C. Ryan, and K. Frampton. *Cohesion Metrics for Predicting Maintainability of Service-Oriented Software*. Tech. rep. RMIT University, 2007.

[Ric14a] C. Richardson. *Microservices: Decomposing Applications for Deployability and Scalability*. May 2014.

[Ric14b] C. Richardson. *Pattern: Monolithic Architecture*. 2014. URL: `http://microservices.io/patterns/monolithic.html`.

[RKKnp] C. Rolland, R. S. Kaabi, and N. Kraiem. *On ISOA: Intentional Services Oriented Architecture*. Tech. rep. Université Paris, np.

[Ros+11] A. Rostampour, A. Kazemi, F. Shams, P. Jamshidi, and A. Azizkandi. *Measures of Structural Complexity and Service Autonomy*. Tech. rep. Shahid Beheshti University GC, 2011.

[RS07] P. Reldin and P. Sundling. *Explaining SOA Service Granularity– How IT-strategy shapes services*. Tech. rep. Linköping University, 2007.

[Shi+08] B. Shim, S. Choue, S. Kim, and S. Park. *A Design Quality Model for Service-Oriented Architecture*. Tech. rep. Sogang University, 2008.

[Sim05] O. Sims. *Developing the architectural framework for SOA - part 2-service granularity and dependency management. CBDI Forum Journal*. Tech. rep. CBDI, 2005.

[SSPnp] R. Sindhgatta, B. Sengupta, and K. Ponnalagu. *Measuring the Quality of Service Oriented Design*. Tech. rep. IBM India Research Laboratory, np.

[Ste06] C. Steghuis. *Service Granularity in SOA Projects: A Trade-off Analysis*. Tech. rep. University of Twente, 2006.

[WB98] P. Weill and M. Broadbent. *Leveraging The New Infrastructure: How Market Leaders Capitalize on Information Technology*. Harvard Business School Press, 1998.

[WV04] L. Wilkes and R. Veryard. "Service-Oriented Architecture: Considerations for Agile Systems." In: *np* (2004).

[WXZ05] Z. Wang, X. Xu, and D. Zhan. *A Survey of Business Component Identification Methods and Related Techniques*. Tech. rep. International Journal of Information Technology, 2005.

[WYF03] H. Washizakia, H. Yamamoto, and Y. Fukazawa. *A metrics suite for measuring reusability of software components*. Tech. rep. Waseda University, 2003.

[Xianp] W. Xiao-jun. *Metrics for Evaluating Coupling and Service Granularity in Service Oriented Architecture*. Tech. rep. Nanjing University of Posts and Telecommunications, np.

[ZL09] Q. Zhang and X. Li. *Complexity Metrics for Service-Oriented Systems*. Tech. rep. Hefei University of Technology, 2009.